PROYECTO DE
FIN DE CARRERA

# Utilización de lenguaje C para diseño de Hardware

**DANIEL GÓMEZ – GABRIEL HÉGUY - MATÍAS PUIG**

**PROYECTO DE FIN DE CARRERA**
**INSTITUTO DE INGENIERÍA ELÉCTRICA**
**FACULTAD DE INGENIERÍA**
**UNIVERSIDAD DE LA REPÚBLICA**

**Marzo 2003**

## TUTOR:

Ing. Juan Pablo Oliver (IIE - Facultad de Ingeniería)

## ESTUDIANTES:

Matías Puig

Gabriel Héguy

Daniel Gómez

## AGRADECIMIENTOS:

A nuestro tutor Juan Pablo Oliver.

A Rosario Curbelo por sus aportes en la confección del presente informe.

A Carlos Héguy y Rosa Squadroni.

A Florencia Roselli.

A Guillermo Laborde.

A todos los que de alguna forma colaboraron con el desarrollo de este proyecto.

# Resumen

Este proyecto estudia la utilización de lenguaje C en el diseño de aplicaciones en hardware, en particular sobre lógica reprogramable. Se evalúan distintas herramientas de conversión de C a algún lenguaje del tipo HDL.

En la primer etapa del proyecto se realizó una búsqueda de información sobre herramientas de este tipo y de las encontradas se seleccionaron dos: Handel C y Transmogrifier C.

La segunda etapa fue de familiarización con las herramientas, investigando los circuitos generados a partir de diseños escritos en C.

En la tercer etapa se analizaron distintos ejemplos para ambas herramientas como ser: algoritmos de división y raíz cuadrada, un árbitro de bus, ejemplo de acceso a memorias externas y de uso de memorias internas, un multiplicador con pipeline y ejemplos de operaciones con matrices. En todos estos ejemplos se relevaron los recursos ocupados y la frecuencia máxima alcanzada para chips de la familia Altera y en algunos casos también se hizo para chips de Xilinx. Además se hicieron comparaciones en algunos ejemplos con versiones escritas en VHDL.

Finalmente se eligió una de las herramientas (Handel C) y un algoritmo escrito en C para analizar la facilidad de traducir éste de la forma más directa posible para su implementación en hardware. El diseño seleccionado fue un filtro de imágenes en tonos de grises que se implementó en la placa ARC-PCI de Altera. Además se realizaron comparaciones de performance entre este diseño y el filtro original (en C) corriendo en diferentes PC.

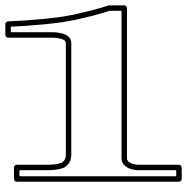
# Contenido

# Figuras

# Tablas

# Siglas y abreviaturas

| | |
|---|---|
| ARC-PCI | Altera Reconfigurable Computer with PCI interface |
| ASCII | American Standard Code for Information Interchange |
| ASIC | Application Specific Integrated Circuit |
| CLB | Combinatorial Logic Block |
| DFF | Flip-Flop tipo D |
| DMA | Direct Memory Access |
| DOS | Disk Operative System |
| DPMI | DOS Protected Mode Interface |
| EAB | Embedded Array Block |
| EDIF | Electronic Design Interchange Format |
| EPROM | Erasable Programmable Read Only Memory |
| FF | Flip-flop |
| FPGA | Field Programmable Gate Array |
| GNU | Hace referencia a software de libre distribución |
| HCC | Handel C Compiler |
| HDL | Hardware Description Language |
| IEEE | Institute of Electrical and Electronics Engineers |
| IIE | Instituto de Ingeniería Eléctrica |
| IOB | Input Output Block |
| IP | Intellectual Property |
| LC | Logic Cell |
| LE | Logic Element |
| LINUX | Versión libre de UNIX |
| LSB | Least Significant Bit |
| LUT | Look Up Table |
| M+PII | Max+Plus II |
| MSB | Most significant bit |
| OSCI | Open System C Initiative |
| PC | Personal Computer |
| PCI | Peripheral Component Interconnect |
| PGM | Portable Gray Map |
| PLD | Programmable Logic Device |
| RAM | Random Access Memory |
| ROM | Read Only Memory |
| RTL | Register Transfer Level |
| SRAM | Static RAM |
| TMCC | Transmogrifier C Compiler |
| UNIX | Avanzado sistema operativo multi-usuario. |
| VHDL | Very high speed integrated circuit Hardware Description Language |
| XNF | Xilinx Netlist Format |



# Introducción

## 1.1 Motivación

A medida que la complejidad de los sistemas se incrementa y el tiempo destinado en los proyectos al diseño se acorta, se vuelve extremadamente importante que las especificaciones del sistema sean capturadas de forma que conduzcan a interpretaciones sin ambigüedades por parte de los ingenieros encargados de las implementaciones. La forma más común de captura de las especificaciones es un documento escrito en lenguaje natural, esto tiene varias desventajas:

- el lenguaje es ambiguo y abierto a las variantes en su interpretación
- la especificación puede estar incompleta y ser inconsistente
- no hay forma de verificar la corrección de dicha especificación.

Estas desventajas han llevado a muchos diseñadores de sistemas, hardware y software, a crear especificaciones ejecutables para sus sistemas. Éstas son modelos funcionales escritos en un lenguaje como C o C++, un modelo funcional es esencialmente un programa que cuando se ejecuta exhibe el mismo comportamiento que el sistema.

Si pensamos en el proceso de diseño, luego de obtener el modelo funcional hay que pasar a la codificación del hardware, definiendo su comportamiento según la función especificada.

La misma complejidad de los circuitos obliga a hacer bancos de prueba cada vez más sofisticados. La propia validación de la funcionalidad se vuelve una tarea compleja debido al gran número de modos de funcionamiento de los circuitos y de requerimientos a los que se les somete. El tiempo de simulación se convierte entonces en uno de los principales problemas para el diseñador, que tiene que esperar a que terminen largas simulaciones para analizar los resultados.

De lo mencionado anteriormente surge la pregunta ¿por que no aprovechar que el modelo funcional desarrollado en C / C++ cumple con las especificaciones y validaciones necesarias, y traducir automáticamente este código a una implementación en hardware?. La traducción conviene que sea en forma automática y no en la forma tradicional, que es realizar el diseño completo del mismo basándose en la experiencia y destreza en la codificación en VHDL o Verilog del diseñador, pues ésta traducción consume un tiempo considerable y es fuente común de errores.

¿Que ventajas posee trabajar en C / C++?

Dichos lenguajes son elegidos por tres razones fundamentales:

- proveen el control y la abstracción de datos necesarios para desarrollar descripciones de sistemas compactas y eficientes
- muchos sistemas contienen tanto hardware como software y para el software uno de estos lenguajes es la elección natural
- los diseñadores están familiarizados con estos lenguajes y con el gran número de herramientas de desarrollo asociadas con ellos.

Un lenguaje orientado a objetos como C++ provee además la habilidad de extender el lenguaje mediante clases, sin agregar construcciones sintácticas nuevas. Una aproximación basada en clases para proveer construcciones de modelado es superior a un nuevo lenguaje propietario ya que permite a los diseñadores continuar utilizando las herramientas con que están familiarizados.

La posibilidad de compilar algoritmos directamente a hardware para su implementación en una FPGA, la facilidad de realizar cambios en las especificaciones originales y la creación de bancos de prueba hace muy atractiva la idea de usar herramientas que traduzcan automáticamente un programa escrito en C a un lenguaje de descripción de hardware (HDL).

## 1.2 Antecedentes

El instituto de Ingeniería Eléctrica de la Facultad de Ingeniería, ha venido trabajando en el área de Lógica Programable desde 1992, ha contado desde entonces con apoyo de la Comisión Sectorial de Investigación Científica de la Universidad de la República, el proyecto "Laboratorio de Microelectrónica" del Programa Conicyt/BID, la empresa Altera y la empresa Xilinx (líderes a nivel mundial en lógica programable) entre otros.

Se llevaron a cabo proyectos como:

- “Coprocesador neuronal basado en lógica programable” 1997-1998 (Conicyt/BID)
- “Realización de algoritmos de tratamiento de imágenes en hardware” 1999 (Fin de carrera)
- “Analizador Lógico de 100MHz utilizando FPGA” 1998 (IBERCHIP)
- "Algoritmos en Hardware Reconfigurable" 2000 (Fondo Clemente Estable)

En todas estas iniciativas se realizaron los diseños en forma tradicional, por tanto la temática de este proyecto, utilización de lenguaje C para diseño de hardware, es original y sin antecedentes en este instituto. Esta metodología permitiría un rápido prototipado de los diseños utilizando lógica reprogramable.

## 1.3 Objetivos

### 1.3.1 Objetivos generales

Este proyecto estudiará la utilización de lenguaje C con la finalidad de diseñar aplicaciones en hardware, en particular sobre lógica reprogramable. Se evaluarán distintas herramientas, que llamaremos compiladores, de conversión del C a un lenguaje del tipo HDL.

La utilización de estas herramientas de alto nivel (compiladores) hace que en general se pierda control estricto del hardware que se está generando, y que en principio se podría pensar que llevaría a diseños más costosos en área o más lentos en sus respuestas. Analizaremos comparativamente algunos ejemplos que codificaremos tanto en C como directamente en VHDL y estudiaremos los costos de ambas implementaciones, tanto en recursos del chip como en horas de diseño y simulación.

### 1.3.2 Objetivos específicos

- Estudio del estado del arte en herramientas de pasaje de software a hardware.
- Familiarización con las herramientas y selección de dos para su evaluación.
- Análisis de funcionamiento de las herramientas elegidas.
- Comparación de éstas utilizando ejemplos sencillos.
- Elección de una de ellas e implementación de un algoritmo más complejo.

El cumplimiento de estos objetivos se realizó en cuatro grandes etapas:

ETAPA 1 : Se buscó de información y selección de dos herramientas para análisis posterior.

ETAPA 2 : Se instalaron las herramientas y se procedió a familiarizarse con los compiladores y sus extensiones del lenguaje C.

ETAPA 3 : Se implementaron ejemplos clásicos tanto del tipo hardware como del tipo software y se realizaron estudios de performance y complejidad (desde el punto de vista del tamaño) comparándolos con diseños realizados utilizando VHDL. En particular se comparó número de celdas, frecuencia máxima de reloj, retardo en las salidas, existencia de glitches.

ETAPA 4 : Se implementó un algoritmo de tratamiento de imágenes (especificado inicialmente en C) seleccionado de acuerdo a los resultados obtenidos en la Etapa 2 y al hardware disponible en el IIE (Placa ARC-PCI de Altera).

## 1.4 Contenido del resto del documento

En el capítulo 2 se encuentra una breve reseña del estado del arte, los criterios de selección y los productos elegidos, Transmogrifier C y Handel C.

En el capítulo 3 se explica como se instalan dichos compiladores, la sintaxis del lenguaje y sus extensiones, así como también una descripción de los circuitos generados por éstos.

En el capítulo 4 se mencionan los problemas encontrados al comenzar a generar circuitos. Además allí se incluye una guía rápida de cómo compilar un programa en C para las herramientas evaluadas.

En el capítulo 5 se encuentran los ejemplos utilizados en la evaluación de los compiladores.

En el capítulo 6 se desarrolla una aplicación de mayor complejidad (filtro de imágenes) para Handel C.

Finalmente en el capítulo 7 se hallan las conclusiones del trabajo realizado y posibles líneas de desarrollo posteriores.

En los apéndices se encuentra un detalle de los programas correctores realizados, reseña de algoritmos sumadores y multiplicadores, librerías de VHDL, fuentes completas de los distintos ejemplos, una breve guía de compilación y material complementario como ser características de los chips utilizados y diagramas de las placas de prueba.

# 2 Selección de herramientas

En este capítulo presentaremos un cuadro comparativo de las posibles herramientas a evaluar. Discutiremos sobre los criterios utilizados en la selección del software y determinaremos en función de estos criterios dos herramientas para continuar el estudio. Para obtener información más detallada sobre las herramientas referirse al Apéndice D.

## 2.1 Estado del Arte

Nos concentramos en la búsqueda de información en Internet de compañías y universidades que estuvieran trabajando o investigando temas relacionados con nuestros objetivos.

Observamos que hay unas cuantas herramientas ofrecidas por distintos fabricantes aunque algunas de ellas se encuentran aún en versiones beta y no son liberadas al público en general. También encontramos universidades que habían trabajado en estos temas y luego comercializaron sus productos.

Posteriormente clasificamos y ordenamos el material encontrado basándonos fundamentalmente en la información comercial proporcionada por el fabricante sobre el funcionamiento y características de los programas pues para la mayoría de ellos no contábamos con ninguna versión de evaluación.

Las principales herramientas encontradas afín con el objetivo de nuestro proyecto son A|RT Builder, Sistemex basado en SuperLog, Cynthesizer, CoCentric SystemC Compiler, Transmogrifier C, Mini proyecto basado en SUIF, SiliconC, HandelC y DK1 Suite.

A continuación presentaremos una tabla comparativa de las principales características de los paquetes encontrados.

**Tabla 2-1** Cuadro comparativo de herramientas para la síntesis de C a hardware

| Paquete | Fabricante | Plataforma | Subset ANSI | Tipos especiales | Estado | Prog. Univ. | Precio |
|---|---|---|---|---|---|---|---|
| A\|RT Library | Frontier Design | HP-UX, Sun Solaris, Windows NT y Linux. | Amplio conjunto N/E. +SystemC | Punto Fijo | V | NO | FREE |
| A\|RT Designer | | | | | | | U$S 45.000 |
| A\|RT Builder | | | | | | | U$S 20.000 |
| Systemsim | Co-Design Automation | Sun Solaris y Linux. | Amplio conjunto. +SystemC | Punto Flotante Typedef | V | NO | U$S 18.000 |
| Systemex | | | | | | | U$S 25.000 |
| Cynlib | Forte Design Systems | N/E | N/E | N/E | V | NO | FREE |
| Cyn++ | | | | | | | |
| CynGDB | | | | | | | |
| | | | | | | | |
| Cynchronizer | | | | | | | U$S 40.800 |
| Cynthesizer | | | | | | | |
| Cynware | | | | | | | |
| Cyntax | | | | | | | |
| OCAPI-xl | IMEC | N/E | N/E | N/E | D | NO | --- |
| Cocentric System Studio | SYNOPSYS | HP-UX, Linux, and Solaris | SystemC | Punto Fijo Punto Flotante | V | SI | U$S 500 |
| CoCentric SystemC Compiler | | | | | | | U$S 2.500(*) |
| CoCentric Fixed-Point Designer | | | | | | | |
| Transmogrifier C | University of Toronto | Para compilar | Aceptable conjunto | --- | D | --- | FREE |
| Miniproyecto basado en SUIF | Instituto Indio de Tecnología | Para compilar | Amplio conjunto | Punto Flotante Punteros | PG | --- | FREE |
| Silicon C basado en SUIF | MIT | N/E | Amplio conjunto | --- | D | --- | FREE |
| Handel C | Oxford University | MS-DOS | Conjunto reducido | --- | D | --- | FREE |
| DK1 Design Suite | Celoxica | Windows NT, 2K, XP | Amplio conjunto | typedef | V | SI | U$S 300 |

(*) El precio corresponde a un amplio paquete de programas que incluye los mencionados en el cuadro.
V = Vigente
D = Discontinuado
PG = Proyecto de grado
N/E = No especificado.

## 2.2 Criterios de Selección

Los criterios para la selección de las distintas herramientas que surgieron de la búsqueda, son los siguientes:

- Precio de las licencias / años de validez
- Utilidad o funcionalidad
- Soporte técnico o estado del paquete
- Plataforma

Los criterios fueron considerados en el orden descrito. Cabe destacar la poca disponibilidad de las empresas a proporcionar información sobre los costos de sus herramientas, muchas de ellas no respondieron y por tanto nos basamos en información obtenida de terceros como revistas electrónicas de diseño, por ejemplo EE TIMES; otras ofrecían grandes conjuntos de herramientas de diseño, de las cuales solamente nos interesaba una de ellas, lo que lleva a que sus precios fueran considerablemente grandes. También encontramos resistencia al intentar obtener información sobre sus Programas Universitarios, en muchos casos no obtuvimos respuestas o estos programas eran destinados a universidades de la Comunidad Europea. Por otro lado encontramos fabricantes que ofrecían paquetes que estaban aún en versiones betas, nos ofrecimos como beta-testers pero sin resultados satisfactorios.

Pasemos a detallar los criterios mencionados:

**Precio de las licencias / años de validez**
Aquí como planteamos antes, los precios no fueron sencillos de obtener y para algunos casos no disponíamos de datos. Se entendió que una buena forma de comparar las distintas ofertas era considerar el precio para una licencia node locked dividido el número de años de vigencia de ella. Se trabajó con el precio de licencia node locked pues en general las licencias flotantes tienen mayor costo. Pero nuevamente recalcamos que este tipo de información la disponemos para algunos paquetes solamente.

**Utilidad o funcionalidad**
Aquí se tomó en cuenta como característica importante y relevante el subconjunto soportado del lenguaje C, tipos de datos soportados y el tipo de archivo de salida generado, sea VHDL, XNF, etc. En principio no resulta demasiado significativo el trabajo con números en punto flotante, pero si al menos con enteros con signo y todas sus operaciones. En lo que respecta a las instrucciones de control de flujo de programa, se entiende que en principio alcanza con soportar alguna instrucción de bifurcación y alguna de generación de bucles. Por último el tipo de archivo de salida debe ser tal que nos permita sintetizar y/o mapear el diseño para las familias de Altera y de Xilinix.

**Soporte técnico o estado del paquete**
Aquí tomamos en cuenta el tipo de soporte técnico ofrecido y la duración del mismo, acceso a ayudas on line, librerías, etc; así como también para el caso de herramientas que no siguieran desarrollándose en la actualidad su fecha de finalización.

**Plataforma**
Otro aspecto importante a considerar, si bien en la Faculta de Ingeniería, y en particular en el Instituto de Ingeniería Eléctrica, encontramos una amplia variedad de arquitecturas y sistemas operativos, éste no deja de ser un factor a tener en cuenta al momento de realizar la compra. Las arquitecturas y plataformas aceptadas son: PC (procesadores Intel x86 o similares) y SUN (procesadores microSPARC II), luego Windows (NT, ME, 2000, XP), Linux (FreeBSD, RedHat, etc) y Solaris 2.x.

En el planteo inicial del proyecto se consideraba contar con fondos para la compra de licencias cuyo costo fuera inferior a U$S 1.000 y que no tuvieran vencimiento, pero durante su desarrollo y en función de los datos obtenidos no fue posible lograr las dos condiciones por tanto las opciones se redujeron a paquetes gratuitos, con programas universitarios gratuitos o licencias de evaluación.

Entonces tomando en cuenta las consideraciones económicas, la lista de candidatos se redujo rápidamente a cuatro opciones las cuales con la aplicación del resto de los criterios se redujeron a dos que era lo deseado en los objetivos del proyecto.

A continuación presentamos la lista preliminar de las herramientas y luego en los siguientes puntos detallaremos las razones para su aceptación o rechazo. Ellas son:

- Suif to VHDL translator. (Miniproyecto basado sobre SUIF)
- DK1 Design Suite (Handel-C versión 3.1 de Celoxica)
- Handel C (versión 0.1 – Enero 97 )
- Transmogrifier C (revisión 3.1 – Enero 96 )

## 2.3 Paquetes Seleccionados

### 2.3.1 Handel C (versión 0.1 – Enero 97)

Es un producto gratuito, comenzado a desarrollarse en el año 1996 por la Universidad de Oxford. La versión que contamos para utilizar fue desarrollada en el año 1997, luego este producto fue comercializado por Embedded Solutions y actualmente por Celoxica.

Aunque posee un conjunto limitado de instrucciones del lenguaje C este es aceptable para nuestra evaluación pues permite construir cualquier algoritmo que no precise punto flotante, utilización de punteros, arrays multidimensionales o definición de estructuras.

De la compilación con el Handel C Compiler (o HCC) obtenemos como salida un archivo tipo netlist, que puede ser tanto VHDL del tipo estructural (architectural), EDIF o XNF. Esta implementación es realizada para las familias Xilinx 2000, 3000, 4000, 6000, pero como no utiliza ninguna característica particular de ellas, fácilmente con cualquier sintetizador se puede compilar para otras familias e inclusive para ASIC.

Estas razones nos hacen considerarlo un buen candidato para su evaluación más exhaustiva.

### 2.3.2 Transmogrifier C (revisión 3.1 – Enero 96)

Es también un producto gratuito y fue desarrollado por al Universidad de Toronto, la última versión liberada es la 3.1 del año 1996.

Este compilador corre bajo Unix (eventualmente puede compilarse para correr en ambientes Windows). Tiene un conjunto limitado de instrucciones de control de flujo, pero más operadores definidos que el Handel C, esto para nuestra evaluación es suficiente. Cabe destacar que el no tener implementado de antemano el producto de enteros hacen que la adaptación para su compilación de programas, que utilicen este operador, sea una tarea más dificultosa.

Genera como salida archivos tipo netlist descritos en formato XNF. Realiza implementaciones para la familias Xilinx 4000 y Flex 8000. En sus opciones por defecto genera archivos XNF bastante genéricos pues no utiliza ninguna característica particular de estas familias, solamente compuertas básicas como por ejemplo and, or, inversores, flip-flop, etc.

Las razones antes expuestas hacen que lo consideremos como un candidato a ser evaluado más en detalle.

## 2.4 Paquetes Descartados

### 2.4.1 Suif to VHDL translator. (Miniproyecto basado sobre SUIF)

En principio sería un candidato ideal pues soporta un amplio subconjunto del lenguaje C, es gratuito y genera archivos de salida en VHDL comportamentales ampliando el rango de arquitecturas de destino.
La razón fundamental para descartarlo es que los archivos generados no son sintetizables, Cómo se puede comprobar directamente de los ejemplos expuestos en el paper de presentación de este miniproyecto basado en Suif [1].

### 2.4.2 DK1 Design Suite (de Celoxica)

Se obtuvo una licencia universitaria gratuita para el paquete DK1 Suite, pero ésta solamente permitía la simulación de los archivos en C que se ingresaban y no se generaba por tanto una salida en VHDL u otro formato estándar. Además la simulación no aportaba información relevante sobre el circuito implementado (frecuencia máxima, recursos consumidos del chip, diagrama de tiempo de las señales, etc), se limitaba tan sólo a una simulación a nivel funcional.

# 3 Puesta en funcionamiento de las herramientas

En este capítulo hablaremos para cada paquete de cómo instalarlo, que opciones ofrece a la hora de compilar un diseño, describiremos la sintaxis del lenguaje C que acepta y finalmente detallaremos los circuitos que generan.

De aquí en más cuando hablemos de programas en Handel C o Transmogrifier C nos estaremos refiriendo a una descripción del circuito hecha en el correspondiente subconjunto de C.

## 3.1 Handel C (HCC versión 0.1 – Enero 97)

### 3.1.1 Instalación, opciones de compilación

Esta herramienta no requiere ningún tipo de instalación especial, la versión que disponemos ya está compilada para MS-DOS y puede correr también en cualquiera de las versiones de Windows. Es conveniente ajustar el valor de la variable de entorno PATH para que apunte al subdirectorio PC_BIN de ésta distribución que es donde se encuentra el ejecutable hcc.exe.

Veamos las distintas opciones ofrecidas por el compilador HCC [2] :

**Tabla 3-1** Opciones de línea de comando para el compilador Handel C

| | |
|---|---|
| -q \| --quiet | Quiet output |
| -v \| --verbose | Verbose output |
| -vv \| --very-verbose | Very verbose output |
| -v[0-9] | Set verbosity level explicitly |
| -cpp \| --c-preprocess | Run external C-preprocessor first |
| -nr \| --no-renaming | Turn off automatic renaming (aliasing) |
| -ast \| --add-statement-time | Add statement time information |
| -ass \| --add-statement-space | Add statement space information |
| -ald \| --add-logic-depth | Add combinational logic depth information |
| -nfd \| --no-force-delays | Don't force minimal delays in loops |
| -nwi \| --no-width-inference | Turn off constant width inferencer |
| -nhc \| --no-health-checks | Turn off source code health checks |
| -nsn \| --no-sanitise | Turn off source code "sanitisation" |
| -pp \| --pretty-print | Pretty print source program after parsing |
| -ppa \| --pretty-print-after | Pretty print source program after compiling |
| -O \| --optimise | Turn on all major optimisations |
| -O[0-9] | Set optimisation level explicitly |
| -s \| --simulate | Simulate after compilation |
| -ns \| --no-simulate | Halt after compilation |
| -ss \| --simulate-steps n | Set cycles between display for simulator |
| -su \| --simulate-until n | Simulate for n step |
| -se \| --simulate-extra n | Simulate for n steps after exhausting input |
| -sr \| --simulate-radix n | Set the base for display of simulator data |
| -sp \| --simulate-power | Estimate power consumption during simulation up down high low |
| -spw \| --simulate-power-weights | Set weights for power estimation |
| -stc \| --simulate-toggle-count | Simulate with toggle counting |
| -ir \| --input-ram rname fname | Set initial values of an eram from a file |
| -or \| --output-ram rname fname | Dump final values from an eram to a file |
| -if \| --input-file fname | Specify an input for simulation data |
| -of \| --output-file fname | Specify an output for simulation data |
| -? \| -h \| --version \| --help | Print this message |

-q | --quiet , -v | --verbose , -vv | --very-verbose , -v[0-9]

Estas opciones controlan la información desplegada al usuario por la salida estándar, STDOUT (en gral. pantalla), partiendo de una mínima información sobre errores de compilación hasta progresos en las distintas instancias de optimización.

-cpp | --c-preprocess

El HCC es capaz de invocar previamente a un proprocesador de C (conocido generalmente como cpp), pudiéndose utilizar algún paquete GNU [1] o los ofrecidos por Microsoft. También es conveniente ajustar el valor de la variable de entorno PATH para que apunte a donde se encuentra el ejecutable del preprocesador. Se genera un archivo temporal con la salida del cpp, se compila y luego se borran estos archivos auxiliares, en caso de no encontrarlo se tratará de compilar el programa tal cual se leyó.

[1] Puede utilizar los ofrecidos por Minimalist GNU For Windows en **http://www.mingw.org/** o también los disponibles en Cygwin en la dirección **http://www.cygwin.com/**

-nr | --no-renaming

Por defecto el compilador renombra automáticamente las variables locales que coincidan con otras variables locales o globales agregándole un underscore delante del nombre, tantos como sean necesarios para que sean diferentes. Este renombre se debe a que algunos de los nombres de las señales del circuito generado están compuestos por los nombres de las variables. Esta opción deshabilita esa funcionalidad.

-ast | --add-statement-time , -ass | --add-statement-space , -ald | --add-logic-depth

Estas opciones fuerzan el modo pretty print (ver más adelante), agregan comentarios por cada instrucción donde se indica el número de períodos de reloj que lleva la ejecución de dicha instrucción, el número de niveles lógicos necesarios y también el espacio requerido indicado en flip-flop y compuertas.

-nfd | --no-force-delays

Normalmente se introduce un estado o delay en los bucles de instrucciones que generarían circuitos que con loops combinacionales. Esta opción deshabilita este procedimiento.

-nwi | --no-width-inference

El compilador puede inferir el ancho en las constantes en función de los anchos de las variables declaradas, poner esta opción deshabilita ese proceso y todas las constantes deberán ser declaradas con el ancho necesario.

-nhc | --no-health-checks

Esta opción deshabilita los controles de alcance, uso y otros chequeos realizados sobre el programa a compilar.

-nsn | --no-sanitise

Lleva a cabo un simple reordenamiento del código del programa para producir una salida más agradable de leer.

-pp | --pretty-print , -ppa | --pretty-print-after

Estas opciones imprimen el código del programa en su representación interna luego de pasar por el parser, la primer opción, y con la segunda luego de la compilación, optimización y otras transformaciones.

-O | --optimise , -O[0-9]

Controlan el grado de esfuerzo en el proceso de optimización. Con –O0 no se optimiza y –O9 o –O es el mayor esfuerzo. La opción por defecto es –O6.

-s | --simulate , -ns | --no-simulate , -ss | --simulate-steps n , -su | --simulate-until n ,
-se | --simulate-extra n , -sr | --simulate-radix n , -sp | --simulate-power ,
-spw | --simulate-power-weights , -stc | --simulate-toggle-count ,
-ir | --input-ram rname fname , -or | --output-ram rname fname

Todas estas opciones controlan distintos aspectos del simulador integrado con el compilador. La simulación es realizada sobre la salida estándar pero para diseños complejos resulta difícil el seguimiento de los distintos valores de las variables.

### 3.1.2 Sintaxis del lenguaje

Veremos la sintaxis necesaria para que un programa escrito en C se pueda compilar con Handel C. Mencionaremos las principales características del programa principal, declaración de variables e interfaces con el mundo exterior.

**Sumario del lenguaje soportado por HCC**

El lenguaje soportado por este compilador es un subconjunto del ANSI C [3], posee variables tipo entero, char y lógicas. También implementa sentencias que permiten realizar asignaciones condicionales, ejecuciones condicionales, bucles de instrucciones y ejecución de procedimientos.

**Tabla 3-2** Resumen de las instrucciones soportadas por Handel C

| **Operadores aritméticos** | | **Operadores Relacionales** | | **Tipos de Datos** | **Sentencias** |
|---|---|---|---|---|---|
| *Con signo* | *Sin signo* | *Con signo* | *Sin signo* | Const<br>const spec<br>bool<br>char<br>int<br>eram<br>ram<br>rom<br>port<br>chan<br><br>void | while (…) {…}<br>do {…} while (…)<br>for (…) {…}<br>if (…) {…} else {…}<br>case (…) {…}<br>{ ; }<br>? :<br>skip<br>stop<br>delay<br>par {…}<br>prialt {…}<br>?<br>! |
| * | .*.<br>+<br>- | ><br><<br>>=<br><= | . >.<br>. <.<br>.>=.<br>.<=.<br>==<br>!= | | |
| **Operadores de bits** | | **Operadores lógicos** | | **Funciones** | |
| &<br>\|<br>^<br>~<br>>><br><<<br><-<br>\\<br>@<br>. (punto) | | &<br>\|<br>^<br>~ | | abs (...)<br>rxrdy (...)<br>txrdy (...)<br>exp2 (…)<br>cond(...) | |

**Nota:** Aparecen resaltadas aquellas instrucciones que no pertenecen al ANSI C o cuya funcionalidad es diferente.

Y destacamos particularmente que NO soporta:

- divisiones, llamado a funciones
- punteros, arreglos, estructuras
- Recursión o mutua recursión

**Extensiones**
Además de las ya conocidas instrucciones del lenguaje C, se agregan un conjunto nuevo que nos permiten definir interfaces con memorias externas, tanto RAM como ROM y definir memorias internas, canales de comunicación internos y puertos de salida.

También se definen constructores especiales que permiten trabajar a nivel de bits realizando operaciones de rotación y de selección o descarte de un determinado número de bits fijado de antemano (constante en tiempo de compilación).

Hay dos constructores que permiten definir o trabajar con instrucciones o bloques de instrucciones en paralelo, una es de composición paralela (Parallel composition), par {...}, que permite indicar que las instrucciones del bloque indicado entre llaves sean ejecutadas en paralelo y no en forma secuencial como se realiza normalmente. El otro constructor llamado de asignación paralela (Parallel assignment) funciona al igual que en otros lenguajes, permitendo en una sola instrucción, asignar valores a distintas variables. Por ejemplo " a , b = a+1 , 56 ; " aquí se incrementa "a" en 1 y se asigna 56 a "b", todo en un solo período de reloj.

**Estructura del programa principal**
Un programa para que se pueda compilar con HCC debe tener la siguiente estructura:

```
        Declaraciones externas
        void main( declaración de objetivo [target] , declaración de interfaz)
{
        declaraciones internas

        sentencias
}
```

**Figura 3-1** Estructura del programa principal para compilar con Handel C.

Pasemos a detallar cada una de las secciones mencionadas en la figura.

**Declaraciones externas**
Aquí podemos definir constantes globales tales como los anchos de los puertos de salidas o de los datos de las RAM o ROM y también *campos de especificaciones* que son una ampliación de las constantes y permiten especificar la arquitectura del objetivo (target), interfaces con las memorias externas y con los puertos.

Para las constantes su estructura es:

```
        const NOMBRE = VALOR : ANCHO;
```

**Figura 3-2** Estructura de definición de constantes.

**Obs**. ANCHO indica el número de bits de la constante.

Para los *campos de especificaciones* su estructura es:

```
const spec NOMBRE = {
                        NOMBRE = { string, string, string, … },
                        NOMBRE ={},
                        NOMBRE = string };
```

**Figura 3-3** Estructura de los campos de especificación.

Los string en este caso son utilizados para indicar a las aplicaciones de Place & Route el nombre del pin al cual queremos asignar dicha señal, válido solo para los archivos de salida en XNF ( parámetro LOCATION ( o LOC) de los símbolos [4]).

**Declaración de objetivo**

Aquí se especifican características de la arquitectura destino del circuito, así como también el formato del archivo de salida generado.

```
void main ( target = NOMBRE _ESP, ... ) { .... } ;
```

**Figura 3-4** Estructura de la declaración del procedimiento principal.

La estructura de su especificación es:

```
const spec  NOMBRE_ESP = {
                        fpga_type = string ,
                        fpga_chip =  string ,
                        clock_pad = string ,
                        not_error_pad = string ,
                        finish_pad = string ,
                        clock_divider = string ,
                        carry_weight = string ,
                        critical_weight = string };
```

**Figura 3-5** Estructura del campo de especificación del target.

Donde:

fpga_type : Indica la familia de fpga. Los valores aceptados son “Xilinx2000”, “Xilinx3000”, “Xilinx4000”, “Xilinx6000”, “VHDL”.

fpga_chip : Indica el chip en particular, es utilizada por las herramientas de Place & Route que leen la especificación en XNF, para otros formatos de salida no se utiliza.

clock_pad : Indica cual es el pin del chip que se utilizará como reloj. También es utilizada solamente en la especificación XNF, en VHDL se pierde esta información.

clock_divider : Es utilizada para generar un divisor del reloj proporcionado, el string debe indicar un número entero que indica el factor de división.

not_error_pad y
finish_pad : Indican que pin del chip es activado para indicar en el primer caso que se ejecutó una sentencia stop y para el segundo que se terminó la ejecución de la máquina de estados (del programa).

carry_weight y
critical_weight : La primera afecta a las líneas de carry de los sumadores tipo "ripple-carry", y la segunda a algunas de las señales del circuito generado. Recomendamos en general, realizar estos seteos desde las herramientas de Place & Route.

**Declaración de interfaz**
Aquí se declaran los canales y puertos de comunicación del programa hacia el resto del mundo (fuera del chip), también son declaradas las memorias RAM externas.
La estructura de la declaración de la RAM es:

```
void main ( eram NOMBRE [NÚMERO DE ELEMENTOS] =  NOMBRE_ESP : ANCHO, ...)
{ ... }
```

**Figura 3-6** Estructura de la declaración de memoria externa

Y la estructura de su especificación es:

```
const spec NOMBRE_ESP={
                        addr = { string , string , …} ,
                        data = { string , string , …} ,
                        ce   = string ,
                        wb   = string  o { string , string , …},
                        en   = string o { string , string , …} };
```

**Figura 3-7** Estructura del campo de especificación de las memorias

Aquí los string indican también que pin del chip corresponden con las señales de la RAM, en particular si para las señales "wb" y "en" indicamos una lista de pins, el HCC copia la misma señal en todos los pins indicados en la lista.

Los puertos son las vías de comunicación principales del programa en C hacia fuera del chip. Poseen un protocolo sencillo mediante un par de señales de control que pueden ser omitidas y el compilador considera que los puertos están siempre listos tanto para recibir como para enviar.

Los puertos se declaran de la siguiente forma:

```
void main ( port (DIRECCIÓN) NOMBRE = NOMBRE_ESP : ANCHO, ...)
{ ... }
```

**Figura 3-8** Estructura de la declaración de los puertos.

Donde *DIRECCIÓN* indica que el puerto puede ser entrada (*IN*) o salida (*OUT*).

Indicar un *campo de especificación* (*NOMBRE_ESP*) es opcional y en caso de no definirlo solo se indica el *ANCHO* del puerto, y el HCC genera internamente las señales de control.

En caso de indicar una especificación su estructura es:

```
const spec NOMBRE_ESP={
                        data = { string , string , …} ,
                        txrdy  = string ,
                        rxrdy  = string };
```

**Figura 3-9** Estructura del campo de especificación de los puertos.

Donde DATA es una lista de pines, comenzando por el menos significativo, y deben existir por lo menos tantos pines como bits de ANCHO del puerto, pueden haber más pines definidos.

Las señales TXRDY y RXRDY son las señales de control del protocolo, se pueden indicar las dos pero el HCC elegirá la correcta según el puerto sea de lectura o escritura. Para el caso de un puerto de entrada (o lectura) se utiliza la señal RXDRY que el dispositivo externo maneja informándole al programa que tiene datos válidos en el puerto. Para los puertos de salida (escritura) se utiliza la señal TXRDY, con ésta el programa le indica al dispositivo externo que hay datos válidos en el puerto.

Los canales (channels) se declaran:

```
void main ( chan (DIRECCIÓN) NOMBRE : ANCHO, ...) { ... }
```

**Figura 3-10** Estructura de la declaración de los canales.

Donde *DIRECCIÓN* indica que el canal puede ser entrada (*IN*) o salida (*OUT*) y *ANCHO* su número de bits.

Es importante destacar que los channels son utilizados para la comunicación fuera del chip en las etapas de depurado utilizando el simulador integrado con el HCC, luego para las implementaciones en hardware es más conveniente utilizar los puertos (ports).

**Declaraciones internas**

Aquí se pueden definir constantes, variables de diferentes tipos, RAM y ROM internas al chip, canales, procedimientos y subexpresiones.

Las variables son declaradas al igual que en C con el agregado opcional del número de bits (ANCHO) del registro que guardará su valor. Su estructura es:

```
TIPO NOMBRE, NOMBRE = valor inicial, ... : ANCHO ;
```

**Figura 3-11** Estructura de la definición de variables

Donde *TIPO* indica el tipo de variable, son aceptados int, char y bool.

El proporcionar un valor inicial es opcional, si el valor es distinto de 0, esta inicialización costará un período de reloj.

El caso de char y bool son casos particulares del int, tan solo nos ahorramos indicarle el ancho de la variable pues para bool es 1 y para char es 8.

Las RAM y/o ROM internas son tratadas como si fueran arrays e internamente la RAM es implementada con flip-flops. Su declaración es:

```
ram TIPO NOMBRE[NUMERO DE ELEMENTOS] : ANCHO;
```

```
rom TIPO NOMBRE = { número, número, ...} : ANCHO;
```

**Figura 3-12** Estructura de la definición de memorias internas.

El *ANCHO* indicado corresponde al de los datos de la memoria. Para el caso de las direcciones su ancho es calculado con $\log_2(n)$ donde "n" es el número total de elementos de la memoria. El *TIPO* es opcional y para el caso de indicarse char o bool no es necesario establecerse el ANCHO.

Los canales son definidos en forma similar a como se refirió en la declaración de interfaz salvo que no es necesario indicar una dirección para el canal. Son útiles para establecer comunicación entre procesos que corran en paralelo. Solamente un proceso puede utilizar un canal como salida (en un mismo instante del tiempo) y solamente un proceso puede utilizarlo como entrada.

Su declaración es:

```
chan TIPO NOMBRE : ANCHO ;
```

**Figura 3-13** Estructura de la definición de canales internos.

Donde *TIPO* al igual que con las variables, es opcional.

Los procedimientos son declarados sin parámetros de entrada o de salida, la única forma de comunicación de datos con el resto del programa es mediante el uso de variables globales o canales. Se prohíbe la recursión, así como también la ejecución de un mismo procedimiento desde diferentes ramas de un código paralelo.

Su declaración es:

```
void NOMBRE() {
        declaraciones internas del procedimiento

        sentencias };
```

**Figura 3-14** Estructura de la declaración de subprocedimientos

Las variables declaradas internamente a los procedimientos mantienen su valor entre las distintas ejecuciones, esto es pues son implementadas con registros que son inicializados a 0 al resetear los flip-flops, salvo que en la declaración se indique un valor inicial con lo cual existirá un ciclo de inicialización de variables. Si los nombres de estas variables coinciden con otras o con otros canales son renombrados automáticamente agregando "_" (underscore) antes del nombre, tantos como sean necesarios (esta facilidad se puede deshabilitar con la opción de compilación –nr)

Las subexpresiones son declaraciones de expresiones que son comúnmente utilizadas. La necesidad surge pues cada expresión requiere un hardware que la implemente y el HCC no es capaz, en forma automática, de reutilizar el hardware generado para una misma expresión que se utilice a lo largo de un programa. Así entonces podemos definir una subexpresión y utilizarla cuando sea necesario sin que ello genere un cargo extra de hardware. Es un concepto muy parecido a una función pero con la diferencia que no se permite el pasaje de parámetros, todas las variables que intervienen en la subexpresión tienen que ser declaradas previamente. Estas subexpresiones pueden ser utilizadas para definir otras subexpresiones.

Su declaración es:

```
TIPO NOMBRE( ) = expr. lógica, aritmética y/o trabajo con bit's : ANCHO ;
```

**Figura 3-15** Estructura de la definición de subexpresiones.

**Sentencias**

Aquí se pueden utilizar cualquiera de las sentencias mencionadas en la Tabla 3-2 con la sintaxis propia del lenguaje C (para las pertenecientes al ANSI C). Solamente destacaremos el uso de las que son extensiones al lenguaje y que son:

par {...}

Le indica al compilador que las instrucciones del bloque entre llaves se ejecuten en paralelo. Por ejemplo:

```
par {
        a = a + 1;
        b = 5;
}
```

Realiza el incremento de "a" en uno y la asignación de 5 a "b" ambas en el mismo período de reloj. Hay que recordar que las variables se implementan con flip-flop y por lo tanto un cambio en ellas se ve reflejado recién en el siguiente ciclo. Por ejemplo:

```
a = 1;
par {
        a = a + 1;
        b = a;
}
c = b;
```

El valor que se carga en la variable "c" es 1 pues el incremento de "a" se realizó en paralelo con la copia de su valor en "b" por lo tanto en "b" se cargó el valor "viejo" de "a".

prialt {...}

Es un constructor que permite alterar el flujo de ejecución del programa, su sintaxis es la siguiente:

```
prialt {
        exp. booleana $ lectura de canal :
                sentencia
o
        exp. booleana :
                sentencia
o
        lectura  de canal :
                sentencia
        ...
}
```

**Figura 3-16** Estructura del prialt {}.

Las distintas alternativas se forman de una expresión booleana y una lectura de un canal. Cada condición especificada en el prialt es evaluada en el mismo período de reloj pero con un orden de prioridad dado por su orden de aparición. Cuando la exp. booleana es verdadera y la lectura del canal es completada con éxito entonces se ejecuta la sentencia de dicha rama. Pueden ser omitidas la expresión o la lectura pero no ambas. El programa no continuará hasta que alguna de las ramas sea ejecutada.

cond(...)

Es una generalización del operador " ? : ", útil cuando es necesaria más de una alternativa. Su sintaxis es:

```
cond ( expresión test, c1 -> exp. 1, c2 -> exp. 2, ... ,
        cn -> exp. n, default -> exp. por defecto)
```

Esta función devuelve el contenido de la "expresión i", si "expresión test" es igual a $c_i$ sino devuelve el valor de "expresión por defecto". Los valores de $c_i$ deben ser distintos entre sí.

skip

Es una instrucción que no hace nada y además no consume ningún período de reloj.

delay

Es una instrucción que no hace nada pero consume un período de reloj en ejecutarse. Opcionalmente acepta un parámetro, delay N, donde N representa el número de ciclos de espera.

stop

Esta instrucción detiene inmediatamente la ejecución del programa. Esto es que lleva la máquina de estados a un estado en el cual se queda indefinidamente (o hasta el reset). Otros procesos que se ejecuten en paralelo con ella continuarán su ejecución.

? y !

Éstas se encargan de leer valores de un puerto o canal y asignarlo a una variable ("?"), y de escribir en un puerto o canal el contenido de una variable o expresión ("!"). Su ejecución lleva un período de reloj. Por ejemplo:

```
// lecturas desde un puerto o canal
entrada ? a; // copia el contenido del puerto
             // en la variable a

// escrituras en puerto o canal
salida ! a; // copia en el puerto el contenido
            // de la variable a
salida ! (a+1); // copia en el puerto el
                // resultado de la expresión a+1
```

<- , \\ , @ y .(punto)

Estos operadores trabajan directamente con los bits de las variables o expresiones, en la siguiente tabla describimos su funcionalidad.

**Tabla 3-3** Descripción de operadores de bits especiales de Handel C.

| **Operador** | **Ancho del Resultado** (bits) | **Descripción** |
|---|---|---|
| e <- p | p | Devuelve los p bits menos significativos de "e" |
| e \\ p | n-p | Devuelve los n-p bits más significativos de "e" |
| e @ f | n+m | Concatena "e" y "f" |
| e.p | 1 | Devuelve el bit p-ésimo de "e" |
| e.(p..q) | q-p+1 | Devuelve los bits de "e" que van de "p" hasta "q" |

e = expresión de n bits de ancho
f = expresión de m bits de ancho
p y q = deben ser constantes en tiempo de compilación.

rxrdy(...) y txrdy(...)

Estas funciones devuelven verdadero si el canal, pasado como parámetro, esta listo para recibir o listo para transmitir, respectivamente.

exp2(…)

Calcula la exponencial base 2 de la expresión o variable pasada como parámetro.

### 3.1.3 Circuito generado

Aquí hablaremos de cómo el compilador HCC genera la máquina de estados resultante de nuestro programa y algunos detalles del circuito para algunas de las instrucciones del lenguaje C, también veremos cuales son los tiempos o ciclos de reloj que cada instrucción emplea.

**Timing y estados**
La semántica del HCC asume que existe un esquema de reloj en el cual hay solo un reloj global y todos los flip-flop cambian en el mismo flanco.
El Handel C genera máquinas de estado secuenciales tipo Mealy donde cada estado es representado por un flip-flop, esta codificación recibe el nombre de **one hot** (o **uno activo**), que produce mejores resultados en arquitecturas ricas en flip-flop como lo son las FPGA y además en general se requiere menos lógica combinatoria para determinar el cambio de estado [5].

Se generan nuevos estados en los siguientes casos:

- Cada asignación de una variable, tanto interna como tipo RAM o ROM
- Cada entrada o salida de un puerto o canal
- Instrucción especial de espera (Delay)
- Para While o For cuyo bloque de instrucciones dure menos de un ciclo de reloj

Las asignaciones a variables, ya sean simples, paralelas o condicionales; así como también las operaciones sobre los puertos (lectura y escritura) requieren solamente de un ciclo de reloj para ser ejecutadas. En particular las variables son formadas con registros de flip-flop y los puertos de salida son formados con lógica combinatoria, por lo tanto hay que tener cuidado con el origen de los datos que salen por los puertos, pues eventualmente podría tener muchos glitches.

Es importante destacar que no se genera ningún tipo de circuito[2] para las variables que no intervengan de alguna forma en las interfaces del circuito final con el mundo exterior

Las expresiones o subexpresiones de una sentencia son generadas con lógica combinatoria por tanto expresiones complejas limitarán la frecuencia máxima del reloj de la máquina de estados. Los operadores condicionales también son generados con lógica combinatoria y por tanto evaluados en un solo período de reloj.

Las instrucciones WHILE y DO ... WHILE no requieren tiempo extra para su ejecución, duraran tanto como dure su bloque de instrucciones multiplicado por el número de repeticiones del mismo.

Las instrucciones IF y CASE también demoran según sus bloques de instrucciones, dependiendo su duración de cual de ellos se ejecute y de la duración del mismo.

Para el caso del FOR se tiene que la sentencia de inicialización llevará tantos ciclos como sean necesarios, en general un ciclo, no hay ciclos agregados por la sentencia de test y la

[2] Dependiendo del nivel de optimización que se use desde la línea de comandos.

duración del bloque de instrucciones será la necesaria más los ciclos necesarios por la sentencia de incremento, en general un ciclo solo.

En la composición paralela (o bloque de instrucciones en paralelo), par {}, podemos destacar que comienzan todas en el mismo instante y la duración total del bloque la determina la rama de instrucciones que dure más períodos de reloj (ciclos) en ejecutarse.

Y por último los procedimientos no generan ningún ciclo especial y demoran en ejecutarse lo que demore su bloque de instrucciones.

En la siguiente figura vemos un diagrama de la máquina de estados más sencilla correspondiente a un programa sin ninguna instrucción de control de flujo (loops, CASE, IF o PAR). Un ejemplo de como sería un programa que genere una máquina así se puede ver en la Figura 3-18.

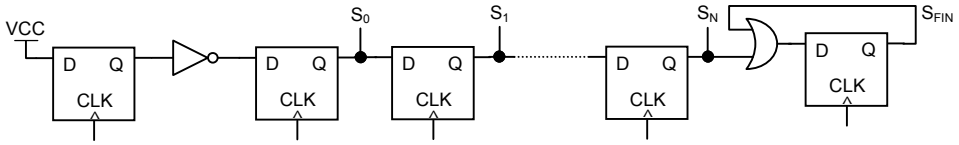



**Figura 3-17** Máquina de estados sencilla.

```
void main( port (in) ent:4, port(out) sal:4){
int a, b, c:4;
    ent ? a;   // estado S0
    ent ? b;   // estado S1
    c = a - b;   // estado S2
    sal ! c;   // estado S3
    c = a + b;   // estado S4
    sal ! c;   // estado S5, sería el SN del diagrama anterior
}
```

**Figura 3-18** Ejemplo de programa sin instrucciones de control de flujo.

A continuación describiremos en detalle los circuitos más relevantes, generados por el compilador, para las distintas entidades de un programa.

**Sobre las variables**

Como mencionamos anteriormente las variables son implementadas con un registro de flip-flop y utilizan clock enable (CE) para cargar los valores en los estados que las variables son actualizadas. No se guardan ni genera ningún tipo de registros para variables intermedias cuyos valores no afecten los datos escritos o leídos de los puertos, canales o RAM.

La entrada de datos del registro de una variable está compuesta por un bloque que llamaremos SELECTOR, donde por un lado entran los datos “nuevos” de la variable para cada uno de los estados donde ésta es actualizada, y por otro lado al SELECTOR le entran como señales de control las variables de estado correspondientes. La salida del SELECTOR será 0 cuando no hay ninguna señal de control activa o valdrá Di cuando este activa la señal $SEL_{Di}$.

La estructura de todo el conjunto podríamos esquematizarla como se puede apreciar a continuación para el caso de una variable de N bits de ancho que es actualizada en "y+1" estados distintos nombrados $S_{x0}$ hasta $S_{xy}$ .

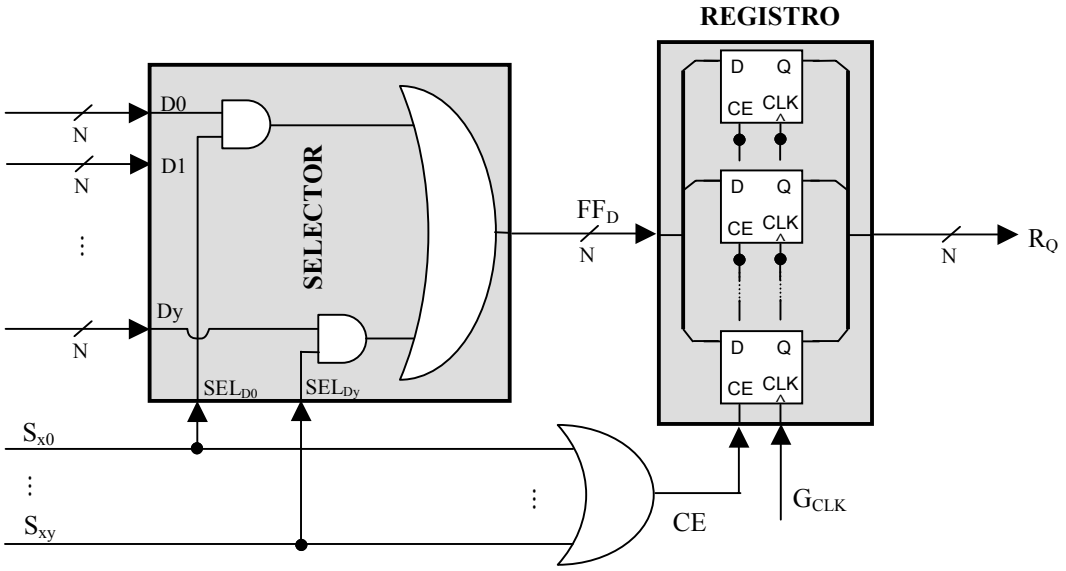



**Figura 3-19** Circuito de manejo del registro de una variable.

Observemos que en los datos Di entra el nuevo valor a guardar en el registro R, la señal $S_{xi}$ se encuentra a 1 pues es la variable de estado para ese estado de la máquina y por lo tanto es utilizada en la señal $SEL_{Di}$ para que la salida $FF_D$ tenga el valor correspondiente a la entrada Di. El selector junto con la lógica que controla el Clock Enable (CE) del registro aseguran la correcta actualización de los valores.

Es importante mencionar que para el caso de trabajar con bloques de instrucciones en paralelo hay que tener cuidado pues si trato de actualizar la misma variable desde bloques diferentes se podría dar el caso que para un mismo período tenga más de una señal de control activa (caso que se puede dar y que comentaremos más adelante) y por tanto el selector tendrá como salida un OR de las dos entradas correspondientes, o sea que no se realizaría una correcta actualización de la variable.

**Sobre los puertos.**

Los puertos de salida no son salidas de registros, es una estructura similar al caso de las variables, consta de un selector y una etapa de generación de una señal equivalente al CE, como se puede apreciar en la Figura 3-20, las salidas del chip están cableadas directamente desde las salidas del selector.

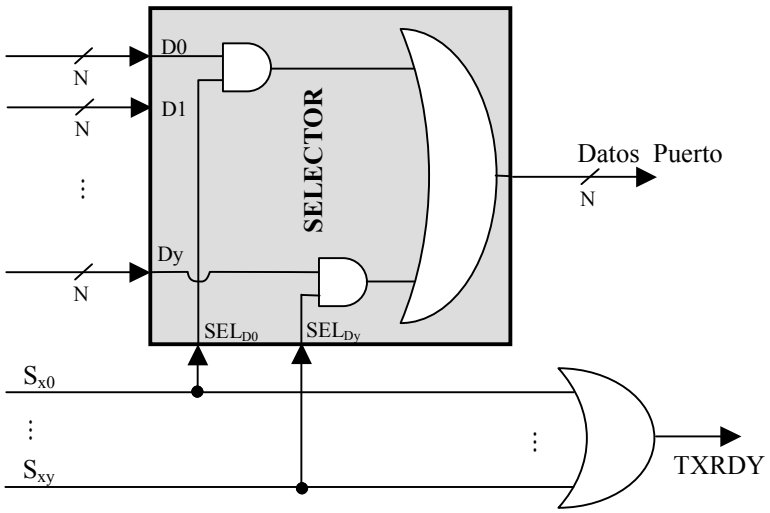



**Figura 3-20** Circuito de manejo de un puerto de salida

Aquí, como en el caso anterior, las señales $S_{xi}$ son las variables de estado de la máquina de estados, que como mencionamos utiliza codificación one hot por tanto solo una señal $S_{xi}$ va a estar activa a la vez. La señal TXRDY se forma con un OR de las variables de estado en los cuales el puerto de salida transmite ($S_{xi}$).

Es importante notar que no contamos con la posibilidad de tener salidas con tercer estado para poder fácilmente integrarlas a un bus de datos, en capítulos siguientes veremos una solución a dicho problema.

Para los puertos de entrada recordemos que estas lecturas guardan sus valores en variables por eso vemos que el CE del registro de la variable incluye a las variables de estado correspondientes al estado donde se realiza la lectura desde el puerto.

Al especificar pines para las señales de control de los puertos observamos que para el puerto de escritura se utiliza solamente la señal TXRDY y el puerto de lectura utiliza la señal RXRDY.

Observamos un problema en la generación del circuito para los puertos de lectura cuando se especifican pines a RXRDY, en este caso se declara la señal como entrada del circuito y además es definida como salida en un flip-flop, esto obviamente causa errores en los sintetizadores de VHDL.

**Sobre la RAM externa**

Analizaremos los circuitos generados para el acceso a RAM externa.
Las direcciones son trabajadas al igual que los puertos de salida, con un selector que utiliza como señales de control las variables de estado de todos los estados donde se trabaja con la RAM (tanto en lectura como en escritura).

Los pines de datos de la RAM se consideran bidireccionales, pero internamente se trabajan con dos líneas, una de entrada y otra de salida con un bufer triestado que lo conecta al pin de salida del chip. La salida de estos datos también se implementa con un selector y en este caso se utilizan como señales de control las variables de estado de los estados donde se realizan escrituras en la RAM.

El siguiente esquema muestra las conexiones del circuito:

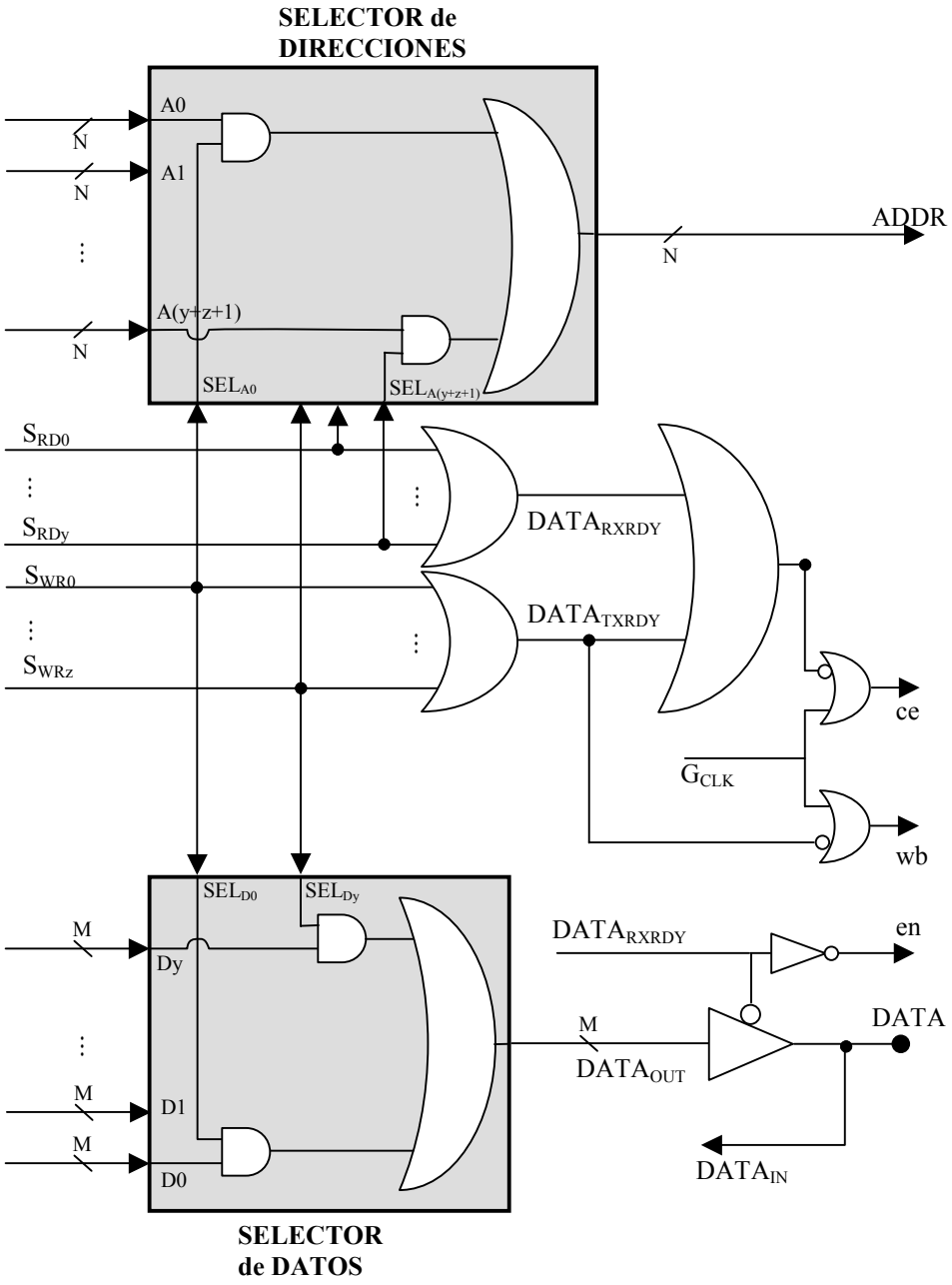



**Figura 3-21** Circuito de manejo de la RAM externa

Donde las variables de estado $S_{RD0}$ a $S_{RDy}$ corresponden a estados donde se realizan lecturas y las variables de estado $S_{WR0}$ a $S_{WRz}$ corresponden a estados donde se realizan escrituras. Los pines de salida de direcciones son los correspondientes a las señales

ADDR, para los datos bidireccionales es DATA, “ce” para el chip enable, “wb” para el pulso de write y por último “en” para el output enable (o pulso de lectura).

Veamos ahora un ejemplo para el que mostramos las señales que se generan.

```
const spec TIPO_RAM = {
        Addr = {"P120", "P124", "P141", "P128",
                "P129", "P130", "P137", "P132"},
        data = {"P2", "P1", "P151", "P152", "P153",
                "P154", "P155", "P156"},
        ce   = "P141",
        wb   = "P127",
        en   = "P92"};
main( target = XXX,
        …
```

```
        eram MEM[256] = TIPO_RAM : 8){
...
        C=MEM[ i ];   //lectura de la ram
        D=MEM[ j ];   //lectura de la ram
        …
        MEM[ b ]=A;  //escritura en la ram

}
```

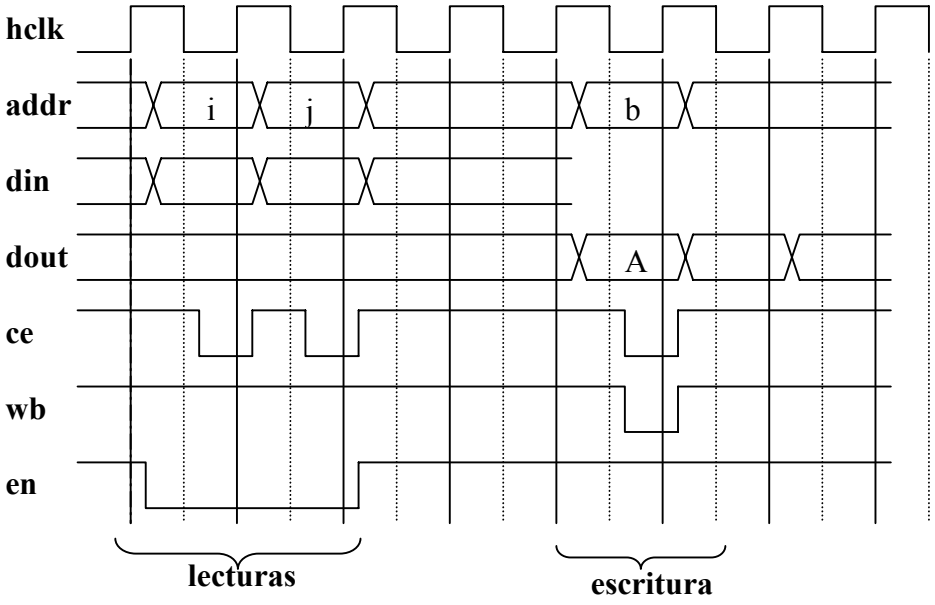



**Figura 3-22** Ejemplo y diagrama de tiempos respectivo de uso de memoria RAM externa.

**Sobre las memorias internas**

RAM interna

Las memorias RAM internas se implementan con flip-flop, esto lo acepta cualquier familia pero se desaprovecha el uso de celdas de memoria en las familias que las tienen (por ejemplo XC4000 y FLEX10K).

En la Figura 3-23 se aprecia el circuito que se implementa con la siguiente declaración de RAM:

```
        ram  X[M+1] : (N+1);
```

La RAM es un banco de M+1 registros (Q_X0,.., Q_XM) de N+1 bits de ancho, la entrada de datos (R_X_IN) proviene de un selector que según el estado decide los valores que se cargan en la RAM al igual que en el caso de las variables. Las direcciones también provienen de un selector, éste tiene direcciones válidas en los estados de lectura y escritura, adémas alimenta un decodificador que activa el registro a usar. Para escribir en un registro se tienen que activar tanto la habilitación (RAM_ENA_i) como un pulso de escritura (R_X_WE), este último es el OR de las variables de los estados en que se escribe en memoria.

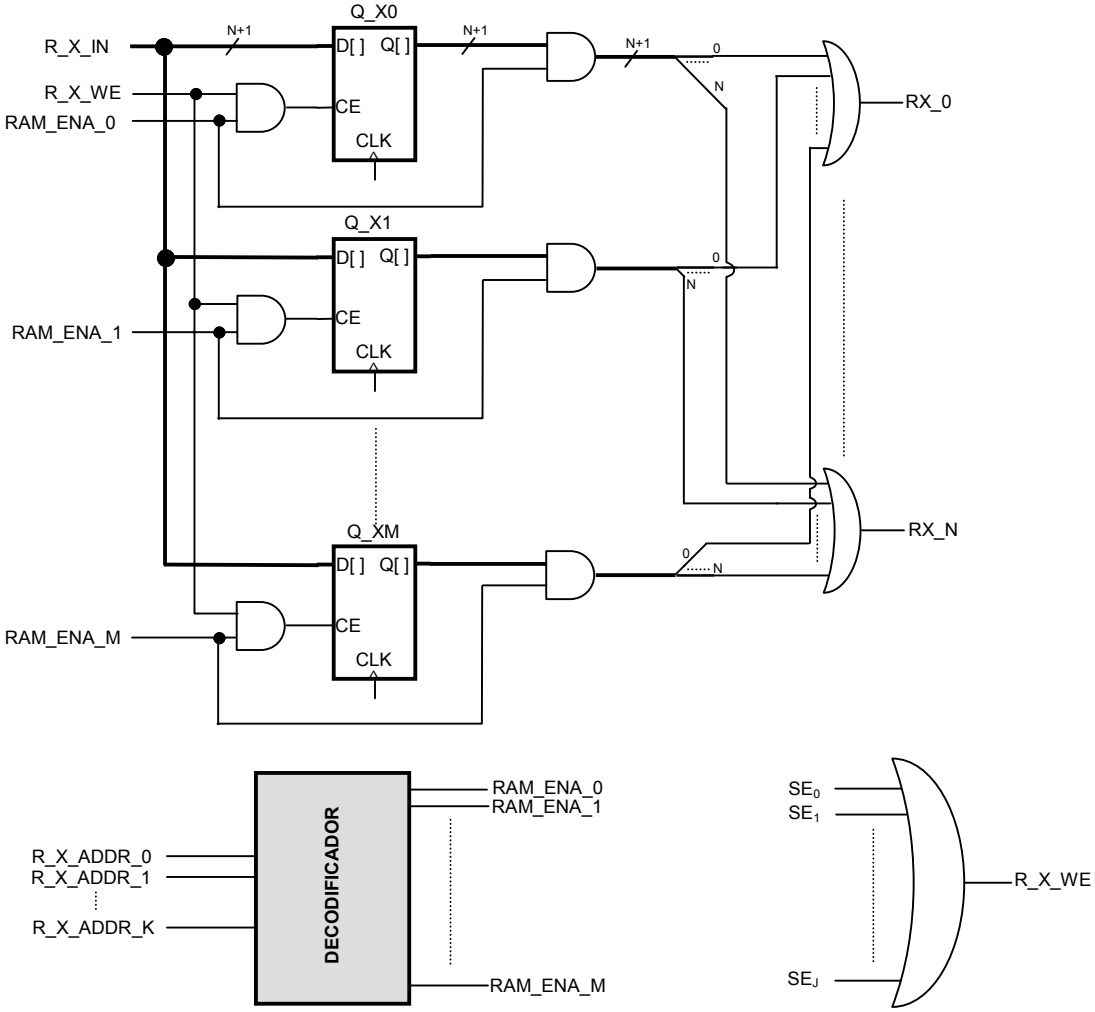



**Figura 3-23** Circuito generado para la RAM interna de Handel C.

En la siguiente figura (Figura 3-24) mostramos un código en el que hay dos lecturas y dos escrituras a RAM y como quedarían los selectores de direcciones y datos para este ejemplo.

```
void main(…  // declaración de puertos
            ){
ram x[tamaño]: ancho;
… // otras declaraciones
x[ dirE 1] = dato 1;   // escritura, estado SE1
… // sigue el programa
x[ dirE 2] = dato 2;   // escritura, estado SE2
…
c = x[ dirL 1];  // lectura, estado SL1
…
c = x[ dirL 2];  // lectura, estado SL2
…
}
```

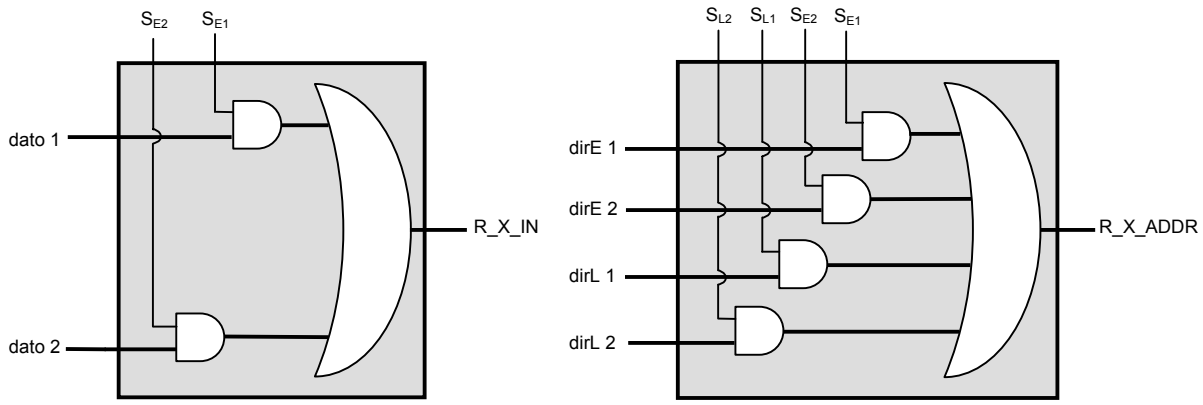



**Figura 3-24** Ejemplo y diagrama de los selectores de datos y direcciones de la RAM externa.

ROM interna

El compilador genera un decodificador cuya entrada es la dirección de acceso a la ROM, ésta se decide mediante un selector. Las salidas del decodificador actúan como señales de control de otro selector que elige entre los datos guardados en la ROM el que se debe mostrar en la salida.

En la Figura 3-25 se puede ver un diagrama del circuito generado para una ROM con M+1 datos y que se accede en N estados ($S_{L1}$,.., $S_{LN}$).

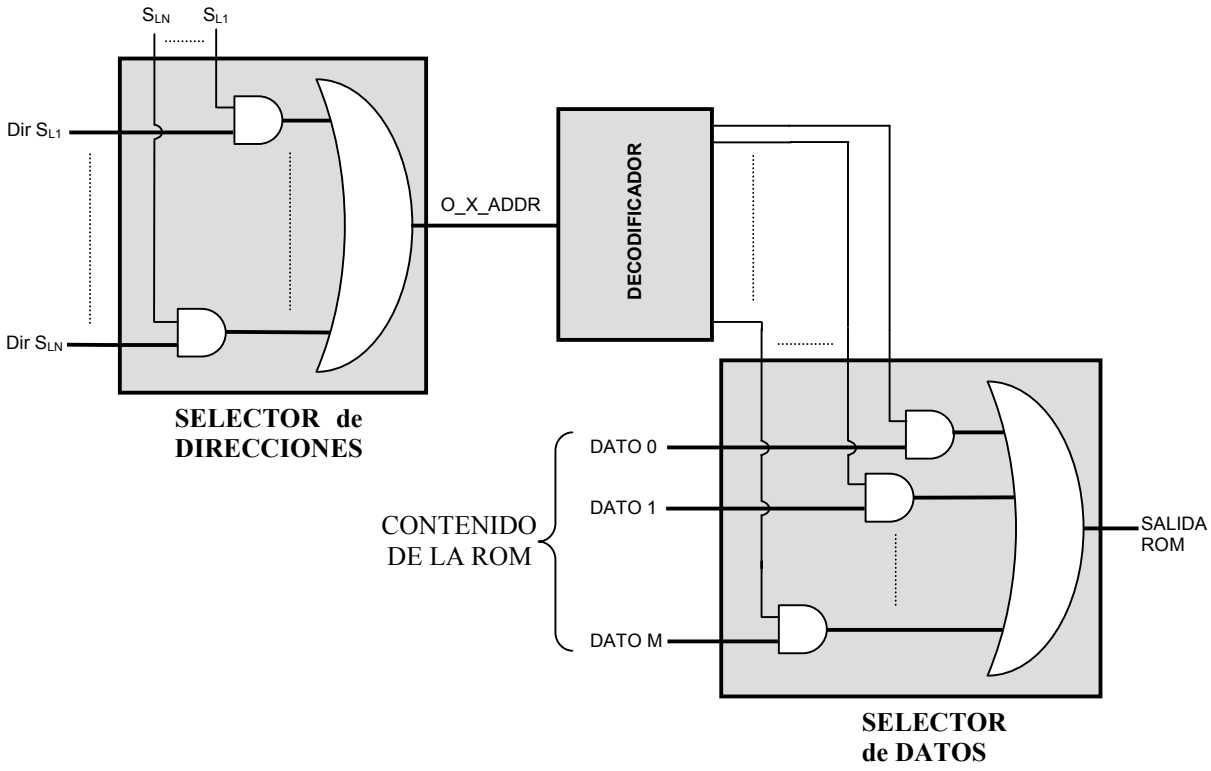



**Figura 3-25** Circuito generado para la memoria ROM.

**Sobre los bucles de instrucciones (iteraciones)**

WHILE

Para los WHILE se genera la lógica combinatoria necesaria que controla los cambios de estado. Ésta se encarga de verificar si la condición de test (indicada en el WHILE) se cumple, siendo así activa la variable de estado correspondiente a la primer instrucción del cuerpo del WHILE. Al llegar al final de este bloque se verifica nuevamente la condición y si ésta se cumple vuelve a activar la variable de estado del primer estado del bloque, en caso de no cumplirse se activa la correspondiente al primer estado a continuación del WHILE.

Se reutiliza el hardware generado para verificar la condición en los estados donde se precisa verificar (antes de entrar al WHILE y en la última instrucción del bloque principal).

Podemos esquematizar el circuito que controla las variables de estado de la siguiente forma:

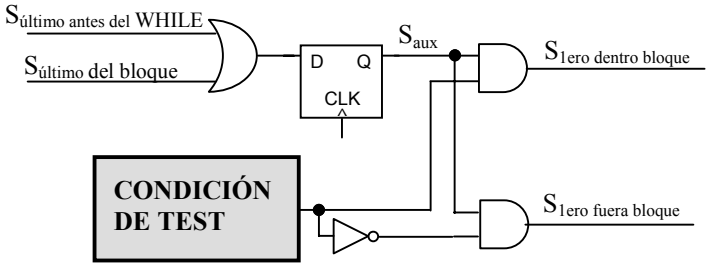



**Figura 3-26** Circuito de control del WHILE

Donde $S_{NOMBRE}$ representa la variable de estado correspondiente al estado ubicado donde se indica en *NOMBRE*. Cuando se menciona la palabra "bloque" se hace referencia al bloque de sentencias que forman el cuerpo del WHILE.

En el siguiente ejemplo mostramos donde se usan los estados mostrados en la Figura 3-26.

```
const hw = 4;
const sw = hw * 2;
void main( port (in) ent:hw, port (out) sal: sw){    // declaración de puertos, sin target
    int a, b: hw;
    int c: sw;
    ent ? a;    // lectura de puerto, se usa S último antes del WHILE
    while(a != 0){
        ent ? b; // lectura de puerto, se usa S 1ero dentro del bloque
        c = c + b; // se usa S 2do dentro del bloque
        a = a - 1; // se usa S último del bloque
    };
    sal ? c; // escritura a puerto, se usa S 1ero fuera del bloque
}
```

Para el caso de la sentencia DO{…}WHILE ( cond ); el circuito generado es el que se muestra en la siguiente figura.

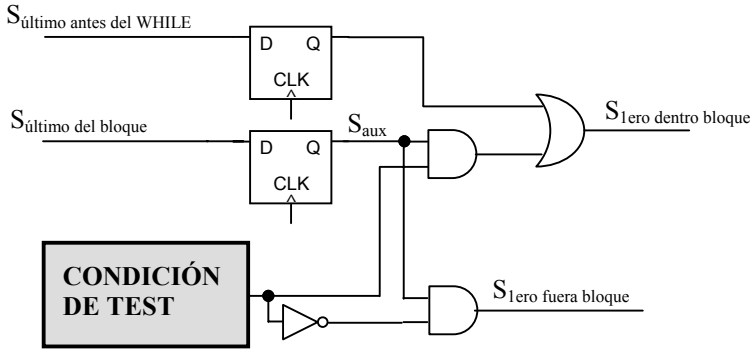



**Figura 3-27** Circuito de control del DO{ }WHILE( ).

FOR

En la implementación del FOR podemos observar que la máquina de estados primero pasa por un estado donde ejecuta las instrucciones de inicialización, en general esto lleva un período de reloj. A continuación ejecuta la primera instrucción del bloque principal o la primer instrucción que se encuentra fuera del bloque principal, esto según sea verdadera o falsa la condición de test introducida. La verificación de esta condición, al igual que para el WHILE, se genera con lógica combinatoria exclusivamente. El bloque principal se ejecuta normalmente y a su término se genera por lo menos un estado más en el cual se ejecutan las instrucciones de incremento indicadas para el FOR. Terminadas éstas se vuelve al punto en el cual se determina el estado siguiente en función de la condición de test.

La lógica de control la podemos esquematizar de la siguiente forma:

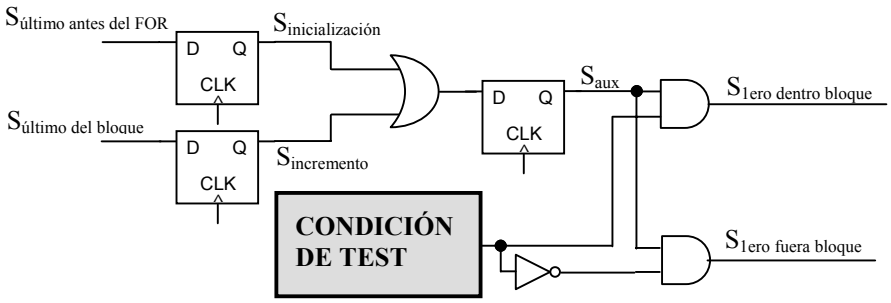



**Figura 3-28** Circuito de control del FOR

Donde $S_{NOMBRE}$ representa la variable de estado correspondiente a los estados que realizan la función que *NOMBRE* indica (por supuesto relativo a lo definido en la sentencia FOR).

A continuación mostramos en un ejemplo como se usan las variables de estado.

```
const hw = 4;
const sw = hw * 2;
void main( port (in) ent:hw, port (out) sal: sw){    // declaración de puertos, sin target
    int a, b: hw;
    int c: sw;
    ent ? a;    /* lectura de puerto, se usa S último antes del FOR , luego de este estado
                   viene S inicialización */
    for(i = a; i != 0; i = i - 1){
        ent ? b; // lectura de puerto, se usa S 1ero dentro del bloque
        c = c + b; // se usa S 2do dentro del bloque
        a = a - 1; // se usa S último del bloque , luego de este estado viene S incremento
    };
    sal ? c; // escritura a puerto, se usa S 1ero fuera del bloque
}
```

## Sobre los operadores y sentencias condicionales

COND

Esta instrucción genera un pequeño selector cuya salida puede entrar al selector de una variable o formar parte de la lógica de una expresión mayor. Para analizar el primer caso veamos un ejemplo:

```
...
c = cond( condición , 0 -> c0, 1-> c1, 2 -> c2, 3 -> c3);
...
```

**Figura 3-29** Ejemplo para mostrar circuito generado por cond( ) al asignarlo a una variable.

Para este mismo ejemplo el circuito que se genera es el siguiente:

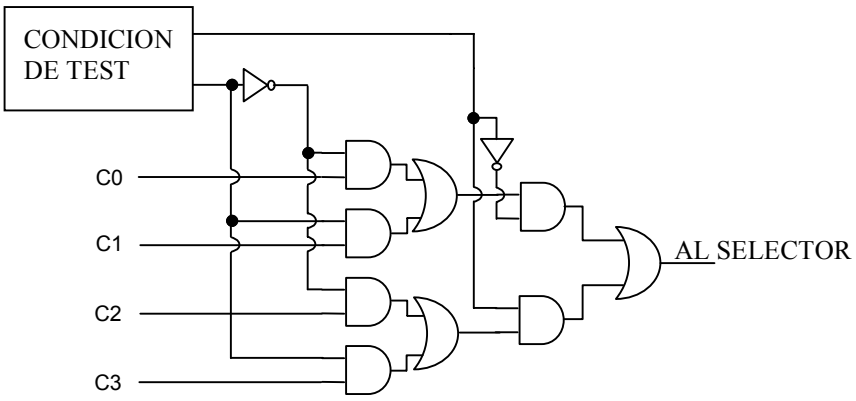



**Figura 3-29** Circuito generado para el ejemplo anterior.

En el selector la señal que activa la salida de este circuito es la señal de estado de la instrucción.

OPERADOR ? :

Este operador es un caso particular del anterior en que la condición de test tiene solo dos posibles valores 0 y 1. Vemos a continuación un ejemplo de su uso y el circuito que se genera.

```
...
c = (condición ? c1 : c0);
...
```

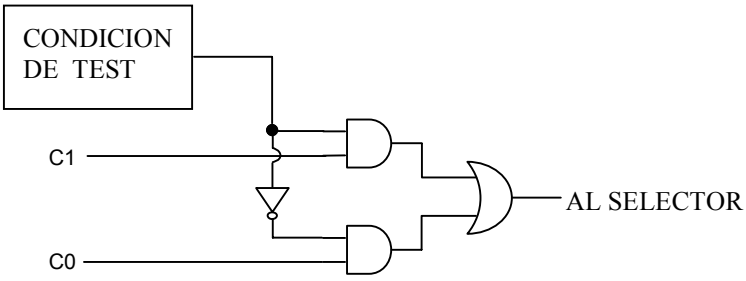



**Figura 3-30** Ejemplo de uso de operador ? : y circuito generado.

En el caso en que el resultado no se asigne a una variable sino que se use en una expresión lo único que cambia es que la salida en lugar de ir al selector de dicha variable forma parte la lógica de dicha expresión.

IF

En la Figura 3-31 mostramos el circuito generado para el caso general en que se tienen varios estados en cada rama, no necesariamente la misma cantidad.

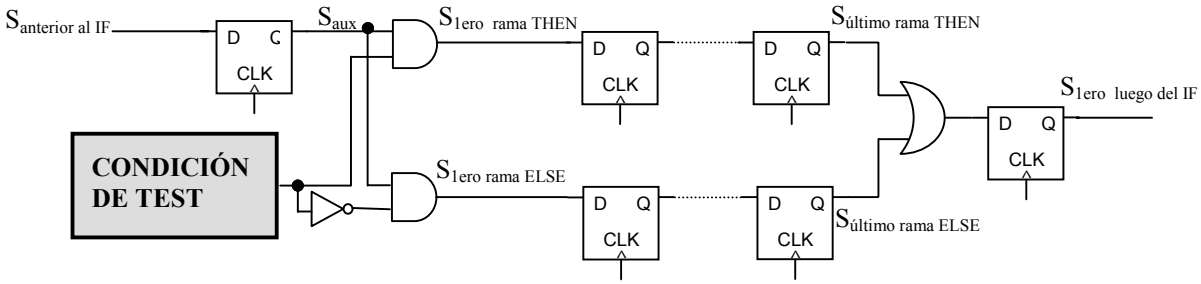



**Figura 3-31** Circuito generado para la sentencia IF.

Si no hay cláusula ELSE la señal de control $S_{1ero\ rama\ ELSE}$ coincide con $S_{último\ rama\ ELSE}$ .
En el siguiente código mostramos las señales de control que se usan en cada caso, éstas como vimos entran al selector de las variables.

```
...
z = Z1;    // esta usa S anterior al IF
if ( cond ){
        c = C1;              // esta usa S 1ero rama THEN
        ...
        k = K1;              // esta usa S último rama THEN
}
else {
        c = C2;              // esta usa S 1ero rama ELSE
        ...
        m = M1;  // esta usa S último rama ELSE
};
y = Y1;              // esta usa S 1ero luego del IF
...
```

**Figura 3-32** Ejemplo de activación de estados en una sentencia IF.

**Sobre los operadores**

Sumas y Restas

Las sumas y restas las implementa con el método de acarreo propagado o Ripple Carry (ver Apéndice A).
No se genera el carry del bit más significativo y por tanto en principio no hay forma de saber si hubo o no overflow pues no se genera ninguna señal. Se podría verificar de diferentes formas, según los datos estén representando números en complemento a dos o no. Si no es complemento a dos se podría observar si el resultado da más chico que alguno de los sumandos, otra forma sería destinar más bits para el resultado definiendo previamente los sumandos de un ancho mayor pues la suma es del ancho de sus sumandos. Si los sumandos están en complemento a dos habría que fijarse en la siguiente condición: si los sumandos son de igual signo y el resultado es de signo contrario entonces hay overflow.

Producto

Handel C implementa la multiplicación de números con y sin signo, usando el método llamado “right shift algorithm” (ver Apéndice A). Este método (para números sin signo)

consiste básicamente en corrimientos y sumas. Para cada bit en uno de uno del multiplicador se desplaza el multiplicando tantas veces como la posición del bit en uno en el multiplicador. En los bits que sean cero se genera un vector con todos los bits "0". Luego se suman todos estos vectores y esta suma es el resultado de la multiplicación.

Para trabajar con números con signo, el HCC utiliza unos bloques que extraen el valor absoluto de los números y luego de calculado el resultado se ajusta el signo convenientemente.

Podemos esquematizar el producto de dos variables con signo, y a modo de ejemplo con "a" y "b" (de 4 bits c/u), y su resultado es guardado en z (de 8 bits).

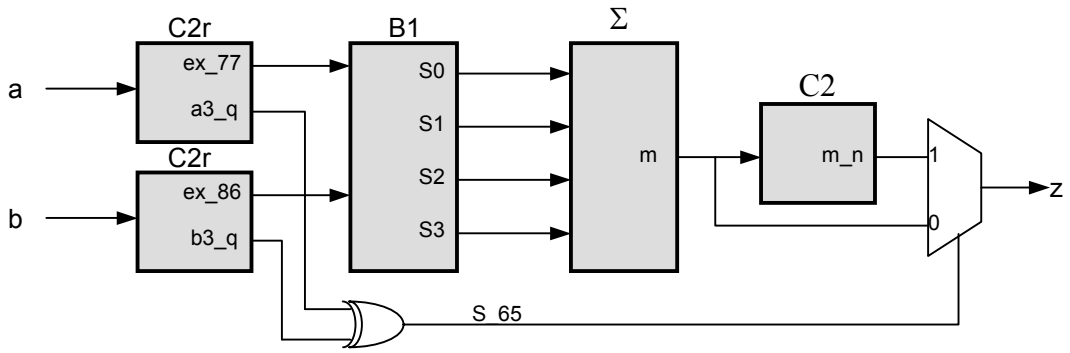



**Figura 3-33** Circuito de multiplicación de dos números

Donde los bloques C2r lo que hacen es calcular el valor absoluto de las entradas y generar una señal con el signo original. El bloque B1 genera los vectores Si correspondientes de hacer el AND de los bit de "a" con el bit i-ésimo de "b", luego el bloque Σ realiza la suma corrida de estos vectores obteniendo el resultado intermedio "m" (sin signo). El bloque C2 calcula el complemento a 2 y por último el multiplexor determina cual valor es el resultado en función del signo de los factores.

Comparadores

La implementación de los operadores de comparación se realiza en base a la resta de sus operandos, por ejemplo para calcular a.<=.b se implementa una lógica combinatoria que calcula "a-b" y luego si hay carry o si todos los bits del resultado son 0 se interpreta que la condición es verdadera.

Tanto para las comparaciones con "<" como con las que son con ">" siempre son implementadas haciendo la misma resta y luego se ajusta convenientemente el próximo estado según el resultado obtenido y la operación de comparación deseada.

Los casos de comparaciones sin signo son generados con una lógica más sencilla pues no hay que hacer consideraciones especiales para el último bit (el de signo), alcanza con calcular el carry de éste para decidir. Para el caso con signo hay que calcular expresamente el último bit, y verificar si ocurre la condición de overflow que es la siguiente: si los operandos son de signo distinto y la resta da de igual signo que el segundo operando entonces hay overflow. Entonces para verificar si a<b se verifica si la resta da negativa y no hay condición de overflow o si da positiva pero si hay overflow,

esto último es como hacer un xor ente el bit de signo del resultado y la condición de overflow.

Los casos de comparaciones que no poseen el "=" son también generados con lógica más sencilla pues solamente se calculan los bits de carry y con ellos se decide; cuando aparece el "=" es necesario además calcular los bits resultado de la resta y verificar que sean todos 0.

**Sobre la composición paralela**

La ejecución de bloques en paralelo tiene algunos detalles que es conveniente destacar. Cuando paralelizamos un conjunto de instrucciones (llamaremos bloque paralelo), como muestra la Figura 3-34, mediante la sentencia par {} la máquina de estados divide su ejecución en ramas, tantas como se tengan que ejecutar. En estas ramas cada una tiene sus propias variables de estado con codificación one hot pero el HCC puede optimizar las variables de estado si las ramas por su naturaleza se ejecutan en forma sincronizada.

```
Par {
        Rama 1;
        Rama 2;
        ...
        rama n;
}
```

**Figura 3-34** Estructura de la sentencia *par*

Entendemos como ramas sincronizadas aquellas que el compilador puede determinar en tiempo de compilación cuantos y cuáles ciclos de reloj son necesarios para su ejecución y por lo tanto se utilizan las mismas variables de estado para controlar sus estados. Cada rama puede ser una instrucción o un bloque de instrucciones indicado entre llaves ("{}").

Veamos algunos ejemplos:

| Ramas sincronizadas | Ramas no sincronizadas |
|---|---|
| par {<br>a = a + 1;<br>b = b + 1;<br>}; | par {<br>while (salir) {<br>a= a+1;<br>salir =a.4; }<br>{ b = b +2;<br>c = c << 2; }<br>}; |
| par {<br>{ a = a + 1;<br>b = 1;};<br>c = 2;<br>}; | par {<br>for (i=1 ; i<9 ; i=i+1) {<br>cout ! i; }<br>{ a = 23;<br>b = b +1; }<br>} |

**Figura 3-35** Ejemplos de composición paralela con ramas sincronizadas y no sincronizadas

Para terminar la ejecución del bloque paralelo y continuar con el resto del programa, es necesario que hayan terminado de ejecutarse todas las ramas, el caso más interesante es cuando no son ramas sincronizadas. Aquí se genera una lógica encargada de la sincronización, que consta principalmente de unos flip-flop que recuerdan las ramas que van terminando y una lógica que genera la variable de estado de la 1er instrucción fuera del bloque paralelo.

Lo primero que hace el HCC es determinar cuales ramas son las que hay que sincronizar. Éstas son las que tengan sentencias de bucles de instrucciones y/o de ejecución condicional. Luego para más de 2 ramas a sincronizar se utilizan los circuitos que describiremos a continuación.

Para cada rama se utiliza la señal de la variable de estado de la última instrucción del bloque (llamaremos $S_{\text{última rama}}$), salvo que ésta sea WHILE, FOR o alguna otra para la cual se tiene una señal que indica que debe ejecutarse la primer instrucción fuera del bloque principal (bloque de esa sentencia en particular) pues no se cumple una condición de test (a esta señal la llamaremos $S_{\text{no test}}$). Según la naturaleza de la señal de la variable de estado se utilizan los siguientes circuitos que se encargan de recordar que esa rama a finalizado y además para el caso de $S_{\text{última rama}}$ ajustar su temporización.

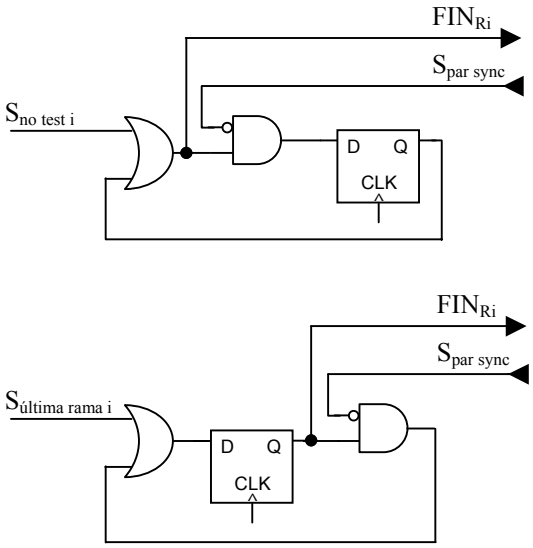



**Figura 3-36** Circuito de control de bloques paralelos. Parte 1.

La señal $S_{\text{par sync}}$ es utilizada como variable de estado e indica que terminó el bloque paralelo y se utiliza para el control de la primer sentencia luego del par {}. Es generada a partir de las señales $FIN_{Ri}$ como un "and" de todas ellas.

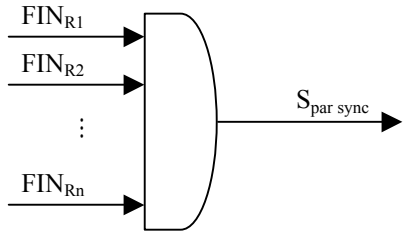



**Figura 3-37** Circuito de control de bloques paralelos. Parte 2.

Veamos el caso particular de 2 ramas pues el compilador genera otros circuitos distintos a los vistos anteriormente, pero su cometido es el mismo.

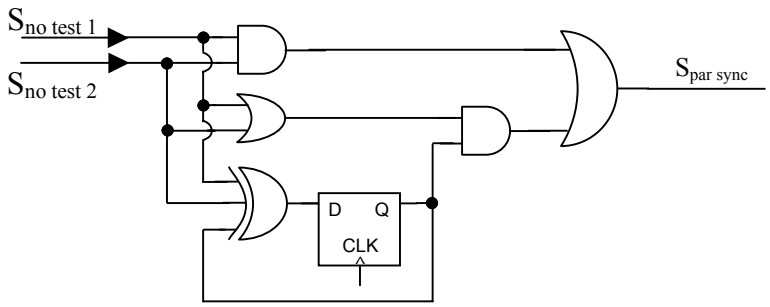



**Figura 3-38** Circuito de control de bloques paralelos. Caso particular.

Si las señales fueran del tipo $S_{última\ rama}$ entonces se la precede de un flip-flop y luego, la salida de éste, se conecta como antes.

## 3.2 Transmogrifier C (TMCC versión 3.1 – Enero 96)

### 3.2.1 Instalación, opciones de compilación

Esta herramienta es distribuida con sus fuentes y por lo tanto puede ser compilada para muchas plataformas siendo muy portable. En particular las pruebas la realizamos en equipos de arquitectura Intel x86 con sistemas operativos Linux y también en arquitecturas Sparc con SunOS.

Para su compilación es necesario un compilador de C, por ejemplo GCC, y dos utilidades: yacc y lex, pudiéndose utilizar sus semejantes en GNU que son bison y flex.
A lo largo del desarrollo del proyecto compilamos con versiones diferentes del compilador GCC no encontrando ningún tipo de problemas en la generación de los ejecutables; y en particular las últimas pruebas las realizamos con el EGCS versión 2.91 sobre un PC con linux Red Hat versión 6.1.

Para compilar se procedió como siempre se hace en estos casos, se corrió el programa gmake y luego de finalizada la compilación se instaló utilizando el comando gmake install. Fue necesario hacer un pequeño ajuste por cuestiones de orden, donde se modificaron las rutas para el ejecutable, librerías y man pages, estos cambios se realizaron sobre el archivo “makefile”. También en él se ajustaron unas variables internas de forma que compilara utilizando las herramientas flex y bison en lugar del lex y yacc.

El TMCC antes de compilar un programa invoca a un preprocesador de C, llamado comúnmente cpp. Las rutas de búsqueda de este ejecutable están definidas en sus fuentes, en particular en el archivo tmcc.y, y son las siguientes: “. /usr/lib/cpp", "/usr/bin/cpp", "/usr/ccs/lib/cpp" y "/lib/cpp". En caso de no contar con un preprocesador puede utilizar una opción de líneas de comando que instruye al TMCC que compile el programa tal cual viene.

Veamos las distintas opciones ofrecidas por el compilador TMCC [6] :

**Tabla 3-4** Opciones de línea de comando para el compilador TMCC

| | |
|---|---|
| -S | Don’t run ppr |
| -v | Save extra ppr output files |
| -On | Optimizer level |
| -c | Don’t use internal clock |
| -p parttype | Specify Xc4000 parttype |
| -dverbose | Estimate circuit size and speed |
| -dn | Set debug level n |
| -fno-carry-select | Use "ripple carry" adders |
| -fcarry-select | Use "carry select" adders (default) |
| -Tnnn | Thread number |
| -target netlist_format | Set output format |
| -a | Don't run cpp |
| -Uxxx | CPP compiler flag |
| -Dxxx | CPP compiler flag |

-S , -v

Estas opciones controlan el uso de herramientas de Place and Route (ppr) luego de generado el XNF. Si queremos que el compilador se detenga luego de generar el XNF usamos la opción –S (opción utilizada por nosotros pues no contábamos con ninguna herramienta de este tipo corriendo en ambientes Unix). La opción -v indica que se guarden los archivos extra generados por las herramientas de ppr.

-On

Esta opción controla el grado de optimización en la generación de los circuitos, si no se especifica por defecto lo hace.
**Obs.** Si bien figura la opción –O no esta implementada ninguna variación en el código en función de ella por lo tanto no se puede controlar el nivel de optimización.

-c

El compilador por defecto usa el oscilador interno más lento (15 Hz) de los chips de la familia XC4000 como reloj del circuito. La opción -c suprime esto y permite utilizar un reloj externo al chip.

-p parttype

Especifica el chip de la familia XC4000 a usar.

-dverbose , -dn

Usando la opción -dverbose el compilador imprime en pantalla una estimación del tamaño del circuito y la velocidad alcanzada así como la cantidad de LUTs y FlipFlops usados. El estimado no es fiable ya que las herramientas de Place and Route pueden encontrar otro camino para implementar el circuito. La opción –dn fija el nivel de información de debug que se imprime en pantalla mientras realiza la compilación.

-fno-carry-select , -fcarry-select

Estas opciones indican que tipo de sumadores va implementar el compilador, la primera indica que se usen sumadores de acarreo propagado o "ripple carry" mientras que la segunda (por defecto) indica que se usen sumadores del tipo "carry select" (ver Apéndice A).

-Tnnn

En caso de armar un diseño con varios hilos de control (threads), al compilar con esta opción se indica el número de hilo a usar en la formación de los nombres internos del circuito para las variables del programa así al juntar varios circuitos (varios hilos) no existan dos variables con el mismo nombre. Más adelante explicaremos detalladamente como armar los diseños de esta forma.

-target output_format

Especifica el formato de los archivos de salida. La opción por defecto es xc4000gates que usa compuertas AND, OR e INV. Otras opciones son : xc4000roms expresa cada LUT como una ROM de 16x1; xc4000eqns expresa cada LUT en forma booleana; flex8000 es similar al formato xc4000gates con la diferencia que no utiliza CE (clock enable) en los flip-flop. Todas estas opciones producen archivos de salida del tipo XNF.

-a , -D , -U

La opción –a evita la ejecución del preprocesdor. Las otras dos opciones son pasadas directamente al cpp y se encargan de predefinir (-D) y no predefinir (-U) como un macro a un determinado nombre.

### 3.2.2 Sintaxis del lenguaje

Veremos la sintaxis necesaria para que un programa escrito en C se pueda compilar con Transmogrifier C. Mencionaremos las principales características del programa principal y funciones, las variables y sus puertos con el mundo exterior.

**Sumario del lenguaje soportado por tmcc**
Es un subconjunto del lenguaje ANSI C con extensiones que nos permiten especificar controlar algunas características del diseño y señales generadas.

Trabaja con variables de tipo entero (integer), constantes y expresiones (aritméticas y booleanas), posee sentencias tipo IF, WHILE y soporta llamadas a funciones con pasaje de parámetros por valor. Todas las directivas de los preprocesadores de C son soportadas incluyendo la utilización de macros.

**Tabla 3-5** Resumen de las instrucciones soportadas por Transmogrtifier C

<table>
<tr><th colspan="2">Operadores aritméticos</th><th colspan="2">Operadores Relacionales</th><th>Tipos de Datos</th><th>Sentencias</th></tr>
<tr><td>Con signo</td><td>Sin signo</td><td>Con signo</td><td>Sin signo</td><td rowspan="2">int</td><td rowspan="5">while () {…}<br>if () {…} else {…}<br>break<br>return<br>{ ; }</td></tr>
<tr><td>>><br>>>=</td><td>+<br>++<br>+=<br>-<br>--<br>-=</td><td>><br><<br>>=<br><=</td><td></td></tr>
<tr><th colspan="2">Operadores de bits</th><th colspan="2">Operadores lógicos</th><th>Funciones</th></tr>
<tr><td colspan="2">^<br>^=<br>~<br><<<br><<=<br>|<br>|=<br>&<br>&=</td><td colspan="2">||<br>&&<br>!<br>!=<br>==</td><td>outputport (...)<br>inputport (...)<br>bus_port (...)<br>bus_idle (…)<br>portflags (…)</td></tr>
</table>

**Nota:** Aparecen resaltados aquellas instrucciones que no pertenecen al ANSI C o cuya funcionalidad es diferente.

Y destacamos particularmente que NO soporta:

- multiplicaciones, divisiones
- punteros, arreglos, estructuras, char, string
- recursión

### Extensiones

Una extensión al lenguaje C nos permite definir el ancho de las variables declaradas.
Las señales de entrada y salida del circuito están asociadas con variables del programa por medio de su definición utilizando las llamadas a las funciones *inputport*, *outputport* y *bus_port*. Todas las variables declaradas de esta forma son conocidas como variables de tipo puerto (o port) y una cuarta función llamada *porflags* permite configurarles propiedades típicas de las señales en los circuitos eléctricos.

### Estructura del programa principal

La estructura de un programa escrito en lenguaje C para que pueda ser compilada por el Transmogrifier C debe ser la siguiente:

```
        Declaraciones externas
        main ()
{
        declaraciones internas

        sentencias
}
```

**Figura 3-39** Estructura del programa principal para compilar con Transmogrifier C

En cualquier punto del programa se puede utilizar la directiva #pragma intbits que permite definir el número de bits de las variables que se declaren posteriormente. Hay que tener cuidado pues la directiva afecta a todas las variables que se declaren posteriormente sin importar si son variables locales a una función o del programa principal. Otro detalle sobre la directiva es que tiene que ser puesta sin indentación o sea tiene que estar al comienzo de la línea.

Ahora pasaremos a detallar cada una de las secciones mencionadas en la figura.

### Declaraciones externas

Aquí se pueden definir las funciones a utilizar en el programa principal o desde otra función. La estructura de la declaración de una función es la siguiente:

```
        #pragma intbits ANCHO
        NOMBRE_FUN ( NOMBRE_PARAM , ... )
        declaración de tipo de los parámetros de entrada
{
        declaraciones internas

        sentencias

        return ( expresión o nombre de variable ) ;
}
```

**Figura 3-40** Estructura de la declaración de las funciones

La directiva #pragama intbits que se indica al comienzo, si bien es opcional, es muy conveniente para indicar el ANCHO en bits de los datos que retorna la función, hay que tener presente que esta directiva afecta los anchos de las variables declaradas posteriormente en el programa (salvo que se coloque otra directiva #pragama intbits).

En la *declaración del tipo de los parámetros de entrada* deben declarase todos los parámetros de entrada del tipo int (que es el único aceptado), y también es conveniente indicar el número de bits de estas variables con la directiva #pragma intbits.

En las *declaraciones internas* se definen las variables locales a utilizar por la función y obviamente pueden definirse de cualquier número de bits.

Es conveniente colocar al final del bloque de *sentencias* la instrucción *return()* que permite a la función devolver valores al programa que la invocó. Hemos encontrado problemas en el compilador si se desea incluir más de una sentencia *return()* en la misma función.

**Declaraciones internas**

Aquí se declaran las variables locales al programa principal, es conveniente indicar el número de bits de las variables y también definir los puertos mediante el juego de instrucciones que veremos a continuación. Estas recomendaciones dan más claridad a la lectura del código y evitan cometer errores posteriores.

| |
|---|
| outputport ( *NOMBRE* , string o entero , string o entero, ... ) |
| inputport ( *NOMBRE* , string o entero , string o entero, ... ) |
| bus_port ( *NOMBRE* , string o entero , string o entero, ... ) |

**Figura 3-41** Estructura de declaración de puertos y buses

Donde *NOMBRE* tiene que ser el nombre de una variable declarada previamente. Los string o enteros son opcionales y sirven para especificar el nombre o número del pin en el chip al que se le van a asignar los bits de dicha variable, comenzando la lista por el menos significativo.

Como su nombre lo indica la sentencia outputport() define los puertos de salida, inputport() los puertos de entrada y bus_port() define pines bidireccionales con salida en tercer estado, muy útiles para ser utilizadas compartiendo recursos con otros dispositivos.

También es conveniente en esta sección, en caso de ser necesario, ajustar las propiedades de los puertos, esto se realiza mediante la instrucción portflags() y su estructura es la siguiente:

| |
|---|
| portflags ( *NOMBRE* , *constante* ) ; |

**Figura 3-42** Estructura de la definición de propiedades de un puerto.

Donde *NOMBRE* debe ser una variable que previamente se haya declarado como un puerto. La constante indica las características eléctricas de las señales correspondientes a ese puerto. En la siguiente tabla veremos el valor constante admitido por el compilador, un nombre sugerido y su efecto en el circuito final.

**Tabla 3-6** Valores admitidos por la función *portflags()*

| Valor | Nombre sugerido | Función |
|---|---|---|
| 0 | PORT_WIRE | Considera al puerto tal como si fuera un cable (o señal interna al chip). Útil para la comunicación entre hilos (threads). |
| 1 | PORT_PIN | Indica que este puerto necesitará pines para salir del chip (en VHDL es declarado en la entity del diseño principal) |
| 2 | PORT_REGISTERED | Obligará al compilador a colocar un registro de flip-flop para este puerto, controlado con el reloj global del sistema. |
| 4 | PORT_PULLUP | Activa las resistencias pullup internas, propias de los chips de Xilinx. |
| 8 | PORT_PULLDOWN | Activa las resistencias pulldown internas, propias de los chips de Xilinx. |

Estas opciones se pueden combinar libremente (sumando sus valores) y el compilador indicará en el caso de existir algún conflicto o incoherencia.

Los valores por defecto de estas propiedades dependen del tipo de puerto que se trate, tenemos que para los puertos de salida son (PORT_PIN | PORT_REGISTERED), los puertos de entrada son solamente PORT_PIN.

Es importante notar que esta función no se aplica a los buses, éstos por defecto reúnen las características de los dos anteriores, a la entrada es PORT_PIN y a la salida es además PORT_REGISTERED.

**Sentencias**

En esta sección se pueden utilizar cualquiera de las sentencias admitidas por el Transmogrifier C y enumeradas en la Tabla 3-5, siguiendo la sintaxis propia del lenguaje ANSI C [3]. Aquí mencionaremos solamente las que no son propias del lenguaje y no fueron abordadas en puntos anteriores.

Se cuenta con la función:

```
bus_idle ( NOMBRE ) ;
```

**Figura 3-43** Estructura de la sentencia *bus_idle()*

Esta función tiene por objetivo colocar al puerto tipo bus con su salida en tercer estado y lo deja en modo entrada o lectura. Cuando se declara un bus, automáticamente queda en modo entrada de datos, luego al hacer una escritura pasa al modo salida hasta que se ejecute la instrucción bus_idle.

### 3.2.3 Circuito generado

**Timing y estados**

La semántica del Transmogrifier C asume que existe un esquema de reloj en el cual hay solo un reloj y todos los flip-flop cambian en el mismo flanco.
Las sentencias de asignación y las sentencias IF corresponden a expresiones de lógica combinatoria. Los valores de estas expresiones se almacenan en flip-flop.

El TMCC genera máquinas de estado secuenciales tipo Mealy donde cada estado es representado por un flip-flop, esta codificación recibe el nombre de **one hot (o uno activo)**, esto produce mejores resultados en arquitecturas ricas en flip-flop como lo son las FPGA, pues además se requiere menos lógica combinatoria para determinar el cambio de estado.

Un nuevo estado de esta máquina es comenzado solamente en uno de los siguientes puntos del programa:

- al comienzo de un ciclo WHILE
- en los llamados a una función

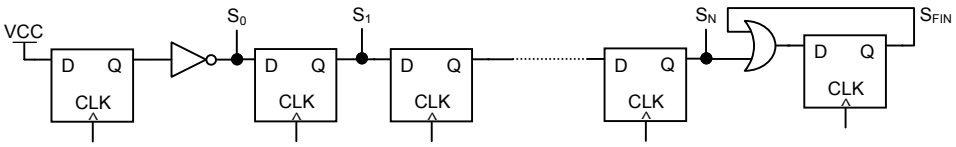
Es importante notar que se mantiene la semántica de C a pesar de que muchas sentencias son asignadas a un mismo estado de la máquina generada. O sea que las sentencias se ejecutan en el orden correcto, las asignaciones a variables ocurren inmediatamente pero los cambios no se ven reflejados en los puertos de salida hasta el próximo flanco de reloj, para salidas PORT_REGISTERED. Veamos unos ejemplos:

| Ejemplo 1 | Ejemplo 2 |
|---|---|
| temp = a;<br>a = b;<br>b = temp; | a = 10;<br>a++;<br>b = a; |

**Figura 3-44** Ejemplo de asignaciones para TMCC

El ejemplo 1 intercambia los valores de “a” y “b” como lo hace cualquier programa en C. El intercambio ocurre en un ciclo de reloj y no se generan flip-flops u otro tipo de circuito para la variable “temp". En el ejemplo 2 vemos que también todas estas asignaciones ocurren en un solo período y que al final del mismo la variable “b” contiene el valor 11.

En la Figura 3-45 vemos un diagrama y ejemplo de código de una máquina de estados sencilla correspondiente a un programa sin divisiones del flujo (loops, IF).




```
main( ){
#pragma intbits 4
int ent, sal;
int a, b, c;
inputport(ent);
outputport(sal);
    a = ent;   // estado S0
    while(0){ };
    b = ent;   // estado S1
    while(0){ };
    c = a - b;   // estado S2
    while(0){ };
    sal = c;   // estado S3
    while(0){ };
    c = a + b;   // estado S4
    while(0){ };
    sal = c;   // estado S5, sería el SN del diagrama anterior
}
```

**Figura 3-45** Diagrama y ejemplo de un programa sin alteraciones en el flujo.

## Sobre las variables utilizadas

Al final de la primera pasada de compilación se tiene una lista de los identificadores de las diferentes variables, los estados donde éstas son modificadas y el nuevo valor asignado. En el armado del circuito final un registro (conjunto de flip-flop tipo D) es creado por cada identificador, la lista de estados y sus nuevos valores es utilizada para crear un SELECTOR que selecciona el correcto valor de la variable en cada estado del sistema y es conectado a la entrada del registro de flip-flop como se puede apreciar en la Figura 3-46. Como los flip-flop están conectados al reloj global en cada flanco se guarda un valor nuevo, entonces para los estados donde la variable no es modificada se realimenta la salida del registro de la variable a una entrada del SELECTOR que llamaremos $D_{default}$ y que es conectada a su salida cuando ninguna señal de control esta activa.

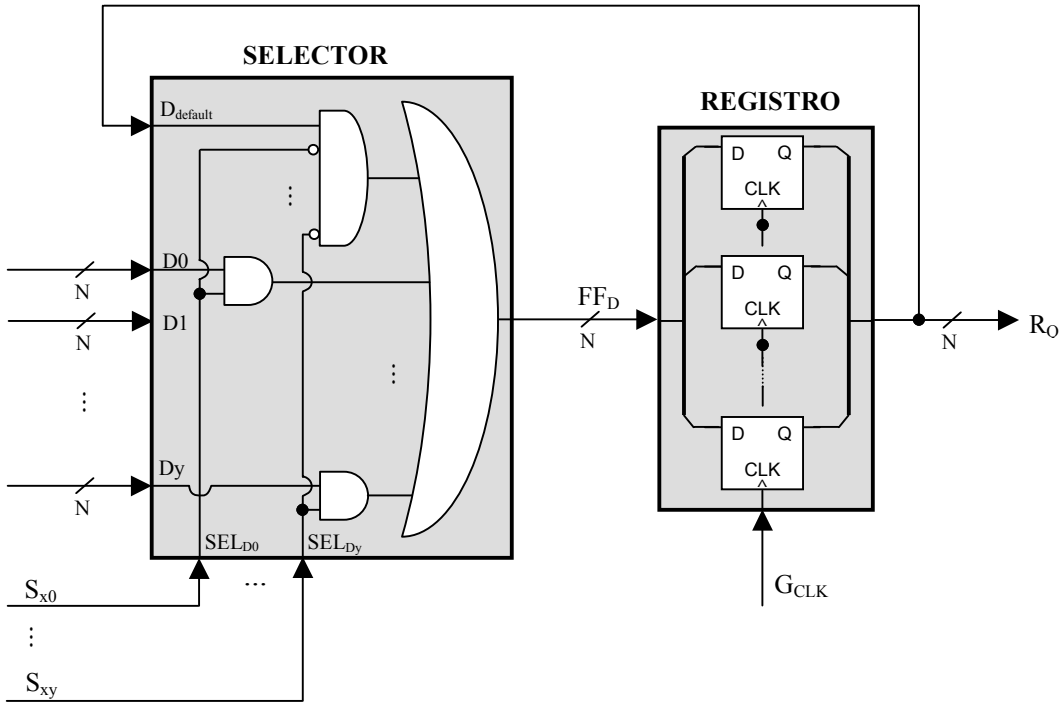



**Figura 3-46** Circuito de manejo del registro de una variable

Donde $S_{xi}$ son las variables de estado de los estados donde se modifica la variable.

En el caso que el compilador detecte que la variable es actualizada solamente en un estado entonces no genera el SELECTOR y lo que se utiliza es el Clock Enable del flip-flop y a su entrada conecta directamente los datos.

**Sobre los puertos**

Para el caso particular de las variables que son definidas como puertos de entrada o salida tenemos que los circuitos que se generan son similares a los vistos para las variables comunes, pero con las salvedades que describiremos a continuación pues dependen de las características definidas para el puerto.

Como vimos anteriormente las propiedades definidas con *portflags* afectan la generación o no del registro de flip-flop de la variable, la asignación de pines del chip y otras propiedades eléctricas como pullup y pulldown en las líneas. Esto último es válido para familias de Xilinx pues estas tienen resistencias programables en los boques de salida (IOB).

Si la salida del puerto no lleva un registro, el selector es conectado directamente a los pines del chip (para PORT_PIN) o utilizado en el circuito generado para otros hilos del programa principal (para PORT_WIRE), en ambos casos la lógica combinatoria del selector queda más sencilla pues no es necesario realimentar ningún valor para usar como salida por defecto, que por su construcción devolverá un 0 en las líneas.

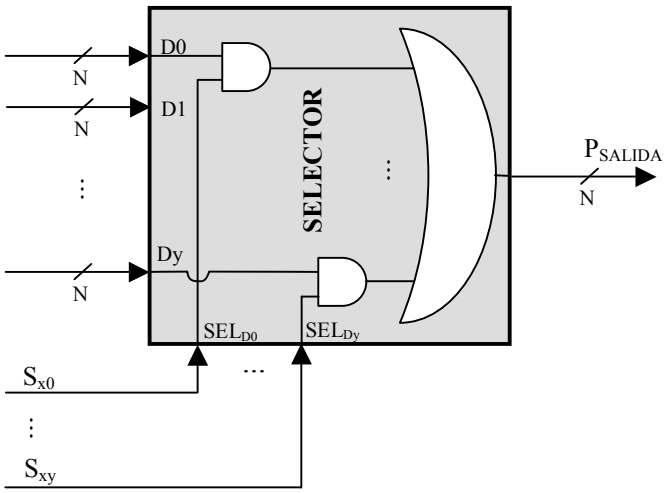



**Figura 3-47** Circuito de manejo de puerto de salida

## Sobre los buses

Para este caso particular de variables tenemos que genera como era de esperarse pines bidireccionales que internamente se dividen en dos señales, una de entrada y otra de salida controlada con un buffer triestado. La lógica combinatoria de la señal de control del buffer se genera en función de las variables de estado donde se realizan escrituras al bus y los estados donde se encuentra la instrucción *bus_idle()* (que vuelve el bus a tercer estado).

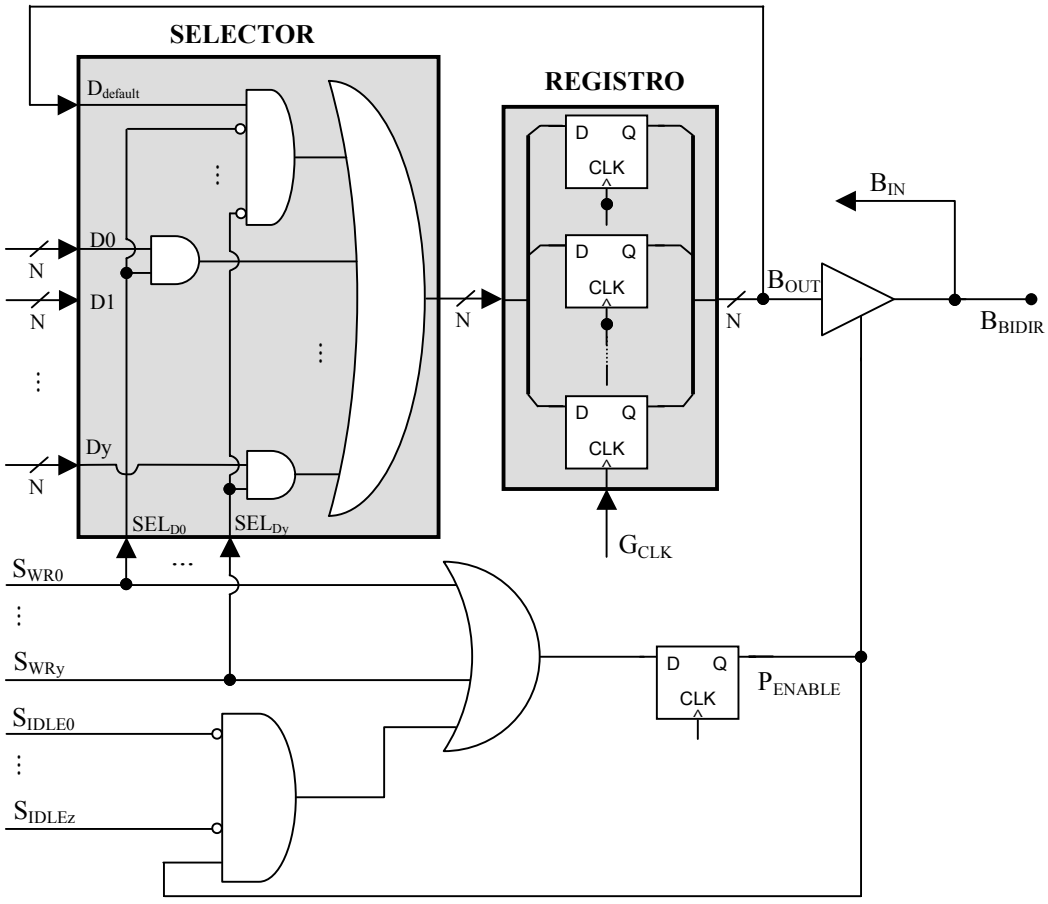



**Figura 3-48** Circuito de manejo de un bus

En el esquema de la Figura 3-48 podemos apreciar el circuito generado para el caso de un bus con sus propiedades por defecto. En él las variables de estado $S_{WR0}$ a $S_{WRy}$ representan los estados donde se realizan escrituras al bus y las variables de estado $S_{IDLE0}$ a $S_{IDLEz}$ representan los estados donde se ejecuta la instrucción bus_idle( ). Señalamos que el compilador no permite que los buses sean declarados como PORT_WIRE.

**Sobre los bucles de instrucciones (iteraciones)**

WHILE

Para el caso del WHILE, se crean dos estados, un estado correspondiente al comienzo del bloque principal propiamente y que es activado si la condición de test del WHILE es verdadera, y el otro estado corresponde a la salida del bloque debido a que la condición de test sea falsa. Al estado del bloque principal le pueden seguir tantos estados como sean necesarios para armar todo el cuerpo del WHILE.

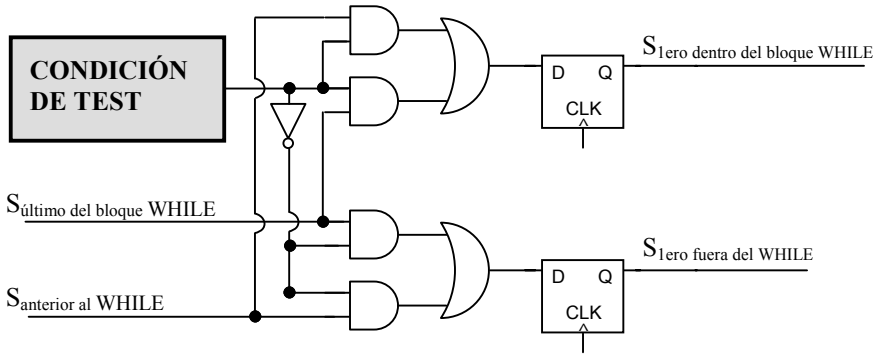



**Figura 3-49** Circuito de control del WHILE

En la siguiente figura vemos un programa de ejemplo en el que se muestran los estados mencionados en el diagrama anterior.

```
main( ){
#pragma intbits 4
int ent, sal;
int a, b, c;
inputport(ent);
outputport(sal);
    a = ent;   // estado S0
    while(0){ };
    b = ent;   // estado S1
    while(0){ };
    c = a - b;   // estado S2 o Santerior al WHILE
    while(c){   // la condición de test es c > 0
        c = c - 1;   /* estado S3 o
S1ero dentro del bloquel WHILE */
        while(0){ };
        sal = c;   // estado S4 o Súltimo del bloque WHILE
    };
    c = a + b; // estado S5 o S1ero fuera del WHILE
    while(0){ };
    sal = c;   // estado S6
}
```

**Figura 3-50** Ejemplo de programa con una instrucción WHILE (con condición).

**Sobre la ejecución condicional**

IF

Analizaremos tres casos de interés: 1) no hay cambios de estado en las ramas (o en la rama) del IF 2) en una de las ramas no hay cambios de estado y en la otra sí 3) hay cambios de estado en las dos ramas.

En el primer caso el IF implementa un pequeño selector en el que la condición decide el valor que se le pasa al selector de las variables que sufren cambios dentro de la sentencia, en este último es que se usa la variable de estado ($S_0$) dentro del que está el IF. Veamos dos ejemplos genéricos y el circuito generado para ambos.

```
.. // estado S0
if (condición){
        c = C1;
}
else{
        c = C0;
};
..
```

```
.. // estado S0
c = C0;
if (condición){
        c = C1;
};
..
```

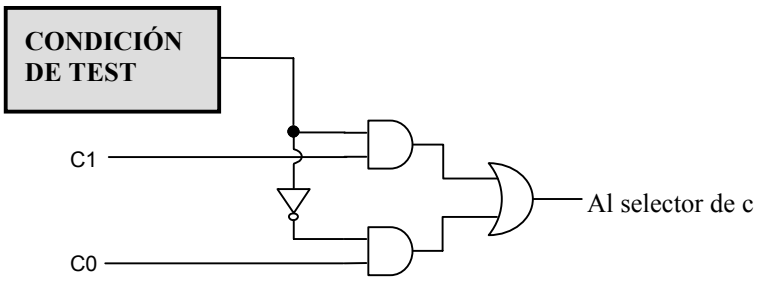



**Figura 3-51** Ejemplos de uso de IF y circuito generado para ellos.

En el segundo caso (una rama con más de un estado) el IF actúa sobre los estados. Veamos un ejemplo de la situación y la lógica de estados que se genera para él.

```
.. // estado S0
if (condición ){
      c = C1;  // estado S0 , se usa Strue
      .. // hay N cambios de estado
      c = CN;  // estado SN
}
else{
       c = D;  // estado S0, se usa Sfalse
}
c = E;  //estado SP, coincide con SN o con Sifthick según la condición
..
```

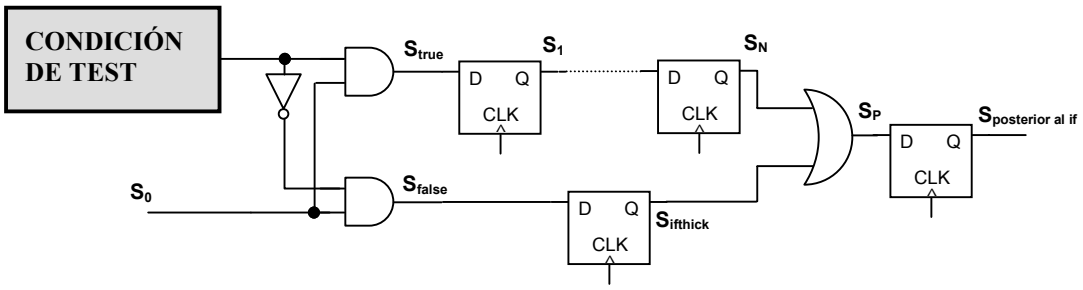



**Figura 3-52** Ejemplo y circuito generado para IF con estados en una rama.

Como se aprecia en la figura aunque no hay estados en el ELSE el compilador genera un estado ($S_{ifthick}$). Si no se agregara este estado, $S_P$ coincidiría con $S_0$ cuando no se da la condición y con $S_N$ cuando se da. Entonces el próximo estado se determinaría de la siguiente forma:

$$S_X = S_0 \ \& \ !condición \ \mid \ S_N \ \& \ condición$$

Pero si se modificara la condición luego de evaluada esto podría ocasionar un cambio anticipado de estado (para el ejemplo, si se modifica en la rama THEN) o que no cambie el estado cuando debería hacerlo.

El tercer caso (más de un estado en cada rama) es similar al anterior, solo que ya no es necesario agregar el estado $S_{ifthick}$. En la siguiente figura mostramos el circuito generado.

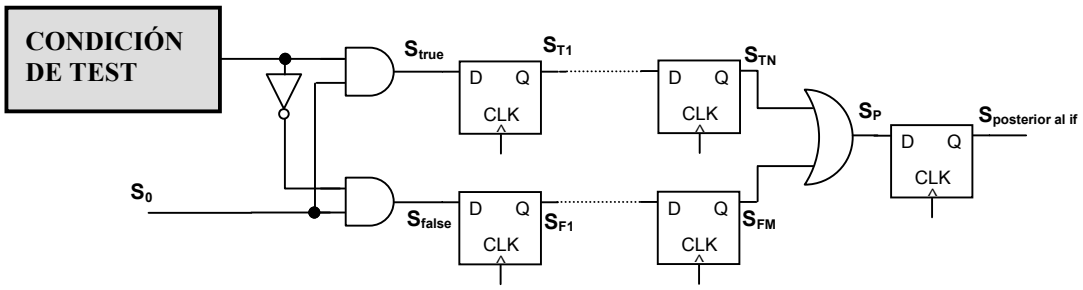



**Figura 3-53** Circuito generado en el caso de que las dos ramas del IF tengan más de un estado.

**Sobre las llamadas a funciones**

Las llamadas a funciones provocan la generación de un nuevo estado, simplificando la compilación de la función, previniendo que pueda ser utilizada dos veces en el mismo período de reloj y permitiendo que dentro de las funciones se puedan utilizar variables globales sin preocuparse si quien la llamó modifica estos valores en este estado de la máquina. En el estado anterior a la llamada de la función se ponen en la entrada D de los registros de las variables definidas como parámetros de entrada, de esta forma cuando se pase al estado inicial ya se tienen los valores cargados.

Las funciones son implementadas como máquinas de estados, están compuestas al menos por un estado definido, estado inicial (o $S_{init}$). Al final se activa una señal (llamada $S_{final}$ ) que indica que indica que se está ejecutando el último estado. La máquina de estados del programa principal al hacer la llamada a la función activa el estado inicial y se posiciona en un estado (llamado $S_{calling}$) esperando que la función alcance su estado final.

La Figura 3-54 es un esquema del circuito correspondiente a los llamados a funciones.

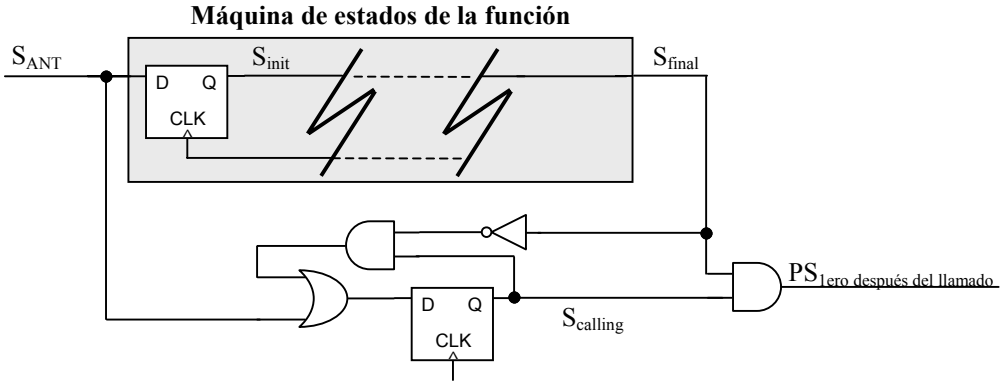



**Figura 3-54** Circuito de control del llamado a función

Donde las variables de estado $S_{ANT}$ corresponde al estado de la máquina principal anterior al que se realiza el llamado. La señal $PS_{1ero\ después\ del\ llamado}$ va a la lógica de control que setea la variable de estado del próximo estado, por tanto si éste es determinado por otra llamada a función entonces será igual a $S_{ANT}$ y si es determinado por un WHILE será $S_{anterior\ al\ WHILE}$ .

Veamos un ejemplo mostrando el uso de las variables de estado para cada línea del código.

| main | función |
|---|---|
| …<br>a = 10; *// se usa $S_{ANT}$*<br>c = fun(a); *// se genera $S_{calling1}$*<br>z = c++; *// se usa $PS_1$ = $S_{calling1}$ & $S_{final}$*<br>c = fun(z); *// se genera $S_{calling2}$*<br>m = c--; *// se usa $PS_2$ = $S_{calling2}$ & $S_{final}$*<br>… | #pragma intbits 8<br>fun(a)<br>int a;<br>{<br>a <<= 2; *// se usa $S_{init}$*<br>while(0){ }; *// inserto nuevo estado*<br>return(~a); *// se usa $S_{final}$*<br>} |

**Figura 3-55** Ejemplo del llamado a función

Cabe destacar que no se puede llamar dos veces (o más) a una función en la misma expresión. Pese a que se generan los estados correctos no se almacenan los resultados intermedios en ningún registro por lo que se sobrescriben los del primer llamado al llamar por segunda vez a la función.

### Sobre los operadores implementados

Sumas y Restas

El TMCC implementa por defecto sumas y restas con la técnica llamada carry select (ver Apéndice A), provoca diseños más rápidos pero más costosos en área, se le puede indicar explícitamente (con las opciones de línea de comandos vistos anteriormente) que implemente las sumas y restas utilizando ripple carry, reduciendo así el tamaño y lamentablemente la velocidad.

Con respecto a la reutilización del hardware vemos que el TMCC no reutiliza el hardware generado para la realización de las sumas o restas, que en principio podría utilizarse si se tratara de sumadores de igual número de bits e implicara manejar datos de señales que son modificadas en estados diferentes.

Comparadores

Veamos ahora la implementación de los operadores de comparación. Analizamos el caso del menor estricto, por ejemplo "a < b". El circuito que genera el compilador es parecido al realizado en la resta "a - b", pero con simplificaciones pues solamente calcula el valor del último bit y los carry de los bits anteriores. Luego en función de estos datos determina si la condición es verdadera o falsa.

Cuando se quiere hacer la comparación "a <= b", aquí la lógica generada es muy parecida pero se agrega además la comparación de que los bits de "a" y "b" sean iguales. Para los casos de las comparaciones ">" y ">=" lo que se genera es similar a lo anterior, para el ">" es la misma lógica que para "<=" y considera la condición negada. Para ">=" es la misma lógica que para "<".

Por lo tanto es mejor utilizar las comparaciones "< " o ">=" que generan menos lógica combinatoria.

**Multiples hilos de control (threads)**

Un programa simple en TMCC tiene únicamente un circuito de control de la máquina de estados, que comienza al principio de la rutina main(). Diseños con más de un circuito de control corriendo en paralelo se pueden construir compilando por separado un programa por cada hilo (thread). Cada programa tiene su propia rutina main() y corre independientemente de los otros, compartiendo solamente la señal de reloj.

Cuando dos threads se quieren comunicar, ambos declaran una variable de tipo puerto con el mismo nombre, utilizando *portflags()* para indicar que la variable no tenga pins externos (PORT_WIRE) y que por lo tanto es una conexión a otra parte del circuito en el mismo chip.

Es necesario que al momento de compilar cada thread se indique el número de hilo que le corresponde pues así evitamos problemas con los nombres de las señales internas generadas.

Por ejemplo si tenemos un programa principal llamado A.c y dos hilos que se encargan de controlar el interfaz de comunicación con el puerto serial y paralelo, llamaremos SERIAL.c y PARAL.c. La forma de compilar estos tres diseños en un solo chip es la siguiente:

```
tmcc –T1 –c –S  SERIAL.c
tmcc –T2 –c –S  PARAL.c
tmcc –S  A.c  SERIAL.xnf  PARAL.xnf
```

**Figura 3-56** Ejemplo de compilación de un diseño en threads

# ¿ Cómo compilar ?

Como ya vimos en el capítulo anterior los compiladores generan circuitos coherentes con el código C de entrada. En este capítulo veremos los problemas a la hora de compilar y como implementar un ejemplo en la placa UP1 con ambos compiladores.

## 4.1 Primeros problemas

En este punto veremos cuales fueron los problemas encontrados al compilar los primeros programas que imposibilitaban la sintetización de los diseños y por consiguiente su posterior simulación.

La síntesis para chips de Xilinx las hicimos con dos programas incluidos en el paquete Foundation v.1.5; en el caso de archivos XNF usamos la herramienta Design Manager, en el caso de archivos VHDL usamos el Program Manager.

La síntesis para chips de Altera las hicimos con el software Max+Plus II versión 10.1. Este acepta el formato XNF sin embargo al intentar sintetizar nuestros circuitos (en Handel C y Transmogrifier C) el programa se colgaba. En vista de esto intentamos utilizar el Leonardo Spectrum para la síntesis pero este no permite compilar para Altera cuando el archivo origen del diseño es un XNF. Por estas razones comenzamos a usar un traductor de XNF a VHDL en el caso de Transmogrifier C ya que este compilador no ofrece otras opciones y con este formato el Max+Plus II funciona bien.

### 4.1.1 Handel C

Se probaron ejemplos con los posibles formatos de salida, estos son: XNF, VHDL y EDIF.

Los archivos generados en el formato EDIF no pudimos sintetizarlos en ninguna de las herramientas antes mencionadas por errores en la especificación del formato. En la documentación del HCC [2] se indica que la generación de EDIF se encuentra en fase experimental para esta versión del compilador.

Los archivos en formato XNF no presentaron complicaciones (para Xilinx), salvo los nombres de algunas señales en el simulador (Logic Simulator).

Para los archivos generados en VHDL encontramos los siguiente problemas:

- Los componentes declarados no son parte de librerías estándar, por lo que hubo que crear una librería especial (ver Apéndice F) con los siguientes componentes:

| | | | |
|---|---|---|---|
| LIBDFF | LIBOR2 | LIBAN2 | LIBXOR |
| LIBDFFR | LIBOR3 | LIBAN3 | LIBLOG2 |
| LIBINV | LIBOR4 | LIBAN4 | |

  La mayoría de los nombres sugieren la funcionalidad del componente así para LIBOR3 creamos una entidad que lo que hace es:

```
y <= a or b or c;
```

  donde "y" es la salida y "a", "b" y "c" las entradas. Los componentes LIBAN*x* y LIBXOR son análogos. LIBDFF es un flip-flop tipo D, sensible al flanco de subida del reloj, sin Preset, Reset o ClockEnable, LIBDFFR agrega un Reset activo bajo. Las salidas del componente LIBLOG2 se conectan a VCC y GND por lo que en dicha entidad lo que hacemos es asignarle '1' y '0' a estas salidas.

- Mal declaración del componente LIBLOG2 en la declaración de la arquitectura. El compilador genera:

```
Component LIBLOG2
        port ( VDDLOG,GNDLOG : out std_ulogic;   ← sobra este punto y coma
               );
end component ;
```

**Figura 4-1** Ejemplo de declaración de componente LIBLOG2

Este mismo componente se instancia mal ya que no se indica su nombre (LIBLOG2) en la instanciación (declaración de la arquitectura del circuito):

```
CSupply :  port map ( VDDLOG => VCC , GNDLOG => GND ) ;
```

**Figura 4-2** Ejemplo de instanciación de componente LIBLOG2

- Un problema que es particular del Max+Plus II es que este no soporta señales del tipo "std_ulogic" por lo que hay que cambiar todas las declaraciones de señales y puertos reemplazándolo por "std_logic". Este cambio se puede hacer sin necesidad de una función de conversión de tipos ya que "std_logic" es un subtipo resuelto de "std_ulogic".

Estos problemas desaparecen si el formato de destino, en lugar de VHDL, es XNF y además usamos el traductor X2V para generar un archivo VHDL a partir del netlist que genera el compilador. También se puede usar el corrector RENAME2 para que los nombres de los puertos de entrada y salida se mantengan, ya que al compilar a XNF en ocasiones estos nombres cambian bastante. Por más detalles de cómo usar estos programas ver Apéndice B.

### 4.1.2 Transmogrifier C

Los archivos XNF generados presentaban un problema de notación en el símbolo AND. La numeración de las entradas debe ser (según [4] ) I0, I1, I2, I3 sin embargo el TMCC genera símbolos con la siguiente notación I, I1, I2, I3. Esto se solucionó primero mediante un corrector escrito en lenguaje AWK y luego mediante correcciones a las fuentes del TMCC.

La compilación para las familias de Altera se realizó mediante el traductor de XNF a VHDL que mencionamos anteriormente. Sin embargo al traducirlos aparecen varios errores debido a que la especificación del XNF permite nombres de señales que los sintetizadores de VHDL que usamos no aceptan (el traductor X2V mantiene los nombres originales del archivo). Los nombres de señales:

- No pueden contener un guión ("-"). Esto ocurre en los puertos, el TMCC separa el nº de bit del nombre de la señal con ese carácter. También aparece en señales que comienzan con "FFin-".
- No pueden comenzar con un dígito.
- No pueden contener dobles "underscore" ("__").

## 4.2 Ejemplo de implementación de un diseño en la placa UP1

En este punto mostraremos el proceso completo de compilación e implementación de un diseño en la placa UP1 (ver Apéndice E).

La aplicación elegida para esto es uno de los ejemplos incluido en la distribución del TMCC es un contador cuya salida es desplegada en un display de 7 segmentos que cuenta de 0 a 99. Además permite regular el tiempo entre incrementos sucesivos (o sea la velocidad de la cuenta) mediante una entrada (que llamaremos "switches"). Esta entrada la conectaremos a los switches FLEX_SW1 de la placa. Otra entrada generada automáticamente al traducir de XNF a VHDL es una señal de reset llamada "globalReset" que lleva a 0 todos los flip-flop del circuito, esta señal la conectaremos al push-button FLEX_PB1. En el capítulo 5 comentaremos con más detalle la implementación de este ejemplo.

El ejemplo está pensado para trabajar con un oscilador de 15Hz que tienen los chips de la familia XC4000 por lo que hay que adaptarlo para poder usarlo con el oscilador de 25MHz de la placa UP1. Para ello modificamos la función delay( ) agregando tres contadores en cascada por lo tanto ahora cada incremento en "switches" incrementa el período en 170ms cuando antes lo hacía en 40 ns (1 período de reloj). Usamos tres contadores y no uno solo para disminuir el tamaño de los sumadores, pudiendo así cumplir con los requisitos de velocidad (25MHz). Las fuentes de este ejemplo para ambos compiladores están en el Apéndice F pág. 225.

### 4.2.1 Compilación con TMCC

Este compilador corre en ambiente UNIX, en éste la secuencia de comandos para generar el archivo VHDL a partir del archivo C es la siguiente:

```
tmcc –S countup.c myclock.xnf
awk –f nombres countup.xnf > count.xnf
mv count.xnf countup.xnf
perl -- x2v_v6.pl countup.xnf Alt
```

**Figura 4-3** Secuencia de comandos para generar el archivo VHDL del contador.

En caso de que se corrijan las fuentes del Transmogrifier C como se indica en el Apéndice B no serían necesarios los dos pasos intermedios. El archivo myclock.xnf provee la señal de reloj externa al circuito, este puede ser de la siguiente forma:

```
SYM, CLK, BUFG
PIN, O, O, HCLK
PIN, I, I, INPUT_CLK
END
EXT, INPUT_CLK, I
EOF
```

**Figura 4-4** Contenido del archivo myclock.xnf

### 4.2.2 Compilación con HCC

El HCC corre en ambiente DOS, en éste la secuencia de comandos para generar el archivo VHDL a partir del archivo C es la siguiente:

```
hcc –ns –cpp countup.c
awk –f rename2 countup.xnf > tmp.xnf
del countup.xnf
ren tmp.xnf countup.xnf
perl -- x2v_v6.pl countup Alt
```

**Figura 4-5** Secuencia de comandos para generar el archivo VHDL del contador.

### 4.2.3 Síntesis

Una vez generado el archivo VHDL los pasos siguientes son comunes a los dos compiladores (HCC y TMCC). Primero hay que hacer un cambio en el archivo VHDL “a mano”, debido a que el traductor inserta una señal de reset activa en 1 y queremos trabajar esta señal con los push-buttons de la placa que son activos por cero. A continuación, en la Figura 4-6, mostramos los cambios en el código necesarios.

| **Antes** | **Después** |
|---|---|
| entity countup is<br>  port (<br>  INPUT_CLOCK:in std_logic;<br>  globalReset: in std_logic;<br>…<br>begin *-- comienza cuerpo de la arquitectura*<br>  VCC <= '1';<br>  GND <= '0';<br>… | entity countup is<br>  port (<br>  INPUT_CLOCK: in std_logic;<br>  Reset: in std_logic;<br>…<br>  signal globalReset: std_logic;<br>begin *-- comienza cuerpo de la arquitectura*<br>  globalReset <= not Reset;<br>  VCC <= '1';<br>  GND <= '0';<br>… |

**Figura 4-6** Correcciones al archivo VHDL para usar los push-buttons como señal de Reset.

Luego de estos cambios hay que compilar el archivo para el chip EPF10K20RC240-4 asignando pines de la siguiente forma:

**Tabla 4-1** Asignación de pines para el contador.

| **Señal** | **Pin** |
|---|---|
| Hclk | 91 |
| switches(0) | 41 |
| switches(1) | 40 |
| switches(2) | 39 |
| switches(3) | 38 |
| switches(4) | 36 |
| switches(5) | 35 |
| switches(6) | 34 |
| switches(7) | 33 |
| right_digit(0) | 24 |
| right_digit(1) | 23 |
| right_digit(2) | 21 |

| **Señal** | **Pin** |
|---|---|
| right_digit(3) | 20 |
| right_digit(4) | 19 |
| right_digit(5) | 18 |
| right_digit(6) | 17 |
| left_digit(0) | 13 |
| left_digit(1) | 12 |
| left_digit(2) | 11 |
| left_digit(3) | 9 |
| left_digit(4) | 8 |
| left_digit(5) | 7 |
| left_digit(6) | 6 |
| Reset | 28 |

Luego de compilado el diseño en M+PII y con la asignación de pines mencionada se puede proceder a programar el chip (ver manual incluido con la placa UP1).

# 5 Ejemplos

El estudio esta dividido en dos partes. Primero realizamos ejemplos básicos, punto 5.1, donde nos encontramos con los primeros problemas al momento de la compilación y simulación. También analizamos la forma de migrar los códigos en C de un compilador a otro.

En el siguiente punto, 5.2, presentamos casos de complejidad superior y el diseño se realiza a partir de una especificación de comportamiento. Para estos ejemplos realizamos comparaciones con implementaciones realizadas directamente en VHDL. La lectura de estos casos puede resultar un poco tediosa por eso a continuación comentaremos brevemente las principales características de cada uno de ellos.

Árbitro de bus: se realizaron varias implementaciones tanto en C como en VHDL experimentando distintas formas de codificación de un algoritmo y su efecto en la performance del circuito final.

Acceso a memorias y multiplicadores: se ven dos grandes ejemplos, uno que utiliza accesos a memoria externa y el otro implementa una memoria interna tipo FIFO. En este último ejemplo nos enfrentamos a especificaciones de tiempos de respuesta muy rápidos.

Multiplicador con Pipeline: se examina mediante un conjunto de 6 ejemplos incrementales, la aplicación de la técnica de pipeline a un multiplicador.

Operaciones con matrices: se realizan diferentes implementaciones variando el origen de los datos, el almacenamiento interno de los mismos, también reutilizando hardware. En todos ellos el objetivo es ver el efecto de los cambios en la velocidad y en los recursos utilizados del chip.

Todas las pruebas fueron sintetizadas utilizando el software Max+Plus II versión 10.1 de Altera y para las familias de Xilinx se utilizó Foundation versión 4.1i.

## 5.1 Ejemplos Básicos

A continuación analizaremos varios programas con el objetivo de familiarizarnos con las distintas características de los dos lenguajes, detectar errores en los compiladores y comparar la facilidad para implementar los circuitos deseados.
La forma de trabajo fue, elegir ejemplos que vienen con los compiladores, verificar su funcionamiento mediante simulaciones del hardware generado (utilizando Max+Plus II o Foundation), modificar los programas intentando mejorar y optimizar el algoritmo. Luego los tradujimos al lenguaje del otro compilador intentando conservar el mismo comportamiento y repetimos el proceso de generación y simulación.

Las fuentes completas de todos los programas analizados se encuentran en el Apéndice F a partir de la pág. 227.

### 5.1.1 Contador con retardo y salidas en 7 segmentos

Uno de los ejemplos que viene con la distribución del TMCC es un contador cuya salida es desplegada en un display de 7 segmentos que cuenta de 0 a 99, permitiendo además regular mediante una entrada (que llamaremos "switches") el tiempo entre incrementos sucesivos (o sea la velocidad de la cuenta). A continuación vemos el código del programa.

```
#pragma intbits 8  /* fija ancho de valor de retorno */
seven_seg(x)
#pragma intbits 4  // fija ancho de x
        int x; // declaro el parametro de la función
        {
#pragma intbits 8
/* fija el ancho de las variables declaradas a partir de aquí
(result, n, y, leftdigit, rightdigit) */
        int result;

        x = x & 0xf; result = 0;
        if(x == 0x0) result = 0xfc;
        if(x == 0x1) result = 0x60;
        … // los casos de x=2 a x=0xE se omiten
        if(x == 0xf) result = 0x8e;
        return(~result);
        }

delay(n)
        int n;
        {
        while(n != 0)
                n = n - 1;
        }

twodigit(y)
            int y;
            {
            int tens;
            int leftdigit, rightdigit;
            /* declaro puertos de salida */
            outputport(leftdigit);
            outputport(rightdigit);

            tens = 0;
            while(y >= 10) {
                        tens = tens + 1;
                        y = y - 10;
                        }
            leftdigit = seven_seg(tens);
            rightdigit = seven_seg(y);
            }

// comienza programa principal
main(){
#pragma intbits 8
        int count, switches;
        // decl. switches como puerto de entrada
        inputport(switches);

            count = 0;
            while(1) {
                        twodigit(count);
                        count = count + 1;
                        if(count >= 100)
                                    count = 0;
                        delay(switches);
                        }
            }
```

**Figura 5-1** Código del contador en 7 segmentos para Transmogrifier C.

Como se aprecia en el código el programa consta de un loop principal y tres funciones, en el siguiente cuadro describimos su funcionamiento.

| | |
|---|---|
| main() | Consta de un loop infinito en el cual se realizan tres cosas: **1)** se llama a la función twodigit pasándole como parámetro el valor de un contador **2)** se incrementa dicho contador (va de 0 a 99) **3)** se llama a la función delay pasándole como parámetro el valor de la entrada switches. |
| delay(n) | Decrementa "n" hasta llegar a 0, donde cada cuenta es un período de reloj. No devuelve ningún valor. |
| twodigit(y) | El parámetro "y" es un valor entre 0 y 99. La función primero separa este valor en decenas y unidades y luego con c/u de estos valores llama a la función seven_seg y le asigna el valor de retorno de dicha función a las salidas que van a los displays. Esta función no retorna ningún valor. |
| seven_seg(x) | Esta función toma un entero entre 0 y 9 y devuelve el valor en 7 segmentos correspondiente. |

Al traducir este programa a Handel C hubo que hacer unas pocas modificaciones y fueron sencillas. La primera se debió a que el HCC no soporta funciones y tampoco acepta el pasaje de parámetros a los procedimientos. La solución fue utilizar variables globales para pasarle los parámetros y recibir los resultados del procedimiento.

La segunda modificación fue porque en Handel C el valor de las salidas se mantiene solo por un período de reloj, mientras que en Transmogrifier C las salidas se mantienen mientras no haya una nueva escritura. En otros diseños esto quizás no sea un problema, pero en este ejemplo las salidas alimentan displays, que si solo se mantienen por un período en el display se ve el valor por un instante y luego vuelve a su estado natural (todos los LED encendidos o todos apagados). En estos casos lo que hacemos es escribir en los puertos de salida en paralelo con el resto del programa. Para ello modificamos el programa agregándole otro loop infinito en paralelo con el loop principal, dentro del nuevo loop se actualizan los valores de las salidas (en paralelo) con los de una variable (una por salida). Así mientras no se cambien las variables se está permanentemente mostrando los mismos valores en las salidas. Esta técnica es más eficiente y previene de glitches frente a escribir en los puertos en cada línea del programa.

El código con estas modificaciones se puede apreciar en la siguiente figura.

```
const spec tgt={     // especificación del target
            fpga_type="Xilinx4000",
            clock_pad="P160"};

void main(target = tgt,  // declaración del target
            /* declaro puerto de entrada de 8 bits */
            port (in) switches:8,
            /* declaro puertos  de salida de 8 bits */
            port (out) leftdigit:8,
            port (out) rightdigit:8)
{
// se declaran variables especificando el ancho
int count, n, y, result, leftd, rightd :8;
int x, tens:4;

/* espera en x un n°  de 0 a f y devuelve el mismo n° en 7
seg. (8 bits) */
void seven_seg(){
            if(x == 0x0) result =~ 0xfc;
            … // igual a la versión para Tmcc
}

void retardo(){ // espera n periodos de reloj
            while(n != 0)   n = n - 1;
}

//toma de “count” el numero a mostrar
void twodigit(){
      tens, y = 0, count; // asignación paralela
      while(y >= 10)
            { /* este loop guarda en  tens el digito
             alto  y en "y" el digito bajo */
             tens, y = tens+1, y-10; //asig. paralela
            };
      x = tens;  /* asigno a x el digito alto antes de llamar a
                  seven_seg */
            Seven_seg();
            x, leftd = y.(0..3), result;  /* asig.  paralela, y.(0..3)
            asigna  los bits bajos de y a  x antes de llamar a
            seven_seg, a leftd se le  asigna result (lo que
            devuelve seven_seg()) */
            seven_seg();
            rightd= result; /* trabajo con  rightd y leftd donde
            antes trabajaba con los puertos lefdigit y
            rightdigit */
            }

      // Comienza el programa principal
      count = 0;
      par{
            while(1){   /* este loop lo agregamos para
                        actualizar las salidas */
                  par{ /* escritura en puerto leftdigit */
                     leftdigit ! leftd;
                     /* escritura  en puerto rightdigit */
                     rightdigit ! rightd;
                      }
                  };
             while(1) {
                        twodigit();
                        par{
                           count = count + 1;
                           // lectura en n  del puerto switches
                           switches ? n;
                            }
                        if(count >= 100)  count = 0;
                        retardo();
                        };
            }
}
```

**Figura 5-2** Código del contador en 7 segmentos para Handel C.

Luego observando el procedimiento (función) "seven_seg(x)", vimos que podía ser conveniente cambiar todos estos "if (x== ?? ){result=??};" por una instrucción case(x){..} en el caso de Handel C. En el caso del TMCC probamos anidar las sentencias IF agregándole cláusulas ELSE a cada uno, como vemos en la Figura 5-4.

```
case (x){
0x0: result =~ 0xfc;
0x1: result =~ 0x60;
....
0x8: result =~ 0xfe;
default: result =~ 0xf6;
};
```

**Figura 5-3** Modificación al procedimiento seven_seg() para Handel C

```
if ( x == 0x0 ) { result = 0xfc;}
else {  if( x == 0x1 ) { result = 0x60;}
.....
else {  if( x == 0x8 ) { result = 0xfe;}
else {  result = 0xf6;}
}; ….. };
```

**Figura 5-4** Modificación a la función seven_seg(x) para Transmogrifier C

En la tabla de resultados, Tabla 5-1, Hcc (v2-case) y Tmcc (v2) corresponden a los circuitos con estas últimas modificaciones.

Resultados
Chip EPF10K20RC240-4 (ALTERA)

**Tabla 5-1** Cuadro comparativo de recursos y rendimiento para el contador de 7 segmentos.

| | FFs | Celdas Lógicas | Frec. Max. [ MHz ] |
|---|---|---|---|
| **Hcc** | 88 | 212 | 13.42 |
| **Tmcc** | 61 | 129 | 26.31 |
| **Hcc (v2-case)** | 73 | 167 | 29.76 |
| **Tmcc (v2)** | 61 | 129 | 33.55 |

En este cuadro podemos observar como mejora el rendimiento al realizar las modificaciones a la función seven_seg(x). En el caso de Handel C la frecuencia máxima, que puede lograr el circuito, es de más del doble, además se observa un ahorro en FF y celdas lógicas. En el caso de Transmogrifier C no se modifica el número de FF y celdas lógicas sin embargo también hay un incremento en la frecuencia máxima aunque no tan significativo.

## 5.1.2 División

Uno de los ejemplos incluidos en la distribución del HCC calcula la división de dos números de 16 bits. Dichos números se ingresan en ciclos consecutivos al comienzo de un loop infinito. Luego se calcula el resultado usando el método de división con restauración (ver Apéndice A). El tiempo que demora este cálculo depende de que tan grande vaya a ser el resultado.

El código de este programa se puede observar en la siguiente figura.

```
/* Integer division by long-division method */
const spec tgt={ // especificación del target
      fpga_type="Xilinx4000", // para que  se  genere XNF
      clock_pad="P160"};  /* es necesario para que no use
                              osc. interno */

const dw = 16;

void main(  target=tgt,
      port (in)  STDIN  : dw, /* declaro puerto de entrada de
                                 ancho dw (16) */
      port (out) STDOUT : dw) // declaro  puerto de salida
{
      int a, b, c : dw;  // declaración de variables
      int bits : 5;
```

```
    while (1) {
        STDIN ? a;  // cargo en a el valor de  stdin
        STDIN ? b;  // en el siguiente ciclo se carga en b
        c, bits = 0, 1;  // asignación paralela
        while (b .<=. a)  /* desplazo b hasta  que sea mayor
        que a. Guardo en bits el nº de rotaciones
        Problema: a debe ser menor que 0x8000 si no, no
        sale del loop */
                b, bits = b << 1, bits + 1;
        do par {
            if (a .>=. b)
                a, c = a - b, (c << 1) ^ 1;  // ^ es = xor
            else
                c = c << 1;
            b, bits = b >> 1, bits - 1;
        } while (bits != 0);
        STDOUT ! c; // se escribe c en el  puerto
    };
}
```

**Figura 5-5** Código del divisor de 16 bits para Handel C.

Este programa no funciona cuando se quiere dividir un número mayor que 0x7fff. Esto se debe a que si *"a"* (dividendo) es mayor o igual que 0x8000 entonces para ciertos valores de *"b"* la condición del segundo WHILE siempre es verdadera. Por ejemplo si a = 0x8000 y b = 0x8000 (o b = 0x4000, 0x2000, 0x1000, etc) la primer vez que se evalúa la condición *"a" y "b"* son iguales por lo que ésta es verdadera, luego *"b"* se rota y pasa a valer 0, que es menor que *"a"* por lo que la condición nuevamente es verdadera. Esto se repite una y otra vez y nunca se sale del loop.

Esto lo solucionamos extendiendo la variable *"b"* en un bit, esto permite que *"b"* siempre pueda llegar a ser mayor que *"a"* al rotarla. La declaración de las variables quedaría así:

```
int a, c : dw;  // declaración de variables
int b: dw+1;
int bits : 5;
```

La traducción a Transmogrifier C fue muy sencilla teniendo que realizar solo dos pequeñas modificaciones al cuerpo del programa.

Para que las dos lecturas de las entradas se realicen en ciclos diferentes (consecutivos) hay que insertar una instrucción "*while( 0 ){ };*" entre ellas. Al insertar este WHILE lo que hacemos es generar un nuevo estado, por lo tanto las instrucciones que están antes y las que están después van a usar variables de estado diferentes que se van a activar en ciclos de reloj consecutivos.

La segunda modificación es debida a que el programa usa un operador de comparación sin signo, esto el TMCC no lo soporta. Esto lo resolvimos ampliando un bit más las variables "a" y "b" (esta ya la habíamos ensanchado).

En la siguiente figura se aprecia el código del programa.

```
// fijo ancho de las próximas variables
#pragma intbits 16
main(){
        int c, stdin, stdout;
#pragma intbits 17
        int a;
#pragma intbits 18
        int b;
#pragma intbits 5
        int bits;
        // declaro puertos de I/O
        inputport(stdin); outputport(stdout);

        while (1) {
                // cargo en a la entrada
                a = stdin & 0x0ffff;
                while( 0 ){};
                // en el siguiente ciclo cargo en b
                b = stdin&0x0ffff;
                c = 0;
                bits =  1;
/* desplazo b hasta que sea mayor que a,  guardo en bits
el nº de desplazamientos */
                while( b <= a ){
                        b = b<<1;
                        bits = bits+1;
                };
                while( bits!=0 ){
                        if (a >= b){
                                a = a-b;
                                c = (c<<1)^1;
                        }
                        else
                                c = c<<1;
                        b = b>>1;
                        bits = bits-1;
                };
                // escribo en el puerto
                stdout = c;
        };
}
```

**Figura 5-6** Código del divisor de 16 bits para Transmogrifier C.

Resultados

Chip EPF10K20RC240-4 (ALTERA)

**Tabla 5-2** Cuadro comparativo de resultados y rendimiento para la división

| | FFs | Celdas Lógicas | Frec. Max. [ MHz ] |
|---|---|---|---|
| **Hcc** | 59 | 199 | 13.44 |
| **Tmcc** | 60 | 277 | 19.60 |

La diferencia que se observa en celdas lógicas y frecuencia máxima se explica por que el Transmogrifier C usa por defecto sumadores con carry select mientras que Handel C usa sumadores ripple carry.

### 5.1.3 Raíz cuadrada

Otro de los ejemplos del HCC hace el cálculo de la raíz cuadrada. El programa es un loop infinito que pide un valor de entrada y 6 ciclos de reloj después devuelve la raíz de ese valor.

A este ejemplo los únicos cambios que le hicimos fueron: agregarle un target (y su especificación) y cambiarle el tipo de las entradas y salidas ( de "chan" a "port" ). Este último cambio es porque las salidas de tipo chan a veces se usan como entradas a alguna compuerta del circuito y al encontrar esto las herramientas de síntesis de VHDL reportan un error. Los canales están pensados para comunicar procesos y no para usarlo en los puertos de I/O del circuito.
El código se puede apreciar en la siguiente figura.

```
/* especificación del target */
const spec tgt={
        fpga_type="Xilinx4000",
        clock_pad="P°160"};

/* declaración de constantes */
const half_wd = 4 : 3;
const wd = half_wd << 1;
const sq = (1 << (wd - 2)) : wd;

void main( target=tgt,  // declaro target
        port (in)  STDIN: wd, // declaro puertos
        port (out) STDOUT : wd)
{
 /* declaraciones de variables, el compilador infiere los
  anchos, por eso no se declaran */
 int a, p, q, r;
 int i;
```

```
 while (1) {
    par { STDIN ? a;  // cargo puerto de entrada en a
          p, q, r = 0, 0, sq;  /* asignación paralela, se
                                  realizan  las 3 en el mismo ciclo */
          i = half_wd; }

   do {
      if (p+q+r .<=. a)  /* comparación sin signo  */
          i, r, q, p=i-1, r>>2, (q>>1)+r, p+q+r;
      else
          i, r, q    = i - 1, r >> 2, (q >> 1);
   } while (i != 0);
   STDOUT ! q;  // doy como salida q
 }
}
```

**Figura 5-7** Código de raíz cuadrada de 8 bits para Handel C.

Al traducirlo para Transmogrifier C los cambios que hubo que hacer (además de las declaraciones) fueron: eliminar las declaraciones de constantes, sustituyendo donde aparecían por su valor y cambiar la instrucción "*do { .... } while (i != 0);*" por "*while (i != 0){ .... };*" que como "i" se inicializa en 8 justo antes de dicho loop es lo mismo usar uno que otro . Este programa demora 6 ciclos de reloj en devolver la salida. Las fuentes de éste programa se pueden ver en la siguiente figura.

```
#pragma intbits 8  /* fijo anchos de stdin y  stdout*/
main(){
        int stdin,stdout;
#pragma intbits 9  /* fijo anchos de a,p,q y r */
        int a, p, q, r;
#pragma intbits 4  /* fijo ancho de i */
        int i;
        /* declaro puertos */
        inputport (stdin); outputport(stdout);

        while (1) {
                a = stdin & 0x0ff;  /* en los 8 bits bajos
                      de a cargo stdin y en el alto 0 */
                p = 0;
                q = 0;
                r = 0x80;
                i = 4;
```

```
                while(i!=0){
                        if (p+q+r <= a){
                                i = i-1;
                                p = p+q+r;
                                q = (q>>1)+r;
                                r = r>>2;
                        }
                        else{
                                i = i-1;
                                r = r>>2;
                                q= q>>1;
                        };
                };
                /* escribo en el puerto de salida */
                stdout=q;
        }
}
```

**Figura 5-8** Código de raíz cuadrada de 8 bits para Transmogrifier C.

Después buscando información sobre el TMCC en Internet encontramos dos algoritmos (algoritmos 2 y 3) diferentes escritos en Transmogrifier C para calcular la raíz cuadrada.

Estos programas eran diferentes al que vimos antes, por lo que los modificamos (sin tocar la parte en que calcula la raíz) para poder comparar los algoritmos, los códigos en sus versiones finales se pueden apreciar en el Apéndice F, pág. 230 y pág. 231.

Originalmente la parte que calcula la raíz estaba en una función que se llamaba desde un loop (este estaba dentro de otro loop infinito) y se le pasaba como parámetro el valor de un contador. Modificamos este comportamiento para que tome el valor, al que se calculará la raíz, de una entrada y no de un contador. Además eliminamos la función pasando las instrucciones que calculan la raíz al loop principal.

El código del primer programa (algoritmo 2) se puede apreciar en la siguiente figura.

```
main(){
#pragma intbits 4 /* fijo ancho de bit1*/
        int Bit1;
#pragma intbits 24 /* fijo ancho de N_reg,  R_reg, tmp */
        int N_reg, R_reg, tmp;
#pragma intbits 8 /* fijo ancho de Rt, stdin, stdout */
        int Rt, stdin, stdout;
#pragma intbits 9 /* fijo ancho de N */
        int N;
        /*declaro puertos de I/O */
        inputport(stdin); outputport(stdout);

        while(1){
          // cargo stdin en 8 bits bajos de N y 0 en el alto
              N=stdin & 0x0ff;
              N_reg=N;
              R_reg=0;
              Rt=0;
              Bit1=8;
              while (Bit1--){
                    R_reg=(Rt<<16)|0x4000;
                    while(0){}; /* while puesto para
                          generar un nuevo estado */
                    tmp=(N_reg - R_reg);
                    while(0){}; /* Idem anterior */
                    if (tmp>=0){
                          N_reg=tmp<<2;
                          Rt=(Rt<<1)|1; // | es = OR
                    }
                    else {
                          N_reg=N_reg<<2;
                          Rt=(Rt<<1);
                    };
              };
              stdout=Rt; /* escribo resultado en el puerto */
        };
}
```

**Figura 5-9** Código de raíz cuadrada, algoritmo 2, de 8 bits para Transmogrifier C.

Con este programa no hubo problemas y la traducción a Handel C fue muy sencilla como se aprecia en la siguiente figura.

```
const spec tgt={ /* especificacion del target */
    fpga_type="Xilinx4000",
    clock_pad="P160"};

const cero=0:16; /* declaro un cero de 16 bits de ancho */

void main( target=tgt, // declaracion del target
    port (out) sq:8, // declaracion de puertos
    port (in)  va:8)
{
    /* declaracion de variables, con su respectivo  ancho */
    int N,tt,Rt,nn:8;
    int Bit1:4;
    int N_reg,R_reg,tmp:24;

    while(1){
        va ? N; /* cargo puerto de entrada en N*/
        Bit1,N_reg,R_reg,Rt=8,0 @ N,0,0;
        /* @ indica concatenación */
        while (Bit1!=0){
            R_reg=((cero @ Rt)<<16)|0x4000;
            /* si no le especificaramos el ancho a la constante
            cero no se puede inferir su ancho */
            tmp, Bit1 =(N_reg - R_reg), Bit1-1;
            if (tmp>=0){
                par{ /* asignación paralela */
                    N_reg=tmp<<2;
                    Rt=(Rt<<1)|1;
                }
            }
            else {
                par{
                    N_reg=N_reg<<2;
                    Rt=(Rt<<1);
                }
            };
        };
        sq ! Rt; /* escritura en puerto */
    };
}
```

**Figura 5-10** Código de raíz cuadrada, algoritmo 2, de 8 bits para Handel C.

A continuación mostramos el código en Transmogrifier C del tercer algoritmo.

```
#pragma intbits 8 /* fijo ancho de variables*/
main(){
    int l2,  stdin, stdout;
    int N, nn, tt,u, u2,sal,v;

    /* declaro puertos */
    inputport(stdin); outputport(stdout);

    while(1){
        N=stdin; /* cargo stdin en N */
        if (N > 1){
            tt = N; l2 = 0;
            while (tt>>=2){ l2++;};
            u = 1 << l2;
            v = u;
            u2 = u << l2;
            while (l2) {
                l2--;
                v >>= 1;
                while(0){}; /* while puesto para generar un
                               nuevo estado y simplificar lógica */
                nn = (u + u + v) << l2;
                while(0){}; /* idem anterior */
                nn += u2;
                while(0){}; /* idem anterior */
                if (nn <= N) {
                    u += v; /* u = u + v */
                    u2 = nn;
                };
            };
            sal=u;
        } else { sal=N; };
        stdout=sal; /* escribo en el puerto */
    };
}
```

**Figura 5-11** Código de raíz cuadrada, algoritmo 3, de 8 bits para Transmogrifier C.

Este programa presentó problemas en su compilación ya que el circuito generado es muy grande y supera el tamaño máximo admitido por el compilador. Por esto hubo que reducir el nº de bits de entrada y salida (de 32/16 llegamos a 10/5). Realizando cambios en las fuentes del TMCC y corriendo en máquinas con mayor memoria que la del PC que estábamos usando (32MB RAM) permitirían obtener más bits, sin embargo no creímos que esto aportara mucho.

Luego vimos que este mismo algoritmo no andaba para valores del contador con el último bit en 1. Esto se debe a que el operador >>, del TMCC, al rotar mantiene el bit de signo. O sea, si el bit más significativo es uno, se insertan unos, en caso contrario se insertan ceros. Notamos que si N>=0x2FF al llegar a la instrucción "while(tt>>=2){l2++;}; " la condición nunca es falsa ya que se insertan unos en "tt".

Este problema lo solucionamos agregándole un bit al ancho de las variables y forzando el último bit de N a cero al cargarle la entrada. Es necesario forzar el último bit porque al copiar en una variable (N) otra con menos bits (stdin), los bits faltantes se rellenan con cero. Con este cambio la sección de declaración de variables queda de la siguiente forma:

```
#pragma intbits 8 /* fijo ancho de l2, stdin, stdout*/
main(){
    int l2,  stdin, stdout;
#pragma intbits 9 /* fijo ancho de N, nn,.. ,v */
    int N, nn, tt,u, u2,sal,v;
```

Para traducir este algoritmo (alg. 3) a Handel C hubo que hacerle varias modificaciones al código. El mayor problema fue que el HCC no acepta desplazamientos (<< , >>) que no sean de largo constante en tiempo de compilación. Como el número de posibilidades de desplazamientos es pequeño, ya que si desplazo más veces que el nº de bits el

resultado es 0, pudimos resolver esto con una instrucción CASE que evalúa la variable que da el nº de rotaciones y selecciona una expresión que tiene una rotación de largo una constante.

También hubo que hacer modificaciones con otros operadores ( ++, --, +=, >>= ) que Handel C no soporta. En Handel C no es necesario agregarle un bit a las variables ya que los operadores de desplazamiento insertan ceros siempre y además éste permite usar operadores de comparación sin signo ( .<., .>., .<=., .>=. ).

El código en Handel C de éste tercer algoritmo se observa en la siguiente figura.

```
const spec tgt={  /* especificacion del target */
    fpga_type="Xilinx4000", /* genera XNF */
    clock_pad="P160"}; /* necesario */

const sw=8;

void main(      target=tgt, /* declaracion target */
          port (out) sq:sw, /* declaro puertos */
          port (in)  va:sw)
{
    /* declaración de variables */
    int tt, N, nn: sw;
    int l2, tmp, u, v, u2, m:sw;

    while(1){
        va ? N; // cargo puerto de entrada en N
        if (2 .>. N){  //comparación sin signo de 2 con N
            u=N;
        }
        else {
            tmp, l2 = N>>2, 0; /*asignación  paralela, se
                                 realiza en un Tclk */
            while (tmp!=0){
                tmp, l2 =(tmp>>2), l2+1;
            };
```

```
            /*implementamos u=1<<l2, u2=u<<l2 teniendo en
            cuenta que el máx. valor de l2 en este punto es 3*/
            case(l2){
                0 : { u, u2 = 1, 1;  }
                1 : { u, u2 = 2, 4;  }
                2 : { u, u2 = 4, 16; }
                3 : { u, u2 = 8, 64; }
                default : { u, u2 = 0, 0; }
            };
            v=u;
            while (l2!=0) {
                v, l2 = v>>1, l2-1;
                case(l2){   /*implementa m=(u+u+v)<<l2, el
                            máx. valor de l2 en este pto. es 2 */
                    0 : {m = (u + u + v);  }
                    1 : {m = (u + u + v) << 1;  }
                    2 : {m = (u + u + v) << 2;  }
                    default : {m = 0;  }
                };
                m = m+u2; /* hcc no acepta m+=u2 */
                if (m .<=. N) { /*comparacion sin
                                  signo de m y N */
                    u, u2 =u+v, m;
                };
            };
        };
        sq ! u;  /* escritura en puerto */
    };
}
```

**Figura 5-12** Código de raíz cuadrada, algoritmo 3, de 8 bits para Handel C.

En cuanto a los tiempos de procesamiento, el segundo algoritmo demora un tiempo fijo (26 ciclos de reloj) en calcular la raíz cuadrada, en el tercero el tiempo de cálculo depende del valor de la entrada con un mínimo de 8 ciclos de reloj y un máximo (para 10 bits) de 24 ciclos de reloj.

A continuación veamos los resultados obtenidos en la compilación para Altera de los diferentes algoritmos presentados.

Resultados
Chip EPF10K20RC240-4 (ALTERA)

**Tabla 5-3** Cuadro comparativo de resultados y rendimiento para la raíz cuadrada (algoritmos 1, 2 y 3).

| | I/O bits | FF | Celdas Lógicas | Frec. Max. [ MHz ] | Tclk |
|---|---|---|---|---|---|
| **Hcc (alg. 1)** | 16/16 | 64 | 229 | 11.18 | 10 |
| **Tmcc (alg. 1)** | 16/16 | 88 | 318 | 9.36 | 10 |
| **Hcc (alg. 1)** | 8/8 | 35 | 105 | 17.69 | 6 |
| **Tmcc (alg. 1)** | 8/8 | 47 | 148 | 16.50 | 6 |
| **Hcc (alg. 2)** | 8/8 | 82 | 138 | 27.70 | 26 |
| **Tmcc (alg. 2)** | 8/8 | 97 | 189 | 18.58 | 26 |
| **Hcc (alg. 3)** | 8/8 | 82 | 240 | 20.70 | 8 a 24 |
| **Tmcc (alg. 3)** | 8/8 | 85 | 309 | 24.27 | 8 a 24 |

En el caso de Handel C observamos que cuanto menos demora el algoritmo en dar el resultado menor es la frecuencia máx. alcanzada. Este comportamiento no se conserva en el Transmogrifier C, se aprecia que el algoritmo 3 logra una mayor frecuencia que el algoritmo 2.

En todos los casos el TMCC produjo circuitos más grandes (tanto en FF como en celdas lógicas) que el HCC. En el alg. 1 se obtuvieron frecuencias similares, en el alg. 2 es muy superior el HCC y en alg. 3 es un poco superior el TMCC.

## 5.2 Ejemplos Avanzados

Aquí encontraremos ejemplos de aplicaciones que son típicamente realizadas en hardware (utilizando VHDL) las cuales programamos en C para nuestros compiladores, como también veremos aplicaciones o algoritmos que son típicamente desarrollados en software. Para ambos tipos de aplicaciones evaluamos la performance del circuito generado, en un caso además se realizaron pruebas en la placa de desarrollo UP1 (por más información sobre esta placa ver el Apéndice E)

A los diseños realizados con los compiladores agregamos además la comparación del mismo diseño pero desarrollado en VHDL y contrastamos sus desempeños.

Las fuentes de todos los programas analizados en esta sección se encuentran en el Apéndice F a partir de la pág. 231.

### 5.2.1 Árbitro de bus

Se implementó un árbitro de bus con memoria para los dos compiladores (TMCC y HCC) y también se realizaron pruebas en la placa UP1.

El circuito controla el acceso a un bus (o cualquier otro recurso que sea compartido) de dos dispositivos (que llamaremos A y B) aceptando peticiones por dos entradas y otorgando el bus por dos salidas (una entrada y una salida por dispositivo).

Inicialmente las dos salidas (Aclts, Bclts) están en 0, al llegar un pedido (mediante un 1 en Arts o Brts) el árbitro le otorga el bus al que lo solicito (pone un 1 en Aclts/Bclts). El árbitro mantiene en 1 Aclts (o Bclts) hasta que el dispositivo libera el bus mediante otro pulso a 1 en Arts/Brts. Para asegurar la detección, los pulsos de pedido del bus deben tener como mínimo un período del reloj, no hay suposiciones acerca del ancho máximo.

El árbitro, cuando el bus está ocupado, debe controlar si llegan pedidos del otro dispositivo y memorizar esos pedidos para otorgar el bus inmediatamente después que el dispositivo que lo estaba usando lo libere.

En la Figura 5-13 vemos un diagrama de tiempos mostrando el funcionamiento de pedido y reserva del bus.

En el diagrama destacamos los siguientes instantes:

(1) El dispositivo A pide el bus
(2) El dispositivo A recibe el bus
(3) El dispositivo B pide (reserva) el bus
(4) El dispositivo A devuelve el bus
(5) El dispositivo B recibe el bus

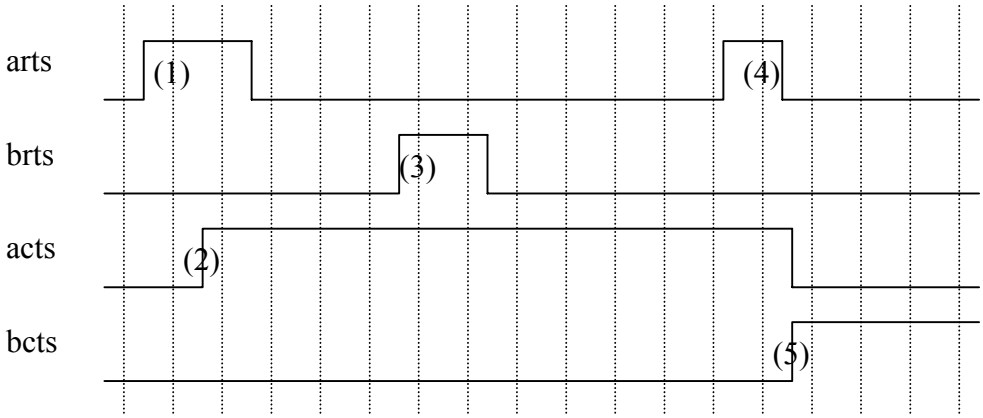



**Figura 5-13** Diagrama de tiempos del árbitro de bus.

El proceso de trabajo con este ejemplo consistió de las siguientes etapas:

1- Diseño del árbitro en C (para TMCC y HCC) refinando éste hasta lograr una salida aceptable (simulación sin glitches).

2- Diseño en VHDL del árbitro con una estructura similar a los programas anteriores (lo explicaremos más adelante) y con el mismo comportamiento.

3- Rediseño del árbitro en VHDL (partiendo de las especificaciones).

4- Intentar aproximar el diseño anterior en TMCC y HCC.

Etapa 1:

El diseño del árbitro fue sencillo para los dos compiladores, se cumplió con las especificaciones excepto que el árbitro debía otorgar el bus en el siguiente ciclo después que se pone a '1' la entrada de pedido. Con estos diseños el árbitro pone en '1' la salida en el segundo ciclo después del pedido.

Con el TMCC para lograr el comportamiento deseado tuvimos que usar entradas o salidas no registradas ( por ej. portflags(bcts,0x1)). En el primer caso lo que ocurre es que cuando el pedido llega cerca del flanco del reloj (<10ns) hay variables (estado) que por los retardos que tienen no llegan a cambiar (antes del flanco), mientras que otras con menos lógica si lo hacen. Con las salidas no registradas el problema es que hay glitches en las salidas.

Esto es igual para el HCC, se tienen los mismos problemas. Usar entradas no registradas es equivalente a trabajar directamente con las entradas (sin cargarlas en una variable). Usar salidas no registradas equivale a usar expresiones o poner a '1' las salidas al mismo tiempo que se cambia de estado.

En la Figura 5-14 se observa el código del árbitro escrito en Transmogrifier C y en la Figura 5-15 se encuentra el código escrito para Handel C.

```
main(){
#pragma intbits 1 //fijo ancho de variables
    int arts,brts,acts,bcts;
    int est_ant_A,est_ant_B;
    int a,b,c,d;
#pragma intbits 2
    int estado,reserva;
    /* declaro puertos */
    inputport(arts); inputport(brts);
    outputport(acts); outputport(bcts);
    /* declaro entradas como registradas */
    portflags(arts,0x3); portflags(brts,0x3);

    while(1){
        a=arts;b=brts; // leo puertos entrada
        c=a & (~est_ant_A); /* formo c y d  con el valor
                               anterior de est_ant_A*/
        d=b & (~est_ant_B);
        est_ant_A= a; // actualizo est_ant_A
        est_ant_B= b;
        if (estado==1){ /* canal ocupado por  A */
            if (c & (~d)){ // A devuelve el bus
                estado=reserva; /* próximo  estado */
                if (reserva==2) bcts=1;
                reserva=0;
                acts=0;
            }
            else if (d & (~c)){reserva=2;} /* B  reserva */
            else if (d & c){ // devolucion y pedido simult.
                estado=2;
                acts=0;
                bcts=1;
            }
            else {acts=1;bcts=0;};
        }
        else if (estado==2){    /* bus ocupado por B */
            if (d & (~c)){ // B devuelve el bus
                estado=reserva;
                if (reserva==1) acts=1;
                reserva=0;
                bcts=0;
            }
            else if (c & (~d)) {reserva=1;} /* A  reserva */
            else if (d & c) { /* devolucion y  pedido
                                 simultaneos */
                estado=1;
                acts=1;
                bcts=0;}
            else {acts=0;bcts=1;};
        }
        else if (estado==0){    // canal libre
            if (c){ /* A tiene prioridad */
                estado=1;acts=1;
                if(d) reserva=2;
            }
            else if (d & (~c)){/* si A esta libre puede
                                  reservar B */
                estado=2;bcts=1;
            }
            else {
                acts=0;bcts=0;
            };
        };
    };
}
```

**Figura 5-14** Código del árbitro de bus, versión 1, para Transmogrifier C.

```
const spec harp1 = { /* especificacion de target */
    fpga_type = "Xilinx4000",
    clock_pad = "P160"};

const LIBRE  = 0;
const OCUPAA = 1;
const OCUPAB = 2;
#define a (temp_a & (~est_ant_A))
#define b (temp_b & (~est_ant_B))

main(   target=harp1,  /* decl. de target */
    port(in) arts: 1, /* decl. de puertos */
    port (in) brts:1,  port (out) acts: 1,
    port (out) bcts: 1)
{
    /* declaracion de variables*/
    int estado,reserva:2;
    int temp_a,temp_b:1;
    int est_ant_A ,est_ant_B:1;

    while(1){
        par{
            arts ? temp_a; /* leo puertos I */
            brts ? temp_b;
            est_ant_A=temp_a;
            est_ant_B=temp_b;
            case (estado){
                OCUPAA: par{ /* bus ocupado por A, el
                 par{} hace que las dos escrituras a puertos
                y el if se ejecuten en el mismo ciclo */
                        acts ! 1;
                        bcts ! 0;
                        if (a & (~b)){ /* A devuelve  el bus */
                            estado,reserva = reserva,LIBRE;
                        }
                        else if (b & (~a)){ // B  reserva el bus
                            reserva=OCUPAB;
                        }
                        else if (a & b) { /* devolucion y pedido
                                             simultaneos */
                            estado=OCUPAB;
                        }
                        else    delay;
                    }
                    OCUPAB: par{ /* bus ocupado por B */
                        bcts ! 1; /* esc. a puertos */
                        acts ! 0;
                        if (b & (~a)){ /* B devuelve el bus */
                            estado,reserva = reserva,LIBRE;
                        }
                        else if (a & (~b)){ /* A reserva el bus */
                            reserva=OCUPAA;
                        }
                        else if (b & a){ /* devolución y pedido
                                            simultaneo */
                            estado=OCUPAA;
                        }
                        else delay;
                    }
                    default: if (a){ /* A tiene prioridad */
                            par{
                                estado=OCUPAA;
                                reserva=(b ? OCUPAB : 0);
                            /* asignación condicional, si b=1 =>
                             reserva = OCUPAB sino reserva = 0 */
                            }
                        }
                        else if (b){ /* si A no reserva lo puede
                                        hacer B*/
                            estado=OCUPAB;
                        }
                        else delay;
                    };
            }
        };
}
```

**Figura 5-15** Código del árbitro de bus, versión 1, para Handel C.

Etapa 2:

Luego de ajustado el comportamiento del árbitro se procedió a diseñarlo en VHDL con el mismo estilo que en los diseños en C, las fuentes de este programa se pueden observar en el Apéndice F pág. 232. Esto quiere decir que usamos las mismas variables que en las versiones anteriores y estas tienen el mismo comportamiento.

El programa consiste de un gran proceso cuyo cuerpo es muy similar a lo que estaba dentro del loop infinito en los programas anteriores (HCC, TMCC).

Etapa 3:

En esta etapa nos olvidamos de todo lo anterior y encaramos el problema desde VHDL intentando lograr una mejor solución, las fuentes están en el Apéndice F pág. 233.

El diseño final consta de 3 máquinas de estados. Las máquinas 1 y 2 generan una señal (pa y pb) a partir de las entradas (arts y brts) que entregan a la tercer máquina, esta genera las salidas (respuestas) en función de dichas señales.

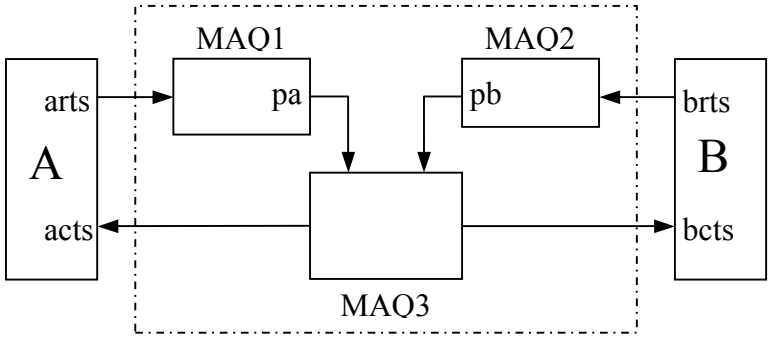



**Figura 5-16** Diagrama de bloques mostrando el árbitro de bus diseñado en VHDL.

**MAQ1 y MAQ2**

Estas máquinas controlan los pedidos de los dispositivos y generan una salida, en función de estos, hacia MAQ3. A continuación mostramos su diagrama de estados y un diagrama de tiempos que muestra su funcionamiento.

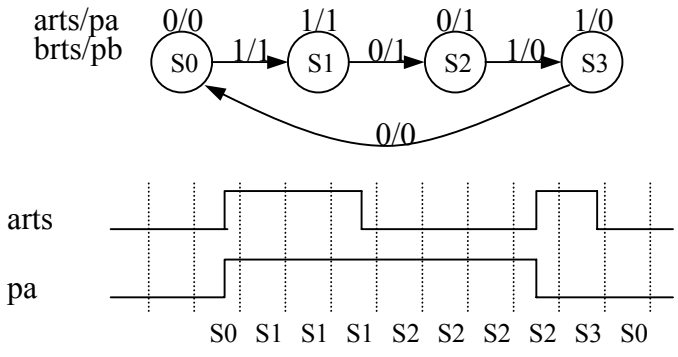



**Figura 5-17** Diagramas de estados y tiempos del bloque de entrada del árbitro de bus en VHDL

**MAQ3**

Esta máquina recibe las salidas de las anteriores (pa y pb) y a partir de ellas genera las entradas a los registros (acts_d, bcts_d) de las señales de respuesta (acts, bcts) a los dispositivos. A continuación mostramos el diagrama de estados de esta máquina y un diagrama de tiempos mostrando el caso en que "A" pide el bus (pa en 1) y luego "B" lo reserva (pb en 1).

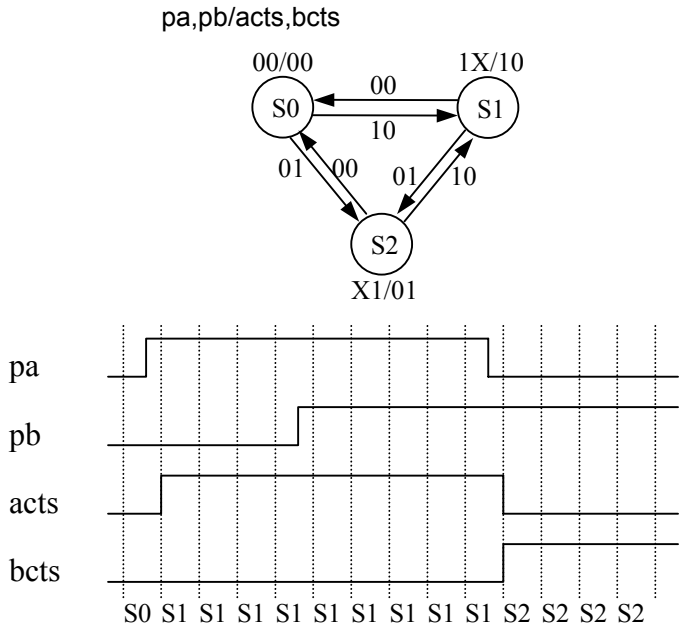



**Figura 5-18** Diagramas de estados y tiempos de MAQ3

Este diseño tiene el mismo problema que los diseños de la etapa 1, que nos llevaron a diseños que responden un ciclo de reloj después de la especificación inicial. En este caso el tiempo que tienen que respetar las señales de pedido del bus (antes del flanco del reloj) son menores (4ns frente a 10ns). Sin embargo si no respetan dicho tiempo este diseño funciona correctamente pero la respuesta se retrasa un período de reloj, en los anteriores el árbitro dejaba de funcionar.

Etapa 4:

En Handel C no podemos imitar exactamente el diseño en VHDL de la etapa 3 ya que no podemos hacer que el valor de una variable ("pa" o "pb") cambie al variar una entrada, éstas se actualizan en los flancos de reloj. Sin embargo con una pequeña modificación podemos lograr un comportamiento muy parecido al diseño de la etapa 3. En éste las máquinas de estado 1 y 2 son del tipo Mealy, es decir que su salida (pa o pb) depende del valor de la entrada. La máquina 3 en cambio es del tipo Moore ya que sus salidas (acts y bcts) solo dependen del estado.

Para el nuevo diseño en HandelC cambiamos el tipo a cada máquina, es decir, ahora las máquinas 1 y 2 serán del tipo Moore y la máquina 3 será del tipo Mealy. En el programa cada máquina de estado es un procedimiento distinto, el procedimiento principal es un loop infinito que ejecuta estos procedimientos en paralelo.

```
const spec tg = { /* espec. del target */
    fpga_type = "Xilinx4000",
    clock_pad = "P160"};
/* defino constantes para los estados */
const s0 = 0;
const s1 = 1;
const s2 = 2;
const s3 = 3;

void main(      target = tg, // decl. del target
          port (in) arts:1, // decl. de puertos
          port (in) brts:1,
          port (out) acts:1,
          port (out) bcts:1){
   /* declaración de variables globales*/
   int pa, pb:1;

   void maq1(){ // maq. 1, maneja arts y da salida pa
      int est1:2; /* variable local */
      case (est1){ /* en cada rama calculo nuevos
                      valores de est1 y pa en paralelo  */
        0 :  est1, pa = (arts ? s1 : s0), arts;
             /* _?_:_ asignacion condicional, si arts=1
              => est1=s1 sino est1=s0 */
        1 :  est1, pa = (arts ? s1 : s2), 1;
        2 :  est1, pa = (arts ? s3 : s2), ~arts;
        3 :  est1, pa = (arts ? s3 : s0), 0;
      };
   }

   void maq2(){// maq. 2, maneja brts y da salida pb
      int est2:2;
      case (est2){
        0:  est2, pb = (brts ? s1 : s0), brts;
        1:  est2, pb = (brts ? s1 : s2), 1;
        2:  est2, pb = (brts ? s3 : s2), ~brts;
        3:  est2, pb = (brts ? s3 : s0), 0;
      };
   }
```

```
   void maq3(){ /* maquina 3, da salidas  acts y bcts
                   en funcion de pa y pb */
      int est3:2;
      case (est3){
         0:  par{
               est3 = (pa ? s1 : (pb ? s2 : s0));
               /* asignación condicional andidada, si
                  pa=1  => est3=s1, si pa=0 y pb=1
                  => est3=s2  sino est3=s0 */
               acts ! pa;
               bcts ! (~pa & pb);
             }
         1:  par{
               est3 = (pa ? s1 : (~pa & pb ? s2 : s0));
               acts ! pa;
               bcts ! (~pa & pb);
             }
         2:  par{
               est3 = (pb ? s2 : (~pb & pa ? s1 : s0));
               acts ! (pa & ~pb);
               bcts ! pb;
             }
         3: delay;
      };
   }

   // Programa principal
   while(1){
      par{ // ejecuto en paralelo las 3 máq de estados
         maq1();
         maq2();
         maq3();
      }
   };
}
```

**Figura 5-19** Código del árbitro de bus, versión 2, para Handel C.

A continuación vemos un diagrama de tiempos mostrando el nuevo comportamiento del circuito.

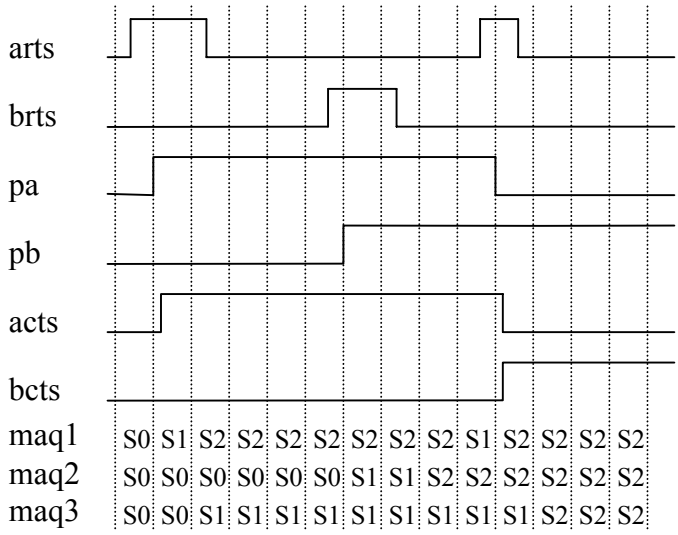



**Figura 5-20** Diagrama de tiempos del árbitro de bus, versión 2, para Handel C.

En Transmogrifier C se pueden realizar cualquiera de las dos opciones, o sea se puede hacer como el diseño de la etapa 3 o como el diseño anterior en Handel C. Pasar de una implementación a otra es tan sencillo como cambiar el orden de las sentencias IF que implementan las máquinas de estados y cambiar el tipo de las salidas.

Para un comportamiento igual al diseño en VHDL hay que poner primero las máquinas 1 y 2 y usar salidas registradas. Para un comportamiento igual al diseño en Handel C hay que poner primero la máquina 3 y usar salidas no registradas. La versión que implementamos es igual al diseño en Handel C, elegimos esta a los efectos de comparar los dos compiladores implementando el mismo circuito.

En la siguiente figura se puede observar su código.

```
main(){
/* declaracion de variables, aquí no es  necesario
especificar que son de ancho uno */
int arts, brts, acts, bcts, pa, pb;
#pragma intbits 2 /*ahora fijo ancho 2 */
int est1, est2, est3;

/* declaracion de puertos */
inputport(arts); inputport(brts);
outputport(acts); outputport(bcts);
/* declaro salidas como no registradas */
portflags(acts, 0x1); portflags(bcts, 0x1);

    while(1){
        if (est3 == 0){ // máquina 3
            /* el and con 1 es para forzar el segundo
               bit a cero */
            est3 = (pb & ~pa) << 1 | (pa & 1); /* si pb=1
                          y pa=0 => est3=2; si pa=1
                         =>  est3=1; sino est3=0 */
            acts = pa; /* esc. en puertos O*/
            bcts = pb & ~pa;
        }
        else if (est3 == 1){
            if (pa == 0) est3 = pb << 1; /* est3 va a 2 si
                                    pb=1 y a 0 si pb=0 */
            acts = pa; /* esc. en puertos O*/
            bcts = pb & ~pa; /* ~ representa
                                 complemento a 1*/
        }
        else if (est3 == 2){
            if (pb == 0) est3 = pa & 1;
            acts = pa & ~pb; /* esc. en O*/
            bcts = pb;
        }

        /* máquina 1 - de Moore las salidas  dependen
           solo del estado, aunque no lo  parezca */
        if (est1 == 0){/* si en un flanco de  reloj arts=1
                  entonces pa pasa a valer 1 al  mismo
                tiempo que est1 pasa a valer 1*/
            pa = arts;
            est1 = arts & 1; // fuerzo bit 2 a 0
        }
        else if (est1 == 1){/* me quedo en est1 dando
                          salida 1 hasta que baja arts */
            pa = 1;
            if (~arts)  est1 = 2;
            else  est1 = 1;
        }
        else if (est1 == 2){/* me quedo en est2 dando
                          salida 1 hasta que sube arts */
            pa = 1;
            if (arts) est1 = 3;
        }
        else { /* me quedo en est3 dando  salida 0
                    hasta que baje arts */
            pa = 0;
            if (arts)       est1 = 3;
            else            est1 = 0;
        };
        // máquina 2 analoga a la anterior
        if (est2 == 0){
            pb = brts;
            est2 = brts&1;
        }
        else if (est2 == 1){
            pb = 1;
            if (~brts)      est2 = 2;
        }
        else if (est2 == 2){
            pb = 1;
            if (brts)       est2 = 3;
        }
        else {
            pb = 0;
            if (~brts)      est2 = 0;
        };
    }
}
```

**Figura 5-21** Código del árbitro de bus, versión 2, para Transmogrifier C.

Resultados

**Tabla 5-4** Cuadro comparativo de resultados y rendimiento para árbitro de bus.

| **ALTERA - Chip EPF10K20RC240-4** | | | | |
|---|---|---|---|---|
| | **FF** | **Celdas Lógicas** | | **Frec. Max. [ MHz ]** |
| **Transmogrifier versión 1** | 12 | 50 | | 42.73 |
| **HandelC versión 1** | 12 | 34 | | 46.29 |
| **VHDL versión 1** | 10 | 36 | | 49.75 |
| **Transmogrifier versión 2** | 10 | 20 | | 64.51 |
| **HandelC versión 2** | 12 | 26 | | 62.11 |
| **VHDL versión 2** | 8 | 10 | | 99.00 |
| **XILINX - Chip XC4010XLPC84-3** | | | | |
| | **FF** | **3-LUT** | **4-LUT** | **Frec. Max. [ MHz ]** |
| **Transmogrifier versión 1** | 10 | 5 | 55 | 36.26 |
| **HandelC versión 1** | 12 | 3 | 29 | 48.61 |
| **VHDL versión 1** | 8 | 4 | 38 | 39.98 |
| **Transmogrifier versión 2** | 8 | 0 | 17 | 92.51 |
| **HandelC versión 2** | 12 | 0 | 14 | 57.29 |
| **VHDL versión 2** | 11 | 1 | 15 | 73.23 |

Al compilar con el software Max+Plus II los resultados fueron los esperados, los primeros diseños (HCC, TMCC) fueron los más lentos y costosos en hardware. El primer diseño en VHDL mejoro un poco la velocidad y el segundo aumentó considerablemente la velocidad con menos hardware. Los diseños en C que imitan a este último no alcanzan su velocidad sin embargo son superiores a los anteriores.

Sin embargo al compilar con Foundation los resultados son totalmente diferentes, no se mantiene la relación encontrada para Altera, de tamaños y frecuencia máxima, entre TMCC, HCC y VHDL.

La simulación de todos los diseños fue correcta. Cabe destacar que se simularon en M+PII a 100MHz los diseños en Handel C y Transmogrifier C en su versión 2. Lo único que ocurría es que se retrasaba la salida en 2 períodos de reloj, el árbitro seguía cumpliendo su función sin hacer nada mal.

A continuación, en la Figura 5-22, vemos un diagrama de tiempos mostrando el comportamiento de cada circuito frente a una petición simultanea de A y B. Luego agregamos una tabla con tiempos de respuesta de la señal acts en cada caso, los datos fueron tomados de las simulaciones con el M+PII.

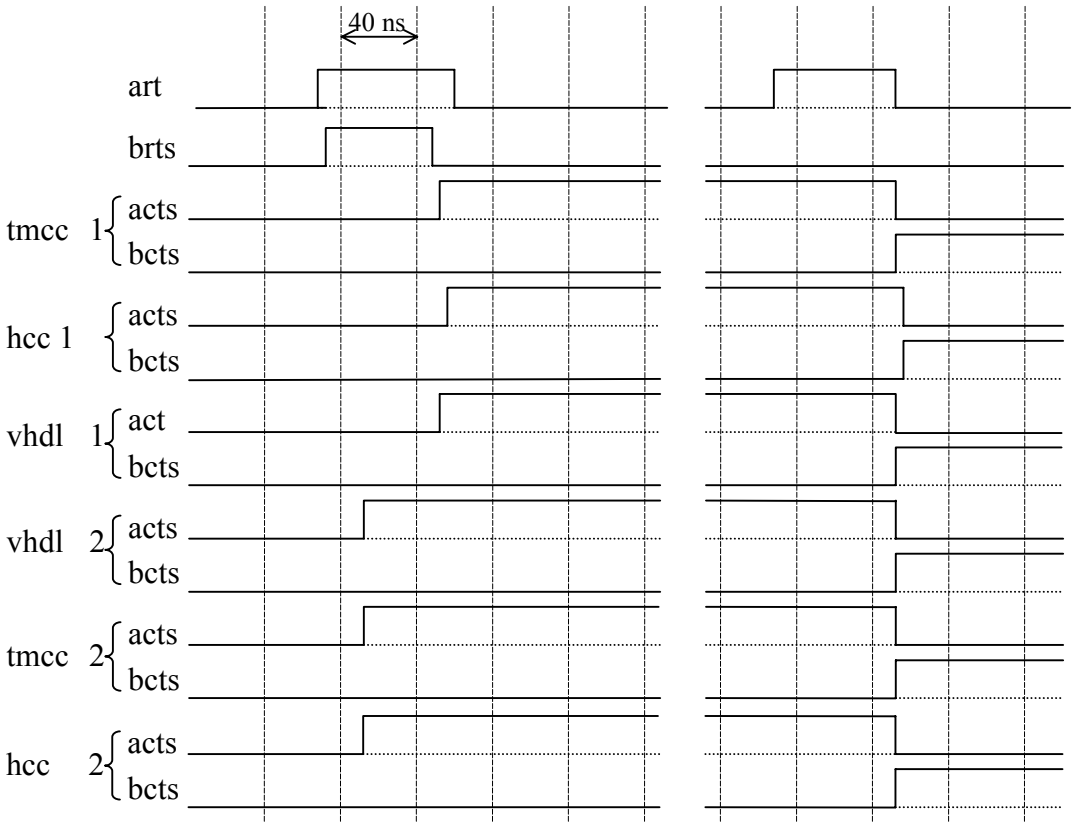



**Figura 5-22** Diagrama de tiempos del árbitro de bus para los diferentes diseños.

**Tabla 5-5** Cuadro comparativo de tiempos de respuesta del árbitro de bus para diferentes diseños.

| | tmcc v.1 | handelC | vhdl v.1 | vhdl v.2 | tmcc v.2 | hcc v.2 |
|---|---|---|---|---|---|---|
| **Subida de acts (ns)** | 13 | 18 | 13 | 13 | 20.9 | 18.6 |
| **Bajada de acts (ns)** | 13 | 18 | 13 | 13 | 20.9 | 18.6 |

NOTA: Retardos medidos desde el flanco del reloj anterior.

## Prueba en Hardware

Para trabajar con la versión 1 de ambos compiladores en la placa UP1, además de las salidas mencionadas se generaron otras dos para ver quién tiene el bus y quién lo está reservando en los displays de 7 segmentos de dicha placa.

Además tuvimos que agregar un eliminador de rebotes a cada entrada de petición del bus (conectadas a los push buttons), dicho eliminador se implementó en VHDL y consiste de dos contadores de 8 bits conectados en cascada. Cuando hay un flanco en una de las entradas, su eliminador comienza a contar y hasta que no llega al final (256*256*40ns = 2.62ms) se ignora cualquier transición en dicha entrada. Existen otras posibilidades más sencillas para eliminar los rebotes que utilizan los dos push buttons para hacer bascular la entrada entre 0 y 1, pero para nuestro diseño necesitábamos un botón para cada entrada.

## 5.2.2 Acceso a Memorias y Multiplicadores

**Acceso a memoria externa y operador multiplicación**
Ideamos un programa para probar accesos a memorias externas, este programa espera una señal de comienzo y luego ejecuta un loop 128 veces. Dentro del loop se leen dos datos (8 bits) consecutivos de una memoria externa de 256x8, luego esos datos se multiplican (sin signo) y el resultado (16 bits) se guarda en las mismas posiciones de memoria (el byte bajo en la dirección más baja).

Para diseñar y verificar para memorias reales, los tiempos de las señales de acceso a la memoria se compararon con los de las memorias presentes en la placa ARC-PCI (ver Apéndice E).

La implementación en Handel C fue muy sencilla debido a que este ya tiene previstos los accesos a memoria externa y además el operador multiplicación (con y sin signo). El código se puede ver en la siguiente figura.

```
const spec yyy = {  //especificación del target
    fpga_type= "Xilinx4000",
    clock_pad= "P13"};
// especificacion de la ram externa
const spec tipo_ram={
    addr = {"mem_Addr<0>", "mem_Addr<1>",
        "mem_Addr<2>","mem_Addr<3>","mem_Addr<4>",
        "mem_Addr<5>","mem_Addr<6>","mem_Addr<7>"},
    data = {"mem_Data<0>","mem_Data<1>",
        "mem_Data<2>","mem_Data<3>","mem_Data<4>",
        "mem_Data<5>","mem_Data<6>","mem_Data<7>"},
    ce   = "mem_ce",
    wb   = "mem_wb",
    en   = "mem_en"
};

main(   target = yyy, //declaracion de target
    port (in) comenzar:1, // decl. de puertos
    port (out) fin:1,
    eram mem[256] = tipo_ram:8) //decl. de la ram externa
{
    //declaración de variables
    int aux:1;
    int a,b,i,j:8;
    int c:16;

    while(1){
        comenzar ? aux; //leo entrada
        // espero a que comenzar sea igual a 1
        while (aux==0){comenzar ? aux;};
        do {     //leo 256 datos y escribo 128 de a 2 lugares
            par{
                a=mem[i]; /*leo posicion i de la ram */
                j=i+1;
            }
            b = mem[ j ];  //leo sig. posicion
            c = a .*. b;   //multiplico datos
            mem[ i ] = c.(0..7);   /*escribo byte bajo en
                                     posicion baja */
            par{
                mem[j]=c.(8..15); /*escribo byte alto en
                                    posicion alta */
                i=i+2;
            }
        } while (i != 0);
        fin ! 1; //aviso que termine
    };
}
```

**Figura 5-23** Código ejemplo de acceso a memoria externa con Handel C.

A continuación mostramos un diagrama de tiempos para la parte destacada del código.

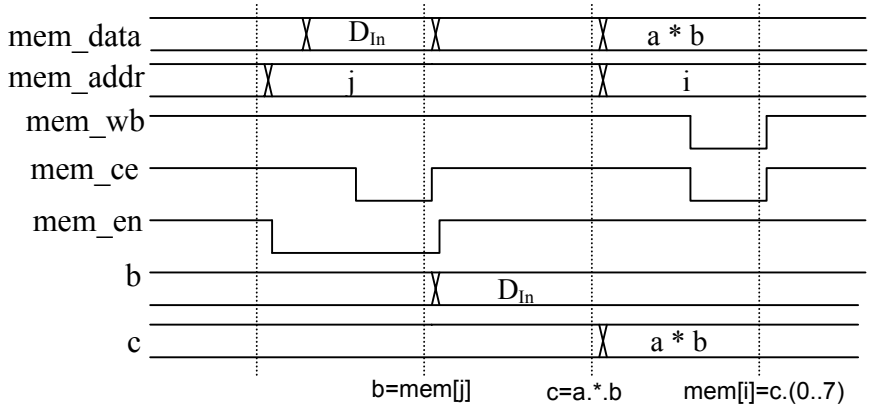



**Figura 5-24** Diagrama de tiempos para las señales de acceso a RAM externa con el Handel C.

El Transmogrifier C no tiene previstos los accesos a memoria y tampoco la multiplicación. Por esto hubo que generar las señales de control de la memoria "a mano" y además (para la versión 1) realizar una función que multiplique dos números de 8 bits. El código de la versión 1 se puede apreciar en la siguiente figura.

```
main(){
    // declaro variables de ancho 1
    int mem_ce, mem_wb, mem_en;
    int comenzar, fin;
#pragma intbits 8
    int a, b, i, j;
    int mem_addr, mem_data;
#pragma intbits 16
    int c;
    //declaro puertos entrada, salida y buses
    inputport(comenzar);
    bus_port(mem_data);
    outputport(fin);outputport(mem_ce);
    outputport(mem_en); outputport(mem_wb);
    outputport(mem_addr);

    mem_en=1;mem_ce=1;mem_wb=1;
    while(1){
        fin=0; i=0; j=0;
        while (comenzar==0){}; // espero pulso de comienzo
        mem_en=0; mem_ce=0; /*activo señales para
                                primer dato */
        mem_addr=i;
        while ((i!=0) | (j!=1)){
            mem_en=1; //desactivo señales de ram
            mem_wb=1; mem_ce=1;
            a=mem_data; //leo dato
            j =i+1;
            while(0){}; //espero a siguiente ciclo
            mem_en=0; /* activo señales de lectura */
            mem_addr=j; mem_ce=0;
            while(0){};
            b=mem_data; //leo dato
            mem_en=1; /* desactivo señales de lectura */
            mem_ce=1;
            /* multiplico a y b se genera un nuevo estado */
            c = multip8(a,b);
            mem_data=c; //escribo c
            mem_ce=0; /* activo señales de escritura */
            mem_addr=i; mem_wb=0;
            while(0){};
            mem_ce=1; mem_wb=1; /* finalizo escritura */
            while(0){};
            mem_ce=0; /* activo señales de escritura */
            mem_addr=j; mem_wb=0;
            mem_data=(c>>8); //escribo byte alto
            while(0){};
            i =i+2;
            mem_ce=1; mem_wb=1; /* finalizo escritura */
            while(0){};
            /* si no llegue al final genero señales para una
               nueva lectura */
            if (i!=0){mem_ce=0; mem_en=0; mem_addr=i;};
            bus_idle(mem_data); /* apronto bus para leer */
        };
        fin=1; // aviso que termine
    };
}
```

**Figura 5-25** Código ejemplo de acceso a memoria externa con Transmogrifier C.

Para esta versión implementamos una función que toma dos enteros de 8 bits como parámetros y devuelve su multiplicación en 16 bits. Dicha función implementa el método "right shift algorithm" en forma combinatoria (ver Apéndice A).

El código de dicha función se puede observar en la siguiente figura.

```
#pragma intbits 16  /* fijo ancho de bits que devuelve la
                       funcion */
int
multip8(a,b)
#pragma intbits 8 // fijo ancho de los parametros
    int a,b;
{
#pragma intbits 16
    //declaracion de variables locales
    int tmp0,tmp1,tmp2,tmp3;
    int tmp4,tmp5,tmp6,tmp7;
    int aux,sum0,sum1,sum2,sum3,mul0,mul1;
    int sal1,sal2,sal3, c;

    aux=a & 0x00ff; /*en aux cargo 8 ceros y a en los bits
                      bajos */
    //desplazamientos
    if ((b & 1 )==1 ){tmp0= aux   ;}
    else {tmp0=0;};
    if ((b & 2 )==2 ){tmp1=(aux<<1);}
    else {tmp1=0;};
    if ((b & 4 )==4 ){tmp2=(aux<<2);}
    else {tmp2=0;};
    if ((b & 8 )==8 ){tmp3=(aux<<3);}
    else {tmp3=0;};
    if ((b & 16 )==16 ){tmp4=(aux<<4);}
    else {tmp4=0;};
    if ((b & 32 )==32 ){tmp5=(aux<<5);}
    else {tmp5=0;};
    if ((b & 64 )==64 ){tmp6=(aux<<6);}
    else {tmp6=0;};
    if ((b & 128)==128){tmp7=(aux<<7);}
    else {tmp7=0;};

    //sumas
    sum0=tmp0+tmp1;
    sum1=tmp2+tmp3;
    sum2=tmp4+tmp5;
    sum3=tmp6+tmp7;
    //sumas 2 nivel
    mul0=sum0+sum1;
    mul1=sum2+sum3;
    //ultimo nivel de suma
    c=mul0+mul1;
    return(c);
}
```

**Figura 5-26** Código multiplicador de 16 bits para Transmogrifier C.

Al realizar dicha función encontramos un problema similar al visto en el ejemplo 5.1.3 (cálculo de raíz cuadrada) con el bit de signo de los números. La solución adoptada en este caso fue asignar el multiplicando (“a”) a otra variable de 16 bits forzando los 8 bits más significativos a cero (aux = a & 0xff), todas las variables que almacenan los desplazamientos y las sumas deben ser declaradas de 16 bits.

A continuación vemos un diagrama de tiempos de la parte en que se hace la segunda lectura, la multiplicación y la primera escritura.

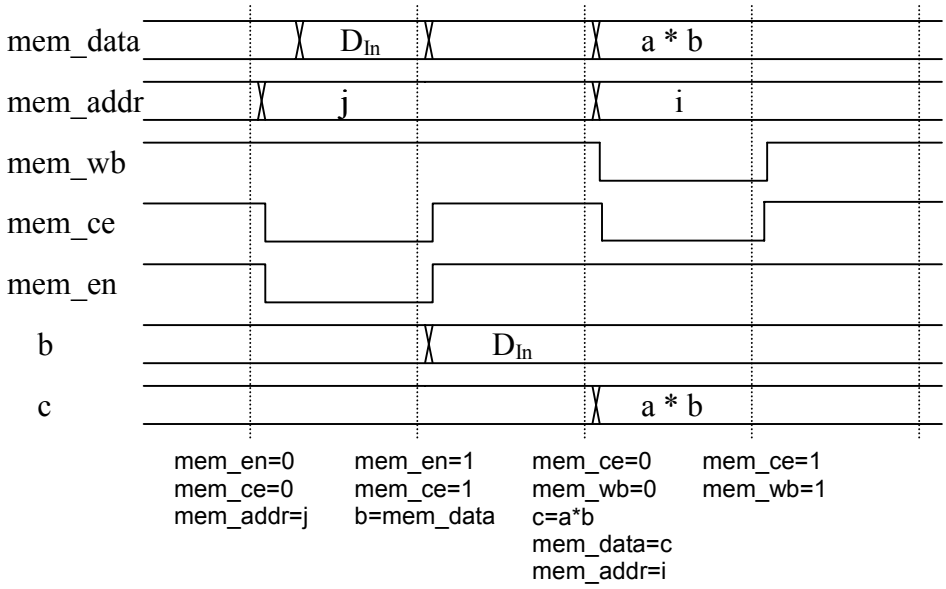



**Figura 5-27** Diagrama de tiempos para las señales de acceso a RAM externa con el Transmogrifier C.

El hecho de tener que desactivar (poner en 1) las señales de control de la memoria entre lecturas y escrituras lleva a que el loop demore más ciclos de reloj que en el caso de Handel C. Este lo que hace es activar las señales "ce" y "wb" solo por medio período de reloj por lo que es posible hacer dos lecturas o escrituras en ciclos de reloj consecutivos. En el caso de dos lecturas consecutivas se deja activa la señal "en" entre los dos ciclos de reloj.

En el Transmogrifier no se puede activar las señales por solo medio período de reloj, por lo que entre dos lecturas o escrituras tenemos que insertar un ciclo para desactivar alguna de las señales. Sí es posible hacer una lectura y una escritura o viceversa ya que, dejando activa “ce”, cuando desactivo “en” activo “wb”.

Luego modificamos el traductor de XNF a VHDL para que inserte multiplicadores. En el caso de Altera se inserta la mega función "lpm_mult" y en el caso de Xilinx se inserta un operador producto “*“. En este ejemplo insertamos un multiplicador (m1) definiendo dos salidas (mul_m1_mn, mul_m1_mr) y una entrada (mul_m1_res). Cuando queremos multiplicar las variables "a" y "b" lo que hacemos es asignar esos valores a las salidas mencionadas y el resultado es leído desde la entrada a otra variable ("c"). El traductor realiza las conexiones y declaraciones necesarias automáticamente. Por más detalles de cómo usar el traductor para insertar multiplicadores ver Apéndice B.

A continuación mostramos los cambios al código de la Figura 5-25.

```
main(){
   int mem_ce, mem_wb, mem_en, comenzar, fin;
#pragma intbits 8
    int a, b, i, j, mul_m1_mn, mul_m1_mr;
    int mem_addr, mem_data;
#pragma intbits 16
    int c, z, mul_m1_res;

    inputport(comenzar); inputport(mul_m1_res);
    bus_port(mem_data);
    outputport(fin);outputport(mem_ce);
    outputport(mem_en);outputport(mem_wb);
    outputport(mem_addr);
    outputport(mul_m1_mn);
    outputport(mul_m1_mr);
    portflags(mul_m1_mn, 0x1);
    portflags(mul_m1_mr, 0x1);

    … // no cambia nada hasta la multiplicacion

        mul_m1_mn=a;
        mul_m1_mr=b;
        z=mul_m1_res;
        while(0){};
        c=z;

    … // no cambia nada mas
}
```

**Figura 5-28** Modificaciones al código del ejemplo de acceso a memoria externa con Transmogrifier C.

Por último generamos además una versión en VHDL que funcionalmente se comporta como la versión implementada por Handel C. Las fuentes se pueden ver en el Apéndice F pág. 237.

Resultados
Chip EPF10K20RC240-4 (ALTERA)

**Tabla 5-6** Cuadro comparativo de velocidad y recursos para ejemplos con memoria externa y multiplicadores.

| | FF | Celdas Lógicas | Frec. Max. [ MHz ] | Tclk por loop (*) | Tiempo de cálculo [ ns ] |
|---|---|---|---|---|---|
| **Hcc** | 57 | 221 | 11.18 | 5 | 447 |
| **Tmcc** | 83 | 391 | 11.83 | 8 | 676 |
| **Tmcc (lpm_mult)** | 82 | 298 | 13.67 | 8 | 585 |
| **Vhdl** | 56 | 276 | 15.43 | 5 | 324 |

(*) En el programa ejecutamos un loop 128 veces, en este hacemos 2 lecturas a memoria, multiplicamos los datos leídos y escribimos el resultado en dos posiciones de memoria. Tclk representa cuantos períodos de reloj demora cada loop.

Las construcciones provistas por Handel C resultan más prácticas y sencillas de utilizar frente a las opciones vistas para Transmogrifier C pero ambas presentan resultados similares.

### Acceso a memoria interna con FIFO

A los efectos de mostrar como usar las diferentes opciones de memorias RAM interna ideamos un ejemplo que emula una memoria tipo FIFO. Comenzamos generando un diseño en VHDL para usar la megafunción lpm_fifo a los efectos de crear un modelo de comportamiento a seguir para los diseños en Handel C y Transmogrifier C. Usamos sólo los parámetros requeridos y estos los ajustamos para tener un tamaño de 64 palabras de 8 bits cada una.

A continuación mostramos una descripción de los puertos que utilizamos y el nombre de las señales de la mega función en que se mapean.

| Señal lpm_fifo | Señal de nuestro diseño | Tipo | Descripción |
|---|---|---|---|
| Sclr | reset_fifo | I | Reset síncrono de la fifo. |
| Wrreq | escribir | I | Pedido de escritura, si la fifo está llena se deshabilita. |
| Rdreq | leer | I | Pedido de lectura, si la fifo está vacía se deshabilita. |
| Data | datoIn | I | Datos de entrada, se guardan en la fifo usando wrreq. |
| Q | datoOut | O | Datos de salida, se leen de la fifo usando rdreq. |
| Full | fifo_llena | O | Cuando está en 1 indica que la fifo está llena. |
| Empty | fifo_vacia | O | Cuando está en 1 indica que la fifo está vacía. |
| Usedw | esp_ocup | O | Cantidad de palabras guardadas en la fifo. Si está en 0 puede ser que la fifo este llena, hay que ver también fifo_llena. |

El código de esta versión en VHDL se puede observar en el Apéndice F pág. 238.

### Handel C versión 1

La primer versión en Handel C usa la memoria interna provista por el lenguaje (tipo ram), que como ya vimos usa flip-flop y no celdas de memoria.
Como queremos que este diseño se comporte igual que su dual en VHDL tiene que responder en un periodo de reloj, esto es activar salidas (de datos y/o de control) y aceptar entradas (de datos y/o control) en cada periodo.

El primer problema que encontramos fue como leer las entradas “leer” y “escribir”, la forma usual de trabajar con entradas es ingresarlas en una variable y luego operar con la variable. Si utilizamos esta forma se haría muy difícil (si no imposible) activar las salidas fifo_llena, fifo_vacia y deshabilitar las entradas “leer”, “escribir” inmediatamente después que se da la condición para hacerlo. En su lugar lo que hicimos fue con las entradas controlar 5 variables:

| | |
|---|---|
| hab_lec, hab_esc | habilitaciones para leer y escribir. |
| wrreq, rdreq | pulsos internos de escritura y lectura que controlan la carga en la RAM, la salida de datos y el incremento de los contadores de lectura y escritura. |
| usedw | palabras usadas en la RAM. |

Si la carga en la RAM dependiera también de dichas entradas estas deberían cumplir con tiempos de setup de unos 40 ns (chip EPF10K20RC240-4), mientras que como las estamos usando solo se requieren 10 ns (mismo chip).

Luego, nos encontramos con que no podíamos reproducir el comportamiento de la lpm_fifo en el caso en que leer y escribir se activan simultáneamente. Al trabajar con memorias RAM, no podemos acceder a dos direcciones al mismo tiempo (una para leer y otra para escribir) entonces usamos dos ciclos de reloj, en el primero se lee el dato de la FIFO y en el segundo se escribe un dato en la FIFO. Si la lectura o escritura están deshabilitados no se usan los dos ciclos de reloj sino que en uno se realiza la operación permitida.

Para dar las salidas fifo_llena y fifo_vacia usamos las variables hab_esc y hab_lec. Cuando deshabilitamos la escritura es porque la FIFO está llena y cuando deshabilitamos la lectura es porque está vacía. Como los bits bajos de usedw se copian en la salida esp_ocup, concatenando fifo_llena y esp_ocup se tiene la cantidad exacta de palabras usadas en la FIFO.

Otro problema encontrado fue al hacer lo siguiente:

```
par{
        ...
        datoIn ? ent;
        datoOut ! (rdreq ? x[contL] : 0);  /* si rdreq es 1 devuelvo contenido
                                              de la RAM, sino saco un 0 */
        if (wrreq) x[contE]= ent;  // si wrreq es 1 guardo dato en la RAM
        ...
}
```

El problema está en las escrituras a la RAM pues las direcciones no se forman correctamente. Como recordaremos del capítulo 3 el selector de direcciones de la RAM (SelRam) tiene como señales de control a las variables de estado donde se realizan las escrituras y lecturas, y de éstas solamente una puede activarse por ciclo de reloj. También recordemos que la composición paralela para este ejemplo determina que el circuito tenga una sola variable de estado para las instrucciones del bloque. Por otro lado recordando la lógica generada para el IF vemos que la variable de estado para la escritura de la RAM se genera con un AND de la condición de test y la variable de estado del bloque paralelo. Entonces el SelRam va a tener, cuando la condición de test es verdadera, dos señales de control activas simultáneamente provocando una generación errónea de las direcciones.

Esto se soluciona haciendo como vemos en la siguiente figura.

```
par{
        datoIn ? ent;
        if (wrreq) x[contE]=ent; // si wrreq es 1 guardo dato en la RAM
        else if (rdreq) datoOut ! x[contL];  /* si rdreq es 1 => devulevo
                                                contenido de la RAM  */
        else delay;
        ...
}
```

A continuación, en la Figura 5-29, mostramos el código de esta primera versión.

```
// fifo usando la ram provista por handelC
const spec tgt={  //espec. del target
   fpga_type="Xilinx4000",
   clock_pad="P160"};

void main( target=tgt,
       port (in) escribir: 1, //decl. de puertos
       port (in) leer : 1,
       port (in) reset_fifo: 1,
       port (out) fifo_llena:1,
       port (out) fifo_vacia:1,
       port (in) datoIn : 8,
       port (out) esp_ocup : 6,
       port (out) datoOut : 8){

   int ent, sal : 8; /* variables para almacenar los datos */
   int contE, contL:6; /* contadores de  escritura y lectura */
   int usedw:7; //palabras usadas
   int wrreq, rdreq:1; // pulsos internos de lectura/escritura
   int hab_lec, hab_esc:1,//señales de habilitación/deshab.
   ram x [64]:8;  // ram interna

   par{
      while(1){
         par{
            esp_ocup ! usedw.(0..5); /* escribo  en puerto */
            //escribo o leo en la ram segun wrreq
            if (wrreq) x[contE]=ent;
            else if (rdreq) datoOut ! x[contL];
            else delay;
            fifo_llena ! hab_esc; //escribo puerto
            fifo_vacia ! ~hab_lec; //escribo  puerto
            // asignaciónes cond. según reset_fifo  , wrreq/rdreq
               contE=(reset_fifo ? 0 : (wrreq?contE+1:contE));
               contL=(reset_fifo ? 0 : (rdreq ?contL+1:contL));
            }
         };
         while(1){
            if (reset_fifo==1)
               hab_lec,hab_esc,wrreq,rdreq,usedw=0,0,0,0,0;
            else par{
               datoIn ? ent; //registro datoIn
               case(hab_lec @ leer @ ~hab_esc @ escribir){
                     /* hab_esc la trabajo al reves que hab_lec
                      xq'  sino al principio indica fifo_llena */
                     12,13,14:wrreq,rdreq,usedw=0,1,usedw-1;
                     3, 7,11: wrreq,rdreq,usedw=1,0,usedw+1;
                     15:{
                        wrreq, rdreq = 0, 1;
                        wrreq, rdreq = 1, 0;
                      }
                     default : wrreq, rdreq = 0, 0;
               };
             /*deshabilito lectura cuando queda una palabra  y se
               hace una lectura o cuando no quedan  palabras */
               hab_lec=(((usedw==1)&leer)|(usedw==0)?0:1);
            /*deshabilito lectura cuando queda un espacio y se
               hace una escritura o cuando no quedan espacios */
               hab_esc=(((usedw==63)&escribir)|usedw.6?1:0);
            }
         };
      }
   }
```

**Figura 5-29** Código de memoria FIFO para Handel C.

## Handel C versión 2

La segunda versión usa la megafunción de Altera lpm_ram_dq. La estructura del programa es igual a la de la versión anterior, la diferencia está en como trabajamos con las señales de la memoria. En la siguiente figura se observa parte del cuerpo del programa mostrando los cambios respecto a la versión anterior. El código completo puede verse en Apéndice F pág. 238.

```
void main(
   …  // el resto de los puertos es igual
   port (out) ram_fifo_we : 1,
   port (out) ram_fifo_din : 8,
   port (in)  ram_fifo_dot : 8,
   port (out) ram_fifo_dir : 6){
…
int rdreq_sh:1   //se agrega esta variable (rdreq desplazada) y se saca ram x[]
…
   par{
      while(1){
         par{
            esp_ocup ! usedw.(0..5);   /* escribo  en puerto */
             //escribo o leo en la ram segun wrreq
            ram_fifo_din ! ent;  // asigno ent a din de la ram
            ram_fifo_dir ! (wrreq ? contE : contL);  /* si wrreq=1 dir=contE si no dir=contL*/
            ram_fifo_we ! wrreq;  /* pulso de we de la ram, coincide con wrreq*/
            datoOut ! (rdreq_sh ? ram_fifo_dot : 0);
            rdreq_sh = rdreq;
            fifo_llena ! hab_esc;   //escribo puerto
… // el resto del programa es igual
```

**Figura 5-30** Modificación al código de memoria FIFO para Handel C.

Es necesario usar dos loops (while) en paralelo debido a que queremos que las asignaciones del primero se ejecuten en cada ciclo de reloj y el segundo en ocasiones (cuando wrreq = 1 y rdreq = 1) dura dos períodos de reloj.

En este caso las salidas demoran dos ciclos de reloj en actualizarse, o sea que en el segundo flanco de reloj después que, por ejemplo, se pide un dato (sin contar el flanco en el que se realiza el pedido) están los datos válidos en la salida. La salida de datos la conectamos a la salida de la lpm_ram_dq, intercalando una variable se obtendría una salida mucho más "limpia" sin embargo el circuito demoraría un ciclo de reloj más en actualizar esta salida. Además si lo que importa es el valor en el flanco de reloj, no es necesario hacer esto ya que el valor es el correcto.

Para compilar para Xilinx (usando módulos Sync_ram) hubo que hacer algunas modificaciones. Primero renombrar las señales con la RAM, de ram_fifo_xxx pasamos a ram_F0_fifo_xxx. El módulo SYNC_RAM es diferente a la lpm_ram_dq con reloj de entrada, en ésta al usar el reloj se registran tanto los datos de entrada y el WE como las direcciones, en este módulo solo se registran los datos y el WE y en la salida de datos se muestra siempre el contenido al que apuntan las direcciones.

Al usar la lpm_ram_dq, en las lecturas cargábamos la salida de la RAM en datoOut un ciclo de reloj después de poner las direcciones correctas en la RAM. Ahora, con SYNC_RAM, tenemos que hacerlo todo en el mismo ciclo, como vemos en la siguiente figura:

| Versión para Altera | Versión para Xilinx |
|---|---|
| …<br>ram_fifo_din ! ent;<br>ram_fifo_dir ! (wrreq ? contE : contL);<br>datoOut ! (rdreq_sh ? ram_fifo_dot : 0);<br>rdreq_sh = rdreq;<br>… | …<br>ram_F0_fifo_din ! ent;<br>ram_F0_fifo_dir ! (wrreq ? contE : contL);<br>ram_F0_fifo_we ! wrreq;<br>datoOut ! (rdreq ? ram_F0_fifo_dot : 0);<br>… |

**Figura 5-31** Modificación al código de memoria FIFO para Handel C según familia de chips.

Esta versión requiere un paso adicional, hay que crear un módulo SYNC_RAM de nombre F0_RAM, ancho de bus 8 y profundidad 64. Esto se hace en la herramienta LOGIBLOX del paquete Foundation.

## Transmogrifier C

En este punto hubo que modificar sustancialmente el programa. Las primeras modificaciones fueron porque el TMCC no acepta instrucciones CASE y tampoco los operadores de asignación condicionales ( _?_:_ ). Estos se sustituyeron por cláusulas IF – ELSE como puede observarse en la Figura 5-32.

```
main(){
   int escribir, leer, reset_fifo, fifo_llena;
   int fifo_vacia, int wrreq, rdreq, rdreq_sh,
   int ram_fifo_we, segunda, seg_sh;
   int hab_esc, hab_escQ, hab_lecQ, hab_lec;
#pragma intbits 8
   int datoIn, datoIn_sh, datoOut;
   int ram_fifo_din, ram_fifo_dot;
#pragma intbits 7
   int usedw;
#pragma intbits 6
   int contE, contL, ram_fifo_dir, esp_ocup;

   inputport(datoIn); inputport(escribir);
   inputport(leer);inputport(reset_fifo);
   outputport(datoOut); outputport(fifo_llena);
   outputport(fifo_vacia); outputport(esp_ocup);
   outputport(ram_fifo_dir);inputport(ram_fifo_dot);
   outputport(ram_fifo_we);outputport(ram_fifo_din);
// declaro datoIn como registrado
   portflags(datoIn, 0x3);
// declaro puertos varios como no registrados
   portflags(fifo_llena, 0x1);  portflags(fifo_vacia, 0x1);
   portflags(datoOut, 0x1);  portflags(esp_ocup, 0x1);
   portflags(ram_fifo_dir, 0x1);
   portflags(ram_fifo_din,0x1);
   portflags(ram_fifo_we,0x1);

   while(1){
      if (reset_fifo == 1) {
         contE = 0; contL = 0;
         esp_ocup = 0; fifo_llena = 0;
         fifo_vacia = 1; usedw = 0;
         hab_esc = 0; hab_lec = 0;
      }
      else{
         /* si en el ciclo anterior wrreq=rdreq=1 entonces
            uso datoIn_sh, si no uso datoIn */
         if (seg_sh==1)  ram_fifo_din=datoIn_sh;
         else   ram_fifo_din=datoIn;
         if (wrreq){
            ram_fifo_dir= contE;
            contE++;}
         else {ram_fifo_dir = contL;};
         if (rdreq) contL++;
         /* si en el pulso anterior rdreq=1 doy como salida
            ram_fifo_dot */
         if (rdreq_sh) {datoOut=ram_fifo_dot;}
         else{    datoOut=0;};
         ram_fifo_we=wrreq;
         esp_ocup = usedw;
         fifo_vacia = ~hab_lec; //esc. en puerto
         fifo_llena = hab_esc;  //esc. en puerto
         rdreq_sh = rdreq;  // rdreq_sh = rdreq retrasada
         seg_sh = segunda;
         datoIn_sh = datoIn;
         if (((usedw==63) & (escribir==1)) | usedw==64){
            /* si queda un espacio y escribo o si no  queda
               espacio se deshab. la escritura */
            hab_esc = 1;
            hab_lec = 1;
         }
         else if (((usedw==1) & (leer==1)) | (usedw==0)){
            /* si queda una palabra y leo o si no quedan
               palabras se deshab. la lectura */
            hab_esc = 0;
            hab_lec = 0;
         }
         else {
            hab_esc = 0;
            hab_lec = 1;
         };
         /* genero pulsos internos de lec/esc y actualizo
            palabras usadas segun  habilitaciones y puertos */
         if ((~hab_escQ & escribir &
                  ((~hab_lecQ)|(~leer))&(~segunda))==1){
            wrreq=1; rdreq=0; usedw=usedw+1; segunda=0;
         }
         else if ((hab_lecQ & leer &
                  ((~escribir)|(hab_escQ))&(~segunda))==1){
            wrreq=0; rdreq=1; usedw=usedw-1; segunda=0;
         }
         else if (((hab_lecQ & leer & ~hab_escQ& escribir) |
                                                  segunda)==1){
            wrreq=segunda; rdreq=~segunda;
            segunda=~segunda;
         }
         else{    segunda=0;
                  wrreq=0; rdreq=0;
         };
         hab_escQ = hab_esc;
         hab_lecQ = hab_lec;
      };
   };
}
```

**Figura 5-32** Código de memoria FIFO para Transmogrifier C.

En las versiones para Handel C para formar el nuevo valor de usedw usábamos hab_lec (hab_esc) y viceversa. Esto funcionaba en paralelo sin problemas. Sin embargo en el TMCC cuando queremos que use la salida de un registro (variable) para formar otro, tenemos que usarlo antes de asignarle un nuevo valor. Para aclarar un poco veamos un ejemplo:

```
if (hab_lec & leer) usedw = usedw – 1;
if (usedw == 0) hab_lec = 0;
else hab_lec = 1;
```

Esto así como está es lo mismo que hacer lo siguiente:

```
if (hab_lec & leer) usedw = usedw – 1;
if (usedw -1 == 0) hab_lec = 0;
else hab_lec = 1;
```

La primera asignación (usedw) usa correctamente el valor anterior de hab_lec, sin embargo en la segunda se usa el resultado de esta y no el valor de usedw anterior. Para solucionarlo introducimos una nueva variable.

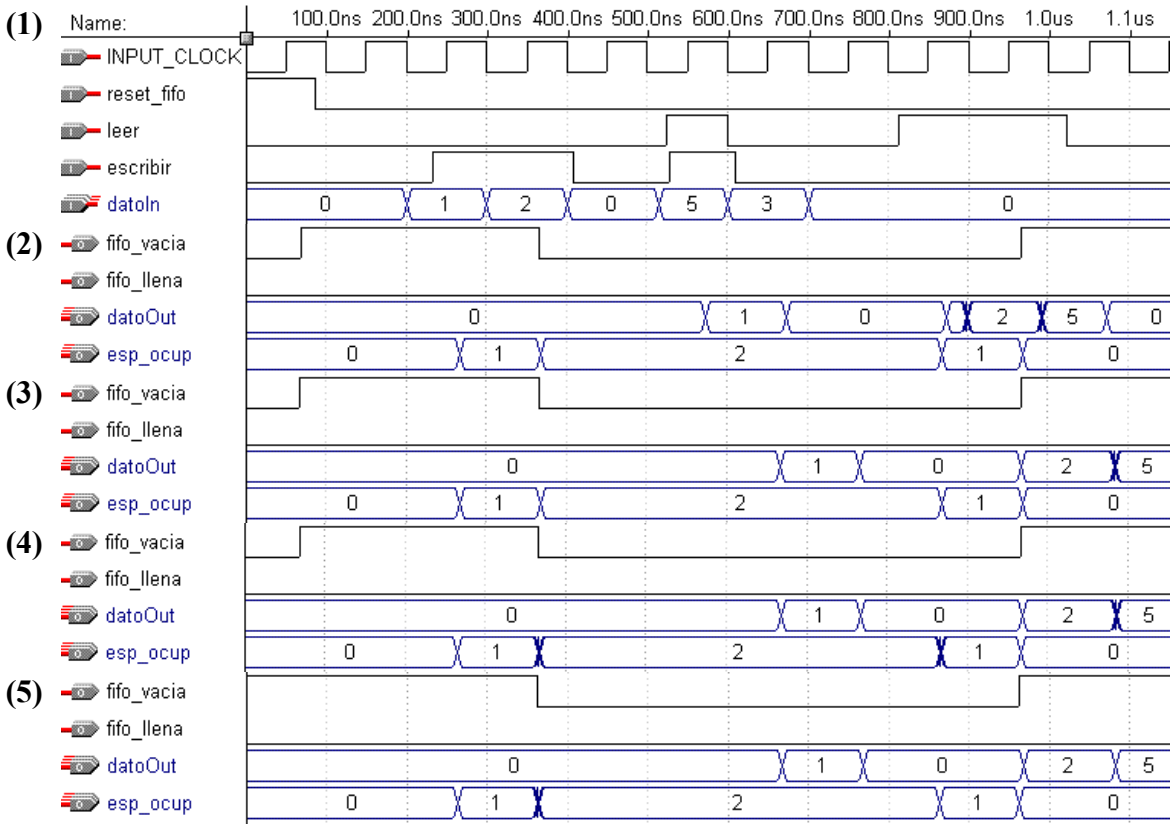
```
usedwQ = usedw;
if (hab_lec & leer) usedw = usedw – 1;
if (usedwQ == 0) hab_lec = 0;
else hab_lec = 1;
```

Una precaución que hay que tener en el caso del Transmogrifier C es siempre asignarles un valor a las señales hacia la RAM, para el caso de salidas no registradas. O sea que si la asignación de una de ellas está en un IF en la cláusula ELSE también asignarle un valor, aunque sea 0. En caso de no hacerlo estas señales se quedan oscilando y se puede llegar a escribir un dato en la RAM sin desearlo, además esta oscilación enlentece mucho las simulaciones.

**VHDL, versión 2**

Generamos una nueva versión en VHDL pero esta vez no utilizamos la megafunción lpm_fifo sino la lpm_ram_dq que posee las mismas limitaciones en el acceso a los datos que los diseños con HCC y TMCC. Las fuentes se pueden ver en el Apéndice F pág. 240.

Simulaciones




(1) Entradas
(2) Salidas de la versión HCC usando ram[]
(3) Salidas de la versión HCC usando lpm_ram_dq
(4) Salidas de la versión TMCC
(5) Salidas de la versión VHDL usando lpm_ram_dq

Resultados

**Tabla 5-7** Cuadro comparativo de resultados y rendimiento para memoria FIFO.

| **ALTERA - chip EPF10K20RC240-4** | | | | | |
|---|---|---|---|---|---|
| | **FF** | **Celdas lógicas** | | **Tclk (*)** | **Frec. Max. [ MHZ ]** |
| **VHDL (lpm_fifo), ver. 1** | 154 | 273 | | 1 | 28.90 |
| **Handel C (ram[ ]), ver. 1** | 547 | 1022 | | 1 | 21.50 |
| **Handel C (lpm_ram_dq), ver. 2** | 40 | 117 | | 2 | 28.00 |
| **Transmogrifier C** | 46 | 147 | | 2 | 26.70 |
| **VHDL (lpm_ram_dq), ver. 2** | 34 | 78 | | 2 | 28.08 |
| **XILINX - chip XC4010EPC84-3** | | | | | |
| | **FF** | **3-LUT** | **4-LUT** | **Tclk** | **Frec. Max. [ MHZ ]** |
| **Handel C (Sync_ram), ver. 2** | 39 | 31 | 159 | 1 | 26.66 |
| **VHDL (Sync_ram), ver. 2** | 26 | 21 | 81 | 2 | 38.69 |

(*) Tclk representa los flancos de reloj entre el pedido y los datos válidos en la salida.

Es más sencillo trabajar con la RAM provista por Handel C, ya que el compilador genera automáticamente las señales de control de la RAM. Sin embargo esta solución es mucho más costosa en hardware y más lenta que usar la lpm_ram_dq de Altera. Además como el Transmogrifier C no soporta el uso de memorias internas la opción a usar estas mediante el traductor es, en este ejemplo, usar 64 variables para almacenar los datos, con la complejidad asociada de leer y escribir datos en la variable deseada.

### 5.2.3 Multiplicador con Pipeline.

La aplicación de la técnica de segmentado o pipeline es útil cuando deseamos incrementar el flujo de datos o throughput de nuestros diseños. No necesariamente implica que nuestros diseños tendrán relojes más rápidos, dependerá de su forma de aplicación.

¿Dónde se aplica? Un caso es cuando tenemos caminos de datos con varios niveles lógicos entre registros (salidas de flip-flop y entradas de otros flip-flop) esto provoca que trabajemos con relojes más lentos para poder respetar los tiempos de setup sumado a los tiempos de propagación entre los niveles. Estos caminos reciben el nombre de camino critico o critical path.

Lo que se hace en estos casos es intercalar entre éstos niveles registros de FF, de esta forma obtenemos ciclos de reloj más rápidos y consecuentemente el aumento de throughput de los datos. Igualmente hay que tener consideraciones especiales para el resto del circuito pues ahora este camino aumento su latencia.

Otro caso de aplicación es cuando los diseños son funcionalmente separables en etapas entonces podemos aplicar la técnica para ir ejecutando las distintas etapas en paralelo intercalando entre ellas registros de FF para guardar los resultados intermedios. Si bien esto no necesariamente aumenta la velocidad del reloj si se aumenta el flujo de datos de salida pues se podría contar con datos válidos en cada período de reloj, pero no olvidemos que también aumento la latencia del diseño.

Una ventaja particular de los FPGA es que son ricos en flip-flop y por tanto el aumentar su uso (por intercalar registros) no se generarían grandes costos en términos de los recursos del chip. [5]

¿Cómo aplicarlo al HCC? Aquí no queda tan clara la incidencia de las líneas de código y el número de niveles lógicos entre registros. Podemos llegar a descartar la lógica necesaria para la transición de los estados pues la máquina generada utiliza la codificación OHE (One Hot Enconded). Entonces los niveles quedarían determinados por las instrucciones en sí, por tanto tendríamos que conocer la forma de construir los circuitos y tratar de utilizar instrucciones con pocos niveles lógicos. En tal caso la aplicación de la técnica para aumentar la velocidad del reloj esta muy limitada, siendo más factible la posibilidad de armar algoritmos que trabajen en forma segmentada.

Veamos un ejemplo, el siguiente segmento del código realiza la operación (a*b) + c y devuelve su resultado con un delay y latencia de 3 ciclos.

```
entradaC ? c;
while(1) {
        entradaA ? a;
        entradaB ? b;
        res ! (a * b) + c;
}
```

**Figura 5-33** Ejemplo de aplicación de pipeline en HCC. Parte 1.

Podemos mejorar el delay y la latencia del circuito paralelizando una porción del código, como se puede apreciar a continuación.

```
while(1) par {
        entradaA ? a;
        entradaB ? b;
        res ! (a * b) + c;
}
```

**Figura 5-34** Ejemplo de aplicación de pipeline en HCC. Parte 2.

Ahora contamos con un circuito que posee un delay de 1 ciclo y una latencia de 2 ciclos. También observamos que el camino entre los registros de "a", "b" y "c" y el registro de "salida" es crítico (suponemos que el resto de programa no influye) entonces podemos intercalar un registro de la siguiente forma:

```
while(1) par {
        entradaA ? a;
        entradaB ? b;
        temp = a * b;
        res ! temp + c;
}
```

**Figura 5-35** Ejemplo de aplicación de pipeline en HCC. Parte 3.

Este cambio seguramente mejore la velocidad máxima de nuestro diseño (dependerá también del resto del programa) pero aumentamos la latencia de los datos. Si este último cambio lo hubiésemos realizado en el primer código (Figura 5-33) también habríamos aumentado la frecuencia máxima pero el delay y la latencia pasarían a ser de 4 ciclos.

La idea general tras este ejemplo es indicarle al HCC que paralelizamos porciones del código y utilizamos variables auxiliares (que se van a convertir en registros) para guardar los resultados intermedios. Esta es la idea que aplicamos en los ejemplos que veremos a continuación, donde se eligió como algoritmo el producto sin signo de dos enteros.

Todas las pruebas se realizaron para el compilador HCC y los datos obtenidos fueron extraídos del Max+PlusII ver. 10.1 para la síntesis en el chip 10K50RC240-3 de Altera. Las fuentes completas de todos los ejemplos vistos en este punto están en el Apéndice F a partir de la pág. 241.

El esquema general del ejemplo se puede apreciar en la siguiente figura, donde trabajamos con dos enteros de 8 bits a la entrada y devolvemos una salida de 16 bits, ambos con registros de flip-flop.


**Figura 5-36** Diagrama en bloques de la estructura básica de los ejemplos de uso de pipeline.

**Ejemplo 1.**
En este ejemplo básico nuestro diseño lee y guarda en un registro los datos de entrada, luego realiza el producto sin signo y guarda el resultado en otro registro, por último devuelve los resultados por los pines de salida.

El cálculo del producto se realiza con el operador producto implementado por el compilador HCC, como se puede apreciar en las fuentes que listamos a continuación.

```
const spec yyy = {
    fpga_type  = "Xilinx4000",
    fpga_chip  = "4003APC84-6",
    clock_pad = "P13"
    };
const spec ent={ data={} };
const spec sal={ data={} };
const dw=8;
const ddw=dw*2;
const zero=0:dw;

main ( target = yyy,
    port (in) mp=ent :dw,
    port (in) mr=ent :dw,
    port (out) res=sal : ddw)
```

```
{
    int ta,tb:dw;
    int tr : ddw;

    while (true) par {
                    // leo operandos de los puertos
                    mp ? ta;
                    mr ? tb;
                    tr = (ta .*. tb);
                    // devuelvo resultado por el puerto
                    res ! tr;
    };
}
```

**Figura 5-37** Código de ejemplo 1 de multiplicador con pipeline.

A continuación mostramos los resultados obtenidos de la síntesis.

**Tabla 5-8** Resultados de la síntesis del ejemplo 1 multiplicador con pipeline.

| | FF | Celdas Lógicas | Frec. Max. [ MHz ] | Delay (clk) | Latencia (clk) |
|---|---|---|---|---|---|
| **Ejemplo 1** | 34 | 181 | 12.54 | 1 | 2 |

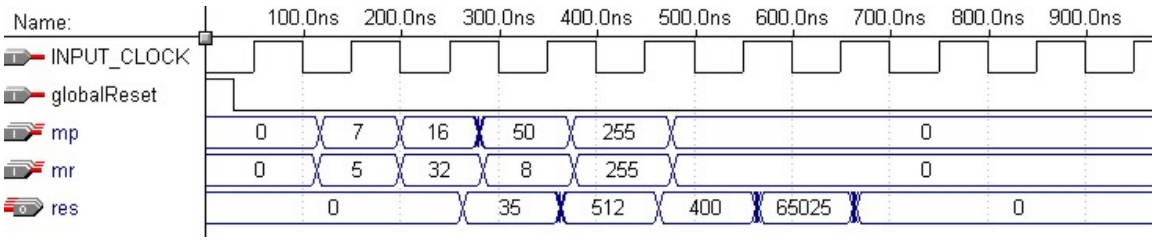



**Figura 5-38** Simulación del ejemplo 1 de uso de pipeline

**Ejemplo 2.**
Aquí analizando el camino crítico del ejemplo anterior observamos que, como era de esperarse, estaba en la implementación del producto y como no tenemos control sobre esta implementación decidimos hacer la nuestra para luego poder intercalar etapas de pipeline fácilmente y así estudiar su efecto en la performance del diseño. Esta implementación consta de rotaciones y sumas, como se describe en el Apéndice A.

Entonces tenemos que el diseño de todo el sistema es similar al anterior pero con una implementación "a mano" del producto. Listamos a continuación solamente la parte del programa principal que fue cambiada.

```
… // igual que antes
main ( ... // igual que antes )
{
  int ta:dw;
  // se agregaron estas declaraciones de variables
  int temp0,temp1,temp2,temp3,temp4 : ddw;
  int temp5,temp6,temp7,tb,tr : ddw;

  res ! (zero @ a);
  while (true) par {
                mp ? ta;
                mr ? tb;
                {  par {  temp0 = (ta.0 ? (zero @ b) : 0); // realizo rotaciones
                          // si el bit 1 de ta es 1 entonces temp1 es b rotado 1 vez
                          temp1 = (ta.1 ? (zero @ b) << 1 : 0);
                          temp2 = (ta.2 ? (zero @ b) << 2 : 0);   // idem anterior pero para bit 2
                          temp3 = (ta.3 ? (zero @ b) << 3 : 0);
                          temp4 = (ta.4 ? (zero @ b) << 4 : 0);
                          temp5 = (ta.5 ? (zero @ b) << 5 : 0);
                          temp6 = (ta.6 ? (zero @ b) << 6 : 0);
                          temp7 = (ta.7 ? (zero @ b) << 7 : 0); };
                   // sumo todos los resultados parciales
                   tr= temp0+temp1+temp2+temp3+temp4+temp5+temp6+temp7;
                }; // fin de la implementacion del producto
                res ! tr;
  }; // fin del while
}
```

**Figura 5-39** Modificaciones al código del ejemplo 1 de multiplicador con pipeline.

**Tabla 5-9** Resultados de la síntesis del ejemplo 2 de uso de pipeline

| | LC | Flip Flops | Max. Frecuencia [ MHz ] | Delay (clk) | Latencia (clk) |
|---|---|---|---|---|---|
| **Ejemplo 2** | 101 | 216 | 13.72 | 2 | 4 |

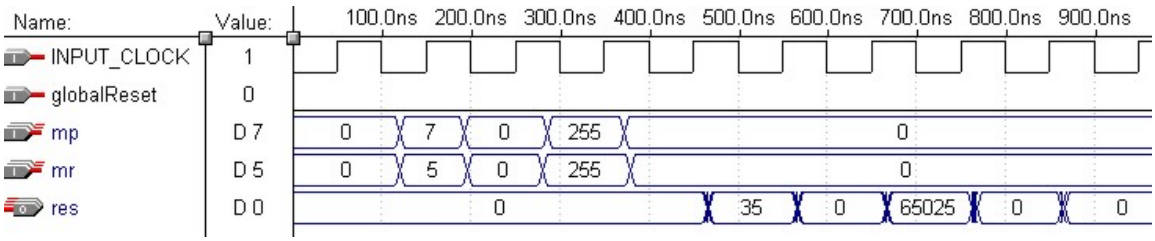



**Figura 5-40** Simulación del ejemplo 2 de uso de pipeline

## Ejemplo 3

Para mejorar el funcionamiento del circuito anterior intercalamos una etapa de pipeline (o sea un registro de FF) dentro del bloque que realiza el producto, el esquema de diseño queda entonces de la siguiente forma.

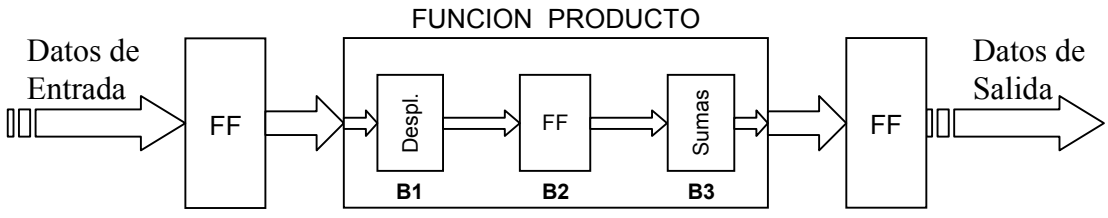



**Figura 5-41** Diagrama en bloques de la estructura del ejemplo 3 de uso de pipeline

El diseño hasta el momento consta de un pipeline de 3 etapas, el código completo se puede encontrar en el Apéndice F pág. 241. Aquí destacaremos solamente una parte del programa principal.

```
while (true) par {
                    mp ? ta;  // leo del Puerto el multiplicando
                    mr ? tb;  // leo del Puerto el multiplicador
                     // rotaciones
                    temp0 = (ta.0 ? (zero @ b) : 0);
                    temp1 = (ta.1 ? (zero @ b) << 1 : 0);
                    temp2 = (ta.2 ? (zero @ b) << 2 : 0);
                    temp3 = (ta.3 ? (zero @ b) << 3 : 0);
                    temp4 = (ta.4 ? (zero @ b) << 4 : 0);
                    temp5 = (ta.5 ? (zero @ b) << 5 : 0);
                    temp6 = (ta.6 ? (zero @ b) << 6 : 0);
                    temp7 = (ta.7 ? (zero @ b) << 7 : 0);
                    // sumas
                    tr= temp0+temp1+temp2+temp3+temp4+temp5+temp6+temp7;
                    res ! tr;  // salida del resultado por puerto
   };
```

**Figura 5-42** Porción de código del ejemplo 3 de multiplicador con pipeline.

El trabajar con las asignaciones paralelas nos permite agregar registros intermedios de forma de aumentar los niveles de pipeline, para ello tenemos que agregar variables auxiliares.

En este ejemplo se paralelizó la entrada de datos junto con las rotaciones, las sumas y la salida del resultado permitiéndonos así que el circuito recibiera datos en cada período de reloj. Las variables auxiliares temp0 a temp7 que serían equivalentes al registro de FF, bloque B2, de la Figura 5-41.

Veamos los resultados de la síntesis y simulación.

**Tabla 5-10** Resultados de la síntesis del ejemplo 3 de multiplicador con pipeline.

| | FF | Celdas Lógicas | Frec. Max. [ MHz ] | Delay (clk) | Latencia (clk) |
|---|---|---|---|---|---|
| **Ejemplo 3** | 98 | 212 | 13.33 | 1 | 3 |

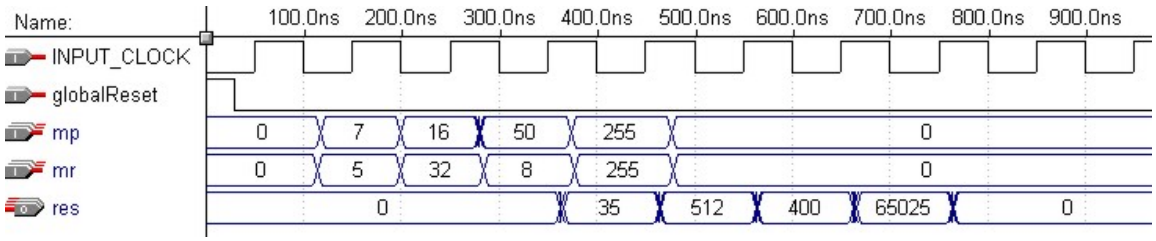



**Figura 5-43** Simulación del ejemplo 3 de multiplicador con pipeline.

## Ejemplo 4

Aquí con el interés de tener sumadores más simples así disminuir la lógica entre registros y poder aumentar la velocidad del reloj, decidimos agregar una etapa intermedia más al producto y pasar a trabajar en lugar de una suma de 8 términos, a dos etapas con sumas de 4 y 2 términos respectivamente. Como se puede apreciar en el extracto del código que mostramos a continuación.

```
while (true) {
        par {
            mp ? ta;
            mr ? tb;
            // rotaciones
            tmp0 = (ta.0 ? (zero @ tb) : 0);
            tmp1 = (ta.1 ? (zero @ tb) << 1 : 0);
            tmp2 = (ta.2 ? (zero @ tb) << 2 : 0);
            tmp3 = (ta.3 ? (zero @ tb) << 3 : 0);
            tmp4 = (ta.4 ? (zero @ tb) << 4 : 0);
            tmp5 = (ta.5 ? (zero @ tb) << 5 : 0);
            tmp6 = (ta.6 ? (zero @ tb) << 6 : 0);
            tmp7 = (ta.7 ? (zero @ tb) << 7 : 0);
             // primer nivel de sumas
            sum0 = tmp3 + tmp0 + tmp5 + tmp7;
            sum1 = tmp2 + tmp1 + tmp4 + tmp6;
            // Segundo nivel de sumas
            sum2 = sum0 + sum1;
            res ! sum2;  // devuelvo resultado por el puerto
        }
  };
```

**Figura 5-44** Modificaciones al código del ejemplo 3 de multiplicador con pipeline.

Nuestro diseño ahora tiene un pipeline de 4 etapas, las fuentes completas de este ejemplo se pueden ver en el Apéndice F pág. 242. El esquema del diseño es el siguiente:

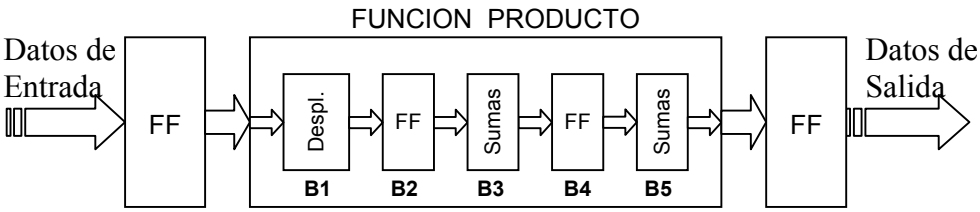



**Figura 5-45** Diagrama en bloques de la estructura del ejemplo 4 de multiplicador con pipeline.

Donde nuevamente las variables tmp0 a tmp7 son equivalentes al bloque B2, las variables sum0 y sum1 son equivalentes al bloque de registros B4 y por último el registro de FF de la salida estaría formado por la variable sum2.

Sus resultados en la síntesis y simulación son los siguientes:

**Tabla 5-11** Resultados de la síntesis del ejemplo 4 de multiplicador con pipeline.

| | FF | Celdas Lógicas | Frec. Max. [ MHz ] | Delay (clk) | Latencia (clk) |
|---|---|---|---|---|---|
| **Ejemplo 4** | 128 | 248 | 20.96 | 1 | 4 |

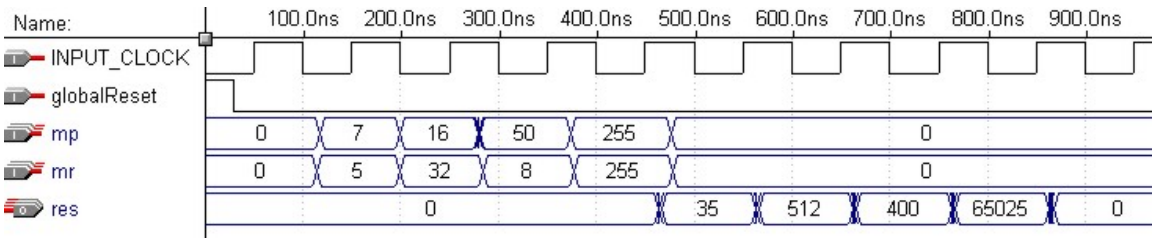



**Figura 5-46** Simulación del ejemplo 4 de multiplicador con pipeline.

## Ejemplo 5

Aquí aumentamos una vez más las etapas de sumadores para trabajar solamente con sumadores de dos términos. Veamos como queda la parte principal del código en esta nueva versión.

```
while (true) {
        par{
            mp ? ta;
            mr ? tb;
            // rotaciones
            tmp0 = (ta.0 ? (zero @ tb) : 0);
            … // igual que antes
            tmp7 = (ta.7 ? (zero @ tb) << 7 : 0);
             // primer nivel de sumas
            sum00 = tmp3 + tmp0 ;
            sum01 = tmp2 + tmp1 ;
            sum02 = tmp5 + tmp7;
            sum03 = tmp4 + tmp6;
             // Segundo nivel de sumas
            sum10 = sum00 + sum01;
            sum11 = sum02 + sum03;
             // tercer nivel de sumas
            sum2   = sum10 + sum11;
            res ! sum2;  // salida de resultado por puerto
        }
   };
```

**Figura 5-47** Modificaciones al código del ejemplo 4 de multiplicador con pipeline.

En el esquema de este circuito, ver Figura 5-48, podemos identificar los bloques de registros: B2 con las variables tmp0 a tmp7 (igual que antes), B4 con las variables sum00 a sum03, B6 con las variables sum10 y sum11, y el bloque de registros de la salida con la variable sum2.

Las fuentes completas de este ejemplo se pueden ver en el Apéndice F pág. 242.

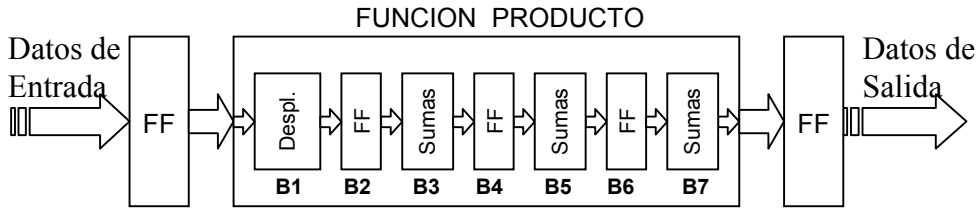



**Figura 5-48** Diagrama en bloques de la estructura del ejemplo 5 de multiplicador con pipeline.

Una vez compilado y simulado analizamos su camino crítico y vimos que los sumadores de 16 bits eran los que "retrasaban" al circuito. Decidimos realizar una variante al código, también con la idea de mejorar la performace de las sumas, y ajustar el tamaño de las variables para que las sumas sean del menor número de bits posible, por ejemplo cuando se rota una vez a uno de los términos, que son de 8 bits, no es necesario guardar su valor en una variable de 16 bits, como hacíamos hasta ahora, alcanza con guardarlo en una de 9 bits. Luego en la primer etapa de sumas la sumamos con otra que tenga solamente 10 bits de ancho, y así con el resto de los términos de la suma.
Esta variante la denominamos Ejemplo 5 versión 2 y se puede observar en el Apéndice F pág. 242.
También para este ejemplo decidimos explorar las distintas opciones del sintetizador para tratar de obtener una mayor performance y es así que utilizando la opción FAST de

síntesis en el M+PII, el circuito así compilado utiliza unas líneas especiales de carry entre celdas lógicas del chip lo que hace aumentar la velocidad del diseño como se puede observar en la siguiente tabla.

**Tabla 5-12** Resultados de la síntesis del ejemplo 5 de multiplicador con pipeline.

| | FF | Celdas Lógicas | Frec. Max. [ MHz ] | Delay (clk) | Latencia (clk) |
|---|---|---|---|---|---|
| **Ejemplo 5** ver. 1 | 167 | 281 | 24.5 | 1 | 5 |
| **Ejemplo 5** ver. 2 | 162 | 273 | 28.57 | 1 | 5 |
| **Ejemplo 5** ver. 2 (op. FAST) | 162 | 290 | 54.94 | 1 | 5 |

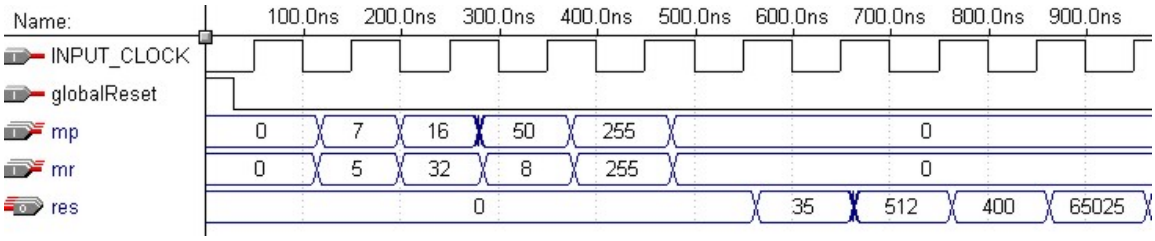



**Figura 5-49** Simulación del ejemplo 5 de multiplicador con pipeline.

## Ejemplo 6

Ahora cambiemos un poco el enfoque y veamos como el Max+Plus II implementa el operador producto, para ello realizamos un código en VHDL que utiliza la mega función lpm_mult aplicada al esquema general del diseño que observábamos anteriormente, esto es con un registro a la entrada y otro a la salida.

Generamos distintas versiones con distintos niveles de pipeline que se indican con facilidad a la mega función lpm_mult como un parámetro de entrada. La versión 1 no posee pipeline, la versión 2 consta de una etapa y la versión 3 consta de tres etapas. Las fuentes de estos diseños se pueden observar en el Apéndice F pág. 243 a pág. 244 respectivamente.

Al igual que para el ejemplo 5, decidimos ver el efecto de las opciones del sintetizador sobre el circuito final y aplicamos la misma opción usada en el ejemplo anterior a la versión 3 de este ejemplo.

En el siguiente cuadro se muestran los resultados de velocidad y área.

**Tabla 5-13** Resultados de la síntesis del ejemplo 6 de multiplicador con pipeline.

| | FF | Celdas Lógicas | Frec. Max. [ MHz ] | Delay (clk) | Latencia (clk) |
|---|---|---|---|---|---|
| **Ejemplo 6** ver. 1 | 32 | 185 | 17.6 | 1 | 2 |
| **Ejemplo 6** ver. 2 | 72 | 172 | 19.8 | 1 | 3 |
| **Ejemplo 6** ver. 3 | 160 | 201 | 32.57 | 1 | 5 |
| **Ejemplo 6** ver. 3 (op. FAST) | 160 | 160 | 103.9 | 1 | 5 |

**Cuadro comparativo final.**
Ahora presentaremos un cuadro comparativo de los ejemplos anteriores agrupando los resultados de aquellos que son funcionalmente iguales, esto es con igual latencia y delay.

**Tabla 5-14** Cuadro comparativo de las distintas versiones del multiplicador con pipeline.

| | FF | Celdas Lógicas | Frec. Max. [ MHz ] | Delay (clk) | Latencia (clk) |
|---|---|---|---|---|---|
| **Ejemplo 1** | 34 | 181 | 12.54 | 1 | 2 |
| **Ejemplo 6** ver. 1 (VHDL) | 32 | 185 | 17.6 | 1 | 2 |
| **Ejemplo 3** | 98 | 212 | 13.33 | 1 | 3 |
| **Ejemplo 6** ver. 2 (VHDL) | 72 | 172 | 19.8 | 1 | 3 |
| **Ejemplo 5** ver. 1 | 167 | 281 | 24.5 | 1 | 5 |
| **Ejemplo 5** ver. 2 | 162 | 273 | 28.57 | 1 | 5 |
| **Ejemplo 6** ver. 3 (VHDL) | 160 | 201 | 32.57 | 1 | 5 |
| **Ejemplo 5** ver. 2 (op. FAST) | 162 | 290 | 54.94 | 1 | 5 |
| **Ejemplo 6** ver. 3 (VHDL) (op. FAST) | 160 | 160 | 103.9 | 1 | 5 |

Podemos observar la influencia de los retardos introducidos por sumadores muy grandes, tanto sea en número de bits como en cantidad de términos, otro aspecto a destacar es el difícil control que se tiene desde el C del hardware generado dificultando poder acelerar el reloj de un diseño en particular. Lo que sí es posible es segmentar o particionar los diseños para que trabajen con varias etapas permitiendo así aumentar el flujo de datos, eso si la lógica no se complica demasiado y hace que tenga relojes más lentos.

Comparando con la megafunción vemos que en general es mejor que nuestro producto pero para algunos casos tampoco resultó una mejora notable de velocidad. Eso sí cuando se utilizan líneas especiales (carry chains) allí es donde se ve la real diferencia en la optimización del diseño que realiza el sintetizador.

Con respecto al área ocupada vemos que las distintas versiones del ejemplo para el HCC se mantuvieron con variaciones no muy significativas (variaciones en el entorno del 10%) y tampoco se observaron diferencias significativas con las implementaciones de la mega función (salvo el caso especial de la utilización de carry chains).

**Comentarios generales sobre la utilización de la técnica.**

- Aplicable a algoritmos que sean fácilmente segmentables o separables en etapas, logrando así aumentar el flujo de datos.
- No es muy buena técnica para aumentar la velocidad del reloj principal tratando de disminuir el número de niveles lógicos entre registros.
- Es un proceso complejo, el agregado de variables auxiliares para introducir las distintas etapas del pipeline hace que el código resultante sea más pesado y complejo de entender.

**Comentarios para el TMCC**
También es posible utilizar pipeline con el TMCC, valiendo las mismas consideraciones realizadas anteriormente. Hay que sí tener en cuenta la forma de escribir las sentencias pues ésta es una gran diferencia entre estos compiladores. Se puede apreciar la implementación del ejemplo del producto para el compilador TMCC en el Apéndice F pág. 244.

## 5.2.4 Operaciones con matrices: Producto entre elementos de dos matrices.

En este ejemplo se trabajó la multiplicación de una matriz de elementos variables por una matriz constante, término a término y la suma de todos los elementos resultantes. En el ejemplo tratamos solo el caso de matrices de 3x3.

La operación a realizar queda entonces:

$$Ro = (A0*B0)+(A1*B1)+(A2*B2)+(A3*B3)+(A4*B4)+(A5*B5)+(A6*B6)+(A7*B7)+(A8*B8)$$

Siendo Bi los elementos de la matriz constante con signo, Ai los elementos de la matriz de entrada y R0 es el resultado de 16 bits.

Las pruebas se realizaron con Handel C y fueron sintetizadas y simuladas para los chips de Altera de la familia Flex10k.

## Ejemplo 1

### Versión 1

En esta versión se utilizó RAM interna implementada en forma de flip-flop dentro del chip. Recordamos que el compilador HCC nos prohíbe acceder en el mismo tiempo de reloj a dos o más elementos de la memoria (vector), tampoco podemos definir arrays de dos dimensiones (int A[3,3]) por lo tanto deben ser de una sola dimensión (int A[9];). El tener estas limitaciones con los vectores y matrices, no facilita la programación y/o la adaptación de otros códigos ya escritos.

Guardamos los datos que ingresamos de forma iterativa al chip en A[i] y en otra memoria (B) los elementos constantes, definidos en tiempo de compilación. El resultado R0 es guardado en una variable y luego sacado fuera del chip a través de un puerto. La finalidad de utilizar memoria interna (tipo ram) en las dos matrices es poder iterar y simplificar la programación.

A continuación mostramos el código fuente del ejemplo.

```
const spec harp1 ={
        fpga_type= "Xilinx4000",
        clock_pad= "P13"   };

const dim = 3;
const dw = 8;
const ddw = 16;

main(   target=harp1,
        //defino puertos de entrada y salida
        port (in) datos  : dw,
        port (out) R0 : ddw)
{
        //defino dimensiones de la memoria interna
        ram int A[(dim*dim)] : dw;
        ram int B[(dim*dim)] : dw;
          int aux:ddw;
          int i : 4;
        // guardo en memoria el valor de la matriz B cte
        B[0] = -1; B[1] = 1; B[2] = 1; B[3] = -1; B[4] = 1;
        B[5] = -1; B[6] = 1; B[7] = -1; B[8] = 1;

        //Ingreso los valores de la matriz A (3x3)
        i=0;
        while (i.<.9) {par{
                i = i+1;
                datos ? A[i];} }
        //realizo las operaciones con un acumulador
        i,aux=0,0;
        while (i.<.9)
        {par { aux = aux + (A[i]*B[i]); i = i+1;}}
        R0 ! aux; // escribo el  resultado por el puerto R0
}
```

**Figura 5-50** Código del ejemplo 1 con memoria interna.

Como se puede apreciar en la Figura 5-51 al comienzo tenemos una demora de 11 ciclos que se deben a la inicialización de los datos de la matriz constante.

Resultados
Chip EPF10K20RC240-3

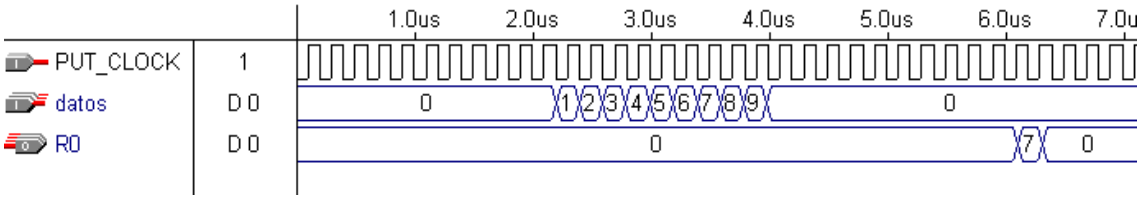



**Figura 5-51** Simulación del ejemplo 1, versión 1 con memoria interna.

**Tabla 5-15** Resultado de la simulación del ejemplo 1 versión 1 con memoria interna.

| Utilizando RAM interna | FF | Celdas Lógicas | Frec. Max [ MHz ] | Tclk Utilizados | Tiempo [ μs ] |
|---|---|---|---|---|---|
| **$B_i$ = ±1** | 193 | 446 (39%) | 10.66 | 32 | 3.00 |
| **$B_i$ = ±127** | 209 | 601 (52%) | 7.88 | 32 | 4.06 |

En la tabla observamos dos versiones del mismo programa, una con una matriz constante, B, con valores de 1 y –1 y la otra con valores de 127 y –127. La razón es ver el efecto en la performance del diseño pues el compilador realizará distintas optimizaciones. Esto lleva a que en particular se varíe la frecuencia máxima.

## Versión 2

Para esta otra versión la matriz de elementos constantes es guardada en variables. Para realizar las operaciones se sigue utilizando una variable auxiliar y el resultado es sacado fuera del chip a través de un puerto. Esta modificación tiene como finalidad ver la performance que tiene el sistema al cambiar la forma de guardar la matriz constante B.

El código de esta versión es el siguiente:

```
const spec harp1 ={
            fpga_type= "Xilinx4000",
            clock_pad= "P13"   };

const dim = 3;
const dw = 8;
const ddw = 16;

main ( target=harp1,
         port (in) datos  : dw,
         port (out) R0  : ddw) {

    ram int A[(dim*dim)] : dw;
    int aux:ddw;
    int i : 4;
    // valor de los elementos de la matriz B cte.
    int B0 = -1;int B1 = 1;int B2 = 1;int B3 = -1;int B4 = 1;
    int B5 = -1;int B6 = 1;int B7 = -1;int B8 = 1;
```

```
            //Ingreso los valores de la matriz A (3x3)
            while (i.<.9)
                par {datos ? A[i]; i=i+1; }

            //realizo las operaciones con un acumulador
            aux = (A[0]*B0) ;
            aux = aux + (A[1]*B1) ;
            aux = aux + (A[2]*B2) ;
            aux = aux + (A[3]*B3) ;
            aux = aux + (A[4]*B4) ;
            aux = aux + (A[5]*B5) ;
            aux = aux + (A[6]*B6) ;
            aux = aux + (A[7]*B7) ;
            aux = aux + (A[8]*B8) ;
            R0 ! aux;
}
```

**Figura 5-52** Código del ejemplo 1, versión 2 con memoria interna y variables para la matriz constante.

Resultados

**CHIP EPF10K20RC240-3**

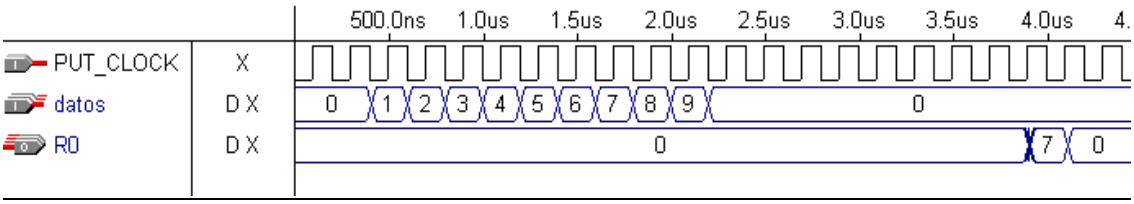



**Figura 5-53** Simulación del ejemplo 1, versión 2 con memoria interna y variables para la matriz constante.

**Tabla 5-16** Resultado del ejemplo 1, versión 2 con memoria interna y variables para la matriz constante.

| Utilizando | FF | Celdas Lógicas | Frec. Max [ MHz ] | Tclk Utilizados | Tiempo [ µs ] |
|---|---|---|---|---|---|
| **RAM interna Mat. B var** | 161 | 462 (40%) | 10.74 | 21 | 1.96 |

Esta solución aunque resultó ser un poco más rápida (en tiempo total de cálculo), es más tediosa de programar y ampliar ya que no es posible la iteración para llevar a cabo las operaciones.

## Versión 3

En esta versión utilizamos, en lugar de una memoria de 9 bytes para guardar la matriz (datos de entrada), tres memorias de 3 bytes, una por cada fila de la matriz de entrada. Guardamos la matriz constante B en variables ya que había sido la solución más rápida de las versiones anteriores.

Trabajando de esta forma tenemos la ventaja de poder acceder a distintos elementos de la matriz en el mismo período de reloj. También utilizamos una variable auxiliar para realizar las operaciones intermedias y sacamos el resultado fuera del chip.
La fuente de esta versión se puede apreciar a continuación.

```
const spec harp1 ={
            fpga_type= "Xilinx4000",
            clock_pad= "P13"
};
const dim = 3;
const dw = 8;
const ddw = 16;

main( target=harp1,
      port (in) datos : dw,
      port (out) R0 : ddw) {

      RAM int A1[dim] : dw;
      RAM int A2[dim] : dw;
      RAM int A3[dim] : dw;
      int i,j,k:2;
      int aux:ddw;

      int B0= -1; int B1= 1; int B2= 1; int B3= -1;
      int B4= 1;int B5= -1; int B6=1; int B7=-1;int B8=1;

      //Ingreso los valores de la matriz A(3x3)
      while (i.<.3)
            { par { datos ? A1[i];
                  i=i+1;}}
      while (j.<.3)
            { par { datos ? A2[j];
                  j=j+1;}}
      while (k.<.3)
            { par { datos ? A3[k];
                  k=k+1;}}

      // realizo las operaciones con acumulador
      aux = (A1[0]*B0) + (A2[0]*B3) + (A3[0]*B6);
      aux = aux + (A1[1]*B1) + (A2[1]*B4) + (A3[1]*B7);
      aux = aux + (A1[2]*B2) + (A2[2]*B5) + (A3[2]*B8);
      R0 ! aux;
}
```

**Figura 5-54** Código del ejemplo 1, versión 3 con tres memorias internas.

Resultados
Chip EPF10K20RC240-3

**Tabla 5-17** Resultado del ejemplo 1, versión 3 con tres memorias internas.

| Utilizando | FF | Celdas Lógicas | Frec. Max. [ MHz ] | Tclk Utilizados | Tiempo [ μs ] |
|---|---|---|---|---|---|
| **RAM interna 3 memorias** | 127 | 546 (47%) | 10.78 | 15 | 1.39 |

Se aprecia la mejora en el tiempo total de cálculo debido al acceso de varios elementos de la matriz al mismo tiempo.

## Versión 4

En este caso utilizamos variables para guardar la matriz de entrada. Cada dato es guardado en una variable distinta, por este motivo no podemos hacer iteraciones pero sí podemos realizar las operaciones de suma y producto en paralelo. Utilizamos una variable auxiliar para realizar las operaciones que luego es escrita en el puerto de salida R0. Las Fuentes de ésta versión se pueden ver a continuación.

```
const spec harp1 ={
        fpga_type= "Xilinx4000",
        clock_pad= "P13"
};
const dw = 8;
const ddw = 16;

main( target=harp1,
      port (in) datos = puerto1 : dw,
      port (out) R0 = puerto2 : ddw)
{
      int B0= -1; int B1= 1; int B2= 1; int B3= -1; int B4= 1;
      int B5= -1; int B6= 1; int B7= -1; int B8= 1;
      int A0,A1,A2,A3,A4,A5,A6,A7,A8:dw;
      int result:ddw;
      // Ingreso los valores de la matriz A(3x3)

      datos ? A0;  datos ? A1;  datos ? A2;
      datos ? A3;  datos ? A4;  datos ? A5;
      datos ? A6;  datos ? A7;  datos ? A8;
      /* otbencion del resultado utilizando una variable
      auxiliar  */
      result = (A0*B0) + (A1*B1) + (A2*B2) + (A3*B3) +
      (A4*B4) + (A5*B5) + (A6*B6) + (A7*B7) + (A8*B8);
      R0 ! result;
}
```

**Figura 5-55** Código del ejemplo 1, versión 4 utilizando solamente variables.

Resultados
Chip EPF10K20RC240-3

**Tabla 5-18** Resultado del ejemplo 1, versión 4 utilizando solamente variables.

| Utilizando | FF | Celdas Lógicas | Frec. Max [ MHz ] | Tclk Utilizados | Tiempo [ μs ] |
|---|---|---|---|---|---|
| **Variables** | 102 | 539 (47%) | 11.64 | 13 | 1.11 |

Claramente se puede observar la conveniencia de utilizar variables pues el cálculo se realiza en un solo período de reloj. Pero nuevamente destacamos la complejidad y poco practicidad de esta forma de programación.

## Versión 5

Aquí tomamos de un banco de memoria los 9 valores de la matriz y los colocamos en variables (para obtener una mejor performance) para realizar las cuentas y el resultado (de 16 bits) es guardado en memoria externa, como dos datos de 8 bits (en la parte baja R0_H y en la parte alta Ro_L). La matriz de elementos constantes es guardada en variables como para las versiones anteriores.

Aquí trabajamos con un bus de datos de 8 bits, los datos son leídos de las direcciones de la 0 a la 8 y el resultado, de 16 bits, es guardado en parte alta y baja ( R0_H y R0_L ), direcciones 9 y 10. También se utilizo una variable auxiliar para ir realizando las operaciones. El código de esta versión se puede apreciar a continuación.

```
const spec ArcPci = {
            fpga_type= "Xilinx4000",
            clock_pad= "P13"
            };
/* definicion de pines para bus de datos, direcciones, ce,
write, read */
const spec APmem={
     addr = {"mem_Addr<0>", "mem_Addr<1>",
              "mem_Addr<2>","mem_Addr<3>",
              "mem_Addr<4>","mem_Addr<5>"},
     data = "mem_Data<0>","mem_Data<1>",
            "mem_Data<2>", "mem_Data<3>",
            "mem_Data<4>","mem_Data<5>",
            "mem_Data<6>","mem_Data<7>"},
     ce   = "mem_ce",
     wb   = "mem_wb",
     en   = "mem_en"   };

const dw = 8;
const ddw = 16;
const tam_ram=2<<5;

void main (   target=ArcPci,
              eram mem[tam_ram]=APmem: dw){
        int R0          :ddw;
        int A0,A1,A2,A3,A4,A5,A6,A7,A8:dw;
        // cargo los valores de la matriz B cte
        int B0=-1; int B1=1; int B2=1; int B3=-1; int B4=1;
        int B5=-1; int B6=1; int B7=-1; int B8=1;

        / * ingreso desde memoria externa los valores
        de la matriz A */
        A0=mem[ 0 ];   //lectura de la RAM
        A1=mem[ 1 ];
        A2=mem[ 2 ];
        A3=mem[ 3 ];
        A4=mem[ 4 ];
        A5=mem[ 5 ];
        A6=mem[ 6 ];
        A7=mem[ 7 ];
        A8=mem[ 8 ];

        // calculo del promedio
        R0 = (A0*B0)+(A1*B1)+(A2*B2)+(A3*B3) +
             (A4*B4)+(A5*B5)+(A6*B6)+(A7*B7)+(A8*B8);
        // cargo en mem. externa el resultado del promedio
        mem[ 9 ]  = (R0\\dw);  // guardo bits mas signif.
        mem[ 10 ] = (R0<-dw); // guardo bits menos signif.
}
```

**Figura 5-56** Código del ejemplo 1, ver. 5 con memoria externa y utilizando variables para el cálculo.

También realizamos una pequeña variante utilizando un bus de datos ( y memoria externa) de 16 bits. Tomando los datos de entrada de 8 bits, como la parte menos significativa de cada word de memoria.

Resultados

Chip EPF10K20RC240-3

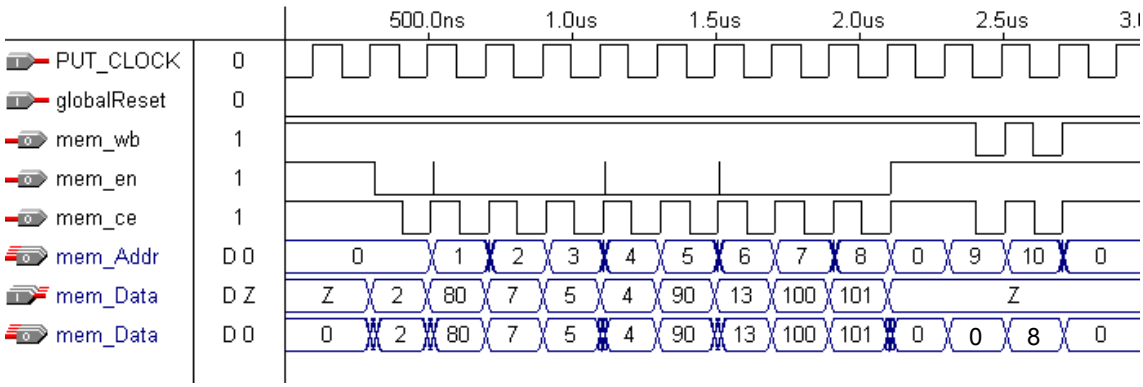



**Figura 5-57** Simulación del ejemplo 1, ver. 5 con memoria externa y utilizando variables para el cálculo.

**Tabla 5-19** Resultado del ejemplo 1, ver. 5 con memoria externa y utilizando variables para el cálculo.

| Utilizando | FF | Celdas Lógicas | Frec. Max [ MHz ] | Tclk Utilizados | Tiempo [ μs ] |
|---|---|---|---|---|---|
| **Memoria Externa 8 bits** | 103 | 556 (48%) | 11.19 | 14 | 1.25 |
| **Memoria Externa 16 bits** | 102 | 549 (48%) | 11.82 | 13 | 1.10 |

Para este ejemplo no se aprecian las ventajas de un bus más ancho, sobre todo pues solamente debemos escribir un dato de salida de 16 bits, para casos con mayor escritura en memoria la diferencia en el número de ciclos (Tclk) sería más notoria.

## Comparación de resultados para Ejemplo 1

Veamos primeramente una tabla con el resumen de las distintas versiones generadas, pero a diferencia de las anteriores descartamos los períodos de reloj necesarios para la inicialización y lectura de datos.

En todas estas pruebas se utilizó el mismo chip EPF10K20RC240-3 y la matriz constante B tuvo valores de 1 y –1.

**Tabla 5-20 Cuadro comparativo con resultados del ejemplo 1.**

| Ejemplo 1 Utilizando | FF | Celdas Lógicas | Frec. Max [ MHz ] | Tclk Utilizados | Tiempo [ ns ] |
|---|---|---|---|---|---|
| **RAM interna Versión 1** | 193 | 446 (39%) | 10.66 | 32 | 3.00 |
| **RAM interna Mat. B var Versión 2** | 161 | 462 (40%) | 10.74 | 21 | 1.96 |
| **3 RAM internas Versión 3** | 127 | 546 (47%) | 10.78 | 15 | 1.39 |
| **Variables Versión 4** | 102 | 539 (47%) | 11.64 | 13 | 1.11 |
| **RAM Externa 8 bits Versión 5** | 103 | 556 (48%) | 11.19 | 14 | 1.25 |

Observamos que la diferencia de FF entre la versión 1 y la versión 2 se debe a la forma en que se guardan las constantes en estos programas, en uno se utiliza RAM interna implementada con FF y en el otro se utilizan variables directamente. En este último caso Handel C minimiza los registros de éstas variables, para la RAM no lo puede hacer.

De la tabla también se puede apreciar que, como era de esperarse, la frecuencia máxima no varía significativamente pues el camino critico se encuentra en la realización de un producto y una suma. Lo que sí varía es el tiempo de cálculo que depende directamente del origen de los datos y de la posibilidad de paralelizar porciones del código. Siendo claramente mejor la solución que posee todos sus datos en variables pero como aspecto negativo es que su programación resulta más tediosa y menos parametrizable.

## Ejemplo 2

Aquí se realizo un proceso ( *void multiplico()* ) que realiza un producto y vierte el resultado en un acumulador. Se trabajaron todos los datos con memoria interna (tipo RAM), de esta forma se pudieron utilizar iteraciones para el cálculo de todas las multiplicaciones del promedio.

El código de esta versión, que llamaremos Procedimiento1, se puede apreciar a continuación.

```
const spec harp1 ={ fpga_type= "Xilinx4000",
                    clock_pad= "P13"   };
const dim =  3;
const dw  =  8; const ddw = 16;

main(     target=harp1,
          port (in) datos : dw,
          port (out) R0 : ddw)
{
          ram int A[(dim*dim)] : dw;
          ram int B[(dim*dim)] : dw;
          int result,salida:ddw;
          int multa,multb:dw;
          int i : 4;
    /* definicion de la expresion que calcula la multiplicion de dos elementos
    y lo vuelca al acumulador */
    void multiplico ( ){
          result=multa*multb;
          salida= salida + result; }
     // comienza el main
    B[0] = -1;B[1] = 1;B[2] = 1; B[3] = -1; B[4] = 1;
    B[5] = -1;B[6] = 1;B[7] = -1; B[8] = 1;

    //Ingreso los valores de la matriz A (3x3)
    while (i.<.9)
              { par { datos ? A[i]; i = i+1;} }
    // loop que calucula el promedio
    salida,i = 0,0;    // salida = 0 ; i=0;

    while (i.<.9) {
              par { multa,multb=A[i],B[i]; i = i+1; }
              multiplico( );
    }
    R0 ! salida;
}
```

**Figura 5-58** Código del ejemplo 2, versión 1 con memorias internas y un procedimiento para el cálculo del producto.

A continuación veremos unas variantes al código presentado en el cual experimentamos diferentes formas de reutilizar el hardware.

| **Procedimiento2** | **Expresión** |
|---|---|
| void multiplico ( ) {<br>salida= salida + multa*multb;<br>} | // en las declaraciones internas<br>int multiplico () = salida + multa*multb;<br>... // todo igual hasta el segundo WHILE<br>while (i.<.9) {<br>par{multa,multb=A[i],B[i];i=i+1;}<br>salida = multiplico();<br>} |

**Figura 5-59** Variaciones al código del ejemplo 2 versión 1

Resultados
Chip EPF10K20RC240-3

**Tabla 5-21** Resultados del ejemplo 2.

| Ejemplo 2 Utilizando | FF | Celdas Lógicas | Frec. Max. [ MHz ] | Tclk Utilizados | Tiempo [ μs ] |
|---|---|---|---|---|---|
| **RAM interna Procedimiento1** (Versión 1) | 223 | 606 (53%) | 11.29 | 48 | 4.25 |
| **RAM interna Procedimiento2** (Versión 2) | 206 | 584 (51%) | 10.61 | 39 | 3.67 |
| **RAM interna Expresión** (Versión 3) | 204 | 576 (50%) | 10.79 | 39 | 3.61 |

Vemos que es similar el comportamiento del circuito con el uso de procedimientos y con expresiones. Para el Procedimiento1, el mayor número de ciclos se debe a que el proceso multiplico() consta de 2 períodos de reloj a diferencia de los otros casos que es sólo uno.

En comparación con el Ejemplo 1, versión 1 (ver Tabla 5-20), donde no se reutiliza el hardware, vemos que los recursos utilizados por las versiones sintetizadas son similares. En principio se podría pensar que no se obtienen beneficios al utilizar procedimientos o expresiones pero lo que está ocurriendo es que una de las matrices es constante y tanto el Handel C como el M+PII optimizan bastante el circuito. En el siguiente punto veremos ejemplos donde se aprecian claras diferencias al no trabajar con constantes.

## 5.2.5 Operaciones con matrices: Multiplicación simple

Se implementó la multiplicación clásica entre dos matrices, dando como resultado una nueva matriz.

Las pruebas se realizaron con Handel C y fueron sintetizadas y simuladas para chips de Altera de la familia Flex10k**.** Estas consisten en el ingreso de los datos al chip (son guardados en un array o variable) y que éste realice las cuentas, el resultado lo guarda en variables, y saca el resultado fuera del chip. También se realizaron pruebas obteniendo los datos de una memoria externa y volcando los resultados nuevamente a esta memoria

## Ejemplo 1

Primero tratamos el caso de guardar los datos en memoria interna (tipo ram) al chip, de ancho 5 bits. Los resultados fueron guardados en una variable y sacados fuera del chip con las dimensiones adecuadas (10 bits), No se guardó toda la matriz resultado, sino que cada elemento antes de ser sacado del chip es pasado por una variable.

La obtención de los resultados, como se ve en la Figura 5-61, es uno cada 4 Tclk, ya que no podemos acceder en el mismo período de reloj a dos o más elementos de la RAM.

El código de esta versión se puede apreciar en la siguiente figura.

```
const spec harp1 = { fpga_type = "Xilinx4000",
                     clock_pad = "P160"};
const dim = 3;
const dw = 5;
const ddw = 10;

main( target=harp1,
      port (in) datos : dw,
      port (out) R0 : ddw) {
      /* defino memoria interna (tipo RAM) de 9 bytes y
      8 bit de ancho */
      ram int A[(dim*dim)] : dw;
      ram int B[(dim*dim)] : dw;
      int result:ddw;
      int inc:4;
      //Ingreso los valores de la matriz A (3x3)
       while (inc.<.9)
            {par { datos ? A[inc];inc=inc+1;}}
      //Ingreso los valores de la matriz B (3x3)
      inc=0;
      while (inc.<.9)
            {par { datos ? B[inc];inc=inc+1;}}
      /* calculo cada elemento de la matriz utilizando un
      acumulardor */
      result =(A[0]*B[0]);result = result + (A[1]*B[3]);
      result = result + (A[2]*B[6]);
      R0 ! result;      // escribo del pueto R0

      result =(A[0]*B[1]);   result = result + (A[1]*B[4]);
      result = result + (A[2]*B[7]);   R0 ! result;
      result =(A[0]*B[2]);   result = result + (A[1]*B[5]);
      result = result + (A[2]*B[8]);   R0 ! result;
      result =(A[3]*B[0]);   result = result + (A[4]*B[3]);
      result = result + (A[5]*B[6]);   R0 ! result;
      result =(A[3]*B[1]);   result = result + (A[4]*B[4]);
      result = result + (A[5]*B[7]);   R0 ! result;
      result =(A[3]*B[2]);   result = result + (A[4]*B[5]);
      result = result + (A[5]*B[8]);   R0 ! result;
      result =(A[6]*B[0]);   result = result + (A[7]*B[3]);
      result = result + (A[8]*B[6]);   R0 ! result;
      result =(A[6]*B[1]);   result = result + (A[7]*B[4]);
      result = result + (A[8]*B[7]);   R0 ! result;
      result =(A[6]*B[2]);   result = result + (A[7]*B[5]);
      result = result + (A[8]*B[8]);   R0 ! result;
}
```

**Figura 5-60** Código del ejemplo 1, multiplicación de dos matrices utilizando memoria interna.

Resultados:
Chip EPF10K50RC240-3

**Tabla 5-22** Resultado del ejemplo 1, multiplicación de dos matrices utilizando memoria interna

| | FF | Celdas Lógicas | Frec. Max [ MHz ] | Tclk Utilizados | Tiempo [ μs ] |
|---|---|---|---|---|---|
| **Ejemplo 1** (RAM interna) | 212 | 592 (21%) | 8.36 | 46 | 5.50 |

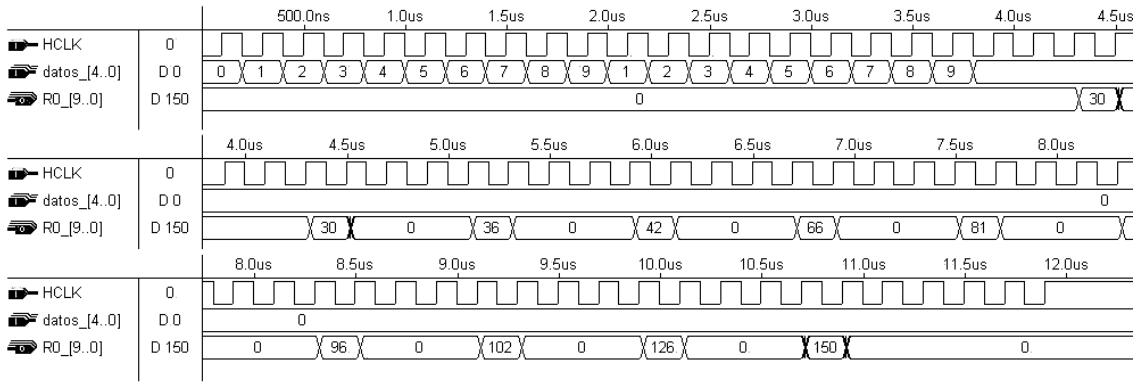



**Figura 5-61** Simulación del ejemplo 1, multiplicación de dos matrices utilizando memoria interna.

## Ejemplo 2

Ahora tratamos el caso de guardar los datos en variables (5 bits). Los resultados son colocados en variables intermedias y luego sacados fuera del chip con las dimensiones adecuadas (10 bits). Aquí como estamos trabajando con variables, se pueden realizar operaciones simultaneas y obtenemos los resultados uno por cada Tclk. No es posible realizar iteraciones en la obtención de datos ya que estamos trabajando con variables.

El código de esta versión es el siguiente:

```
const spec harp1 = {fpga_type = "Xilinx4000",
                    clock_pad = "P160"};

const dw = 5;
const ddw = 10;

main(     target=harp1,
          port (in) datos : dw,
          port (out) salida : ddw){

          int A0,A1,A2,A3,A4,A5,A6,A7,A8:dw;
          int B0,B1,B2,B3,B4,B5,B6,B7,B8:dw;
          int R0,R1,R2,R3,R4,R5,R6,R7,R8:ddw;

         // Ingreso los valores de la matriz A(3x3)
         datos ? A0;   datos ? A1;   datos ? A2;
         datos ? A3;   datos ? A4;   datos ? A5;
         datos ? A6;   datos ? A7;   datos ? A8;
         // Ingreso los valores de la matriz B (3x3)
         datos ? B0;   datos ? B1;   datos ? B2;
         datos ? B3;   datos ? B4;   datos ? B5;
         datos ? B6;   datos ? B7;   datos ? B8;
        /* paralelizo operaciones */
         par{
              // calculo de cada elemento de la matriz
              R0=(A0*B0)+(A1*B3)+(A2*B6);
              R1=(A0*B1)+(A1*B4)+(A2*B7);
              R2=(A0*B2)+(A1*B5)+(A2*B8);
              R3=(A3*B0)+(A4*B3)+(A5*B6);
              R4=(A3*B1)+(A4*B4)+(A5*B7);
              R5=(A3*B2)+(A4*B5)+(A5*B8);
              R6=(A6*B0)+(A7*B3)+(A8*B6);
              R7=(A6*B1)+(A7*B4)+(A8*B7);
              R8=(A6*B2)+(A7*B5)+(A8*B8);
         }
         //escritura del puerto R0
         salida ! R0;   salida ! R1;   salida ! R2;
         salida ! R3;   salida ! R4;   salida ! R5;
         salida ! R6;   salida ! R7;   salida ! R8;
}
```

**Figura 5-62** Código del ejemplo 2, multiplicación de dos matrices utilizando variables.

Resultados

Chip EPF10K50RC240-3

**Tabla 5-23** Resultados del ejemplo 2, multiplicación de dos matrices utilizando variables.

| | FF | Celdas Lógicas | Frec. Max [ MHz ] | Tclk Utilizados | Tiempo [ μs ] |
|---|---|---|---|---|---|
| **Ejemplo 2** (Variables) | 209 | 2317 (80%) | 12.83 | 29 | 2.26 |

## Ejemplo 3

Aquí se utilizaron variables para almacenar los datos y se utilizó un "procedimiento" (*void elemento_matriz( )*) para el cálculo de cada elemento de la matriz, sacando este mismo elemento en el procedimiento a través de una variable intermedia.

El código de esta versión es el siguiente:

```
… /* declaración de target y constantes igual al ejemplo anterior */
main( target=harp1,
        port (in) datos : dw,
        port (out) salida : ddw)
{
   int A0,A1,A2,A3,A4,A5,A6,A7,A8:dw;
   int B0,B1,B2,B3,B4,B5,B6,B7,B8:dw;
   int A_11,B_11,A_22,B_22,A_33,B_33:dw;
   int acum:ddw;
   // expresion que calcula cada elemento de la matriz
   void elemento_matriz(){
            acum = (A_11*B_11+A_22*B_22+A_33*B_33);
            salida ! acum;  //escirtura del puerto salida
   }
   //Ingreso los valores de la matriz A(3x3)
   datos ? A0;
   ... /* el resto de los datos se ingresan igual que antes */
   datos ? B8;

   // Calculo de los elementos de la matriz
   A_11,B_11,A_22,B_22,A_33,B_33=A0,B0,A1,B3,A2,B6;
   par{
      elemento_matriz();   // invoco el procedimiento
      B_11,B_22,B_33=B1,B4,B7; /* en paralelo cargo
                                  nuevos valores */
   }
   par{
      elemento_matriz();
      B_11,B_22,B_33=B2,B5,B8;
   }
   par{
      elemento_matriz();
      A_11,B_11,A_22,B_22,A_33,B_33=A3,B0,A4,B3,A5,B6;
   }
   par{
      elemento_matriz();
      B_11,B_22,B_33=B1,B4,B7;
   }
   par{
      elemento_matriz();
      B_11,B_22,B_33=B2,B5,B8;
   }
   par{
      elemento_matriz();
      A_11,B_11,A_22,B_22,A_33,B_33=A6,B0,A7,B3,A8,B6;
   }
   par{
      elemento_matriz();
      B_11,B_22,B_33=B1,B4,B7;
   }
   par{
      elemento_matriz();
      B_11,B_22,B_33=B2,B5,B8;
   }
   elemento_matriz();
}
```

**Figura 5-63** Código del ejemplo 3, multiplicación de dos matrices utilizando variables y un procedimiento.

Resultados

Chip EPF10K50RC240-3

**Tabla 5-24** Resultado del ejemplo 3, multiplicación de dos matrices utilizando variables y un procedimiento.

| | FF | Celdas Lógicas | Frec. Max [ MHz ] | Tclk Utilizados | Tiempo [ μs ] |
|---|---|---|---|---|---|
| **Ejemplo 3** (Procedimiento y Variables) | 161 | 513 (18%) | 13.98 | 38 | 2.71 |

## Cuadro comparativo de los resultados

Veamos primeramente una tabla con el resumen de los distintos ejemplos generados, pero a diferencia de las anteriores descartamos los períodos de reloj necesarios para la inicialización y lectura de datos.

**Tabla 5-25** Cuadro comparativo de los resultados del producto de matrices.

| | FF | Celdas Lógicas | Frec. Max [ Mhz ] | Tck Utilizados | Tiempo [ μs ] |
|---|---|---|---|---|---|
| **Ejemplo 1** (RAM Interna) | 212 | 592 (21%) | 8.36 | 46 | 5.50 |
| **Ejemplo 2** (Variables) | 209 | 2317 (80%) | 12.83 | 29 | 2.26 |
| **Ejemplo 3** (Procedimiento y Variables) | 161 | 513 (18%) | 13.98 | 38 | 2.71 |

La diferencia que se observa en los dos primeros ejemplos se debe a que el M+P II optimiza y reutiliza partes del hardware generado, no lo hace el HCC pues él genera un circuito similar al del ejemplo 2. En el ejemplo 3 la reutilización del hardware fue realizada en la programación del ejemplo.

Si bien la programación usando datos en variables es más tediosa (ejemplo 2) se obtienen resultados muy buenos en tiempo de cálculo porque se paralelizan todas las operaciones, la mayor demora está en sacar los datos por el puerto de salida.

# 6 Aplicación final

## 6.1 Introducción

La idea de la aplicación final fue tomar un algoritmo escrito enteramente en lenguaje C y pasarlo a hardware utilizando alguno de los compiladores evaluados de la forma más directa posible.

La aplicación final elegida para implementar es un filtro de imágenes en tonos de grises. Nos basamos en un diseño realizado enteramente en lenguaje C realizado en el Instituto de Ingeniería Eléctrica, de aquí en más lo referiremos como *Filtro Original* y sus fuentes se pueden ver en Apéndice F pág. 249.

El compilador elegido fue Handel C pues resulta más cómodo el trabajo con memorias y además tiene operador producto implementado facilitando la programación. Otra razón importante es que Transmogrifier C esta discontinuado y Handel C en cambio sigue en desarrollo por la empresa Celoxica.

La plataforma en la cual se va a probar el diseño es la placa ARC-PCI de Altera (ver Apéndice E), donada por esta empresa al IIE.

Se realizarán dos programas, uno en Handel C que se compilará a hardware y grabará en la placa ARC-PCI y realiza el filtrado de la imagen obteniendo los datos de sus bancos de memoria. El otro programa que tenemos que generar es un programa clásico que corre en el PC y se encarga de la lectura y escritura de la imagen y el filtro a los archivos en el disco, también se encarga de transferir estos datos hacia y desde los bancos de memoria de la placa.

La aplicación debe realizar la convolución discreta de una imagen y el filtro ambos representados como matrices. Esto es, dado un punto de la imagen tomamos una submatriz centrada en él de igual dimensión que la matriz del filtro. Luego realizamos el producto término a término con la matriz del filtro, y el nuevo punto lo obtenemos haciendo la sumatoria de estos productos dividida entre la suma de los coeficientes del filtro. Esta operación no se puede realizar sobre los puntos de los bordes de la imagen y por tanto no son tenidos en cuenta para la nueva imagen.

## Filtro Original

Esta versión del filtro, lee la imagen de un archivo en formato PGM [7] y también lee los coeficientes del filtro desde un archivo en ASCII. Devuelve la imagen filtrada en otro archivo también en formato PGM.

Veamos un pseudocódigo de dicho programa:

```
Cargar imagen desde archivo a una matriz llamada imagen_entrada
Cargar coeficientes del filtro desde archivo a una matriz llamada mat
Sea FFtot = número total de filas del filtro.
Sea FCtot = número total de columnas del filtro.
Para cada fila de la imagen ( = filaImag) {
    Para cada columna de la imagen ( = colImag) {
        Sea Suma = 0
        Para cada fila del filtro ( = filaFiltro) {
            Para cada columna del filtro ( = colFiltro) {
                % Calculo coord. del punto de la imagen (píxel)
                Fila = filaImag – ( FFtot/2 – filaFiltro)
                Col = colImag – (FCtot/2 – colFiltro)
                Suma = Suma + imagen_entrada(fila,col) * mat(filaFiltro, colFiltro)
            }
        }
        imagen_salida(filaImag,colImag) = Suma / suma coef. del filtro
    }
}
Guardar imagen filtrada (imagen_salida) en archivo
```

**Figura 6-1** Pseudocódigo del filtro de imagen.

## Esquema de la placa ARC-PCI

Como podemos observar en la Figura 6-2 la placa consta principalmente de dos bancos de RAM de 32 bits de ancho (RAM1 y RAM2), dos bancos de cache, tres FPGA tipo FLEX10K50 (PLD0, PLD1 y PLD2) y dos buses internos (BUS1 y BUS2). Dicha placa se conecta directamente al bus PCI de un PC.

En el PLD0 se graba la interfaz con el bus PCI que fue diseñada utilizando un IP core de Altera. Dicha interfaz permite definir tres direcciones base que se ven como direcciones de memoria del PC, éstas son:

- physical_address_base0 : donde se encuentran los registros de control de la interfaz
- physical_address_base1 : se mapea el banco RAM1
- physical_address_base2 : se mapea el banco RAM2

Entre las facilidades que ofrece la interfaz destacamos la posibilidad de realizar lecturas y escrituras en modo burst o simples, realizar transferencias directamente a la RAM del PC utilizando DMA (trabajando en modo master) y grabar los chips PLD1 y PLD2 desde el bus PCI (no desde ByteBlaster[3]).

[3] Es un interfaz de hardware conectado al puerto paralelo estándar del PC que permite descargar datos desde el software M+PII o Quartus en modo PS (passive serial) o JTAG (Join Test Actino Group)

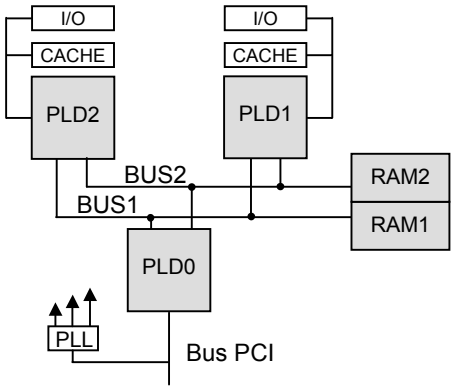



**Figura 6-2** Esquema de conexionado de la placa ARC-PCI.

También el PLD0 oficia de árbitro en los buses internos BUS1 y BUS2, controlando el acceso realizado por los otros PLD y por él. El protocolo manejado por el árbitro consta de una señal de entrada (una por cada PLD y por cada bus), activa por cero, que es utilizada para pedir el bus y debe ser mantenida activa mientras se lo este utilizando. Además se cuenta con otra señal de salida (una por cada PLD y por cada bus), activa por cero, que es utilizada para indicarle al dispositivo que tiene el control del bus. Esta señal permanecerá activa mientras no se cancele el pedido.

### Características del diseño en Handel C a realizar:

Requisitos del diseño:

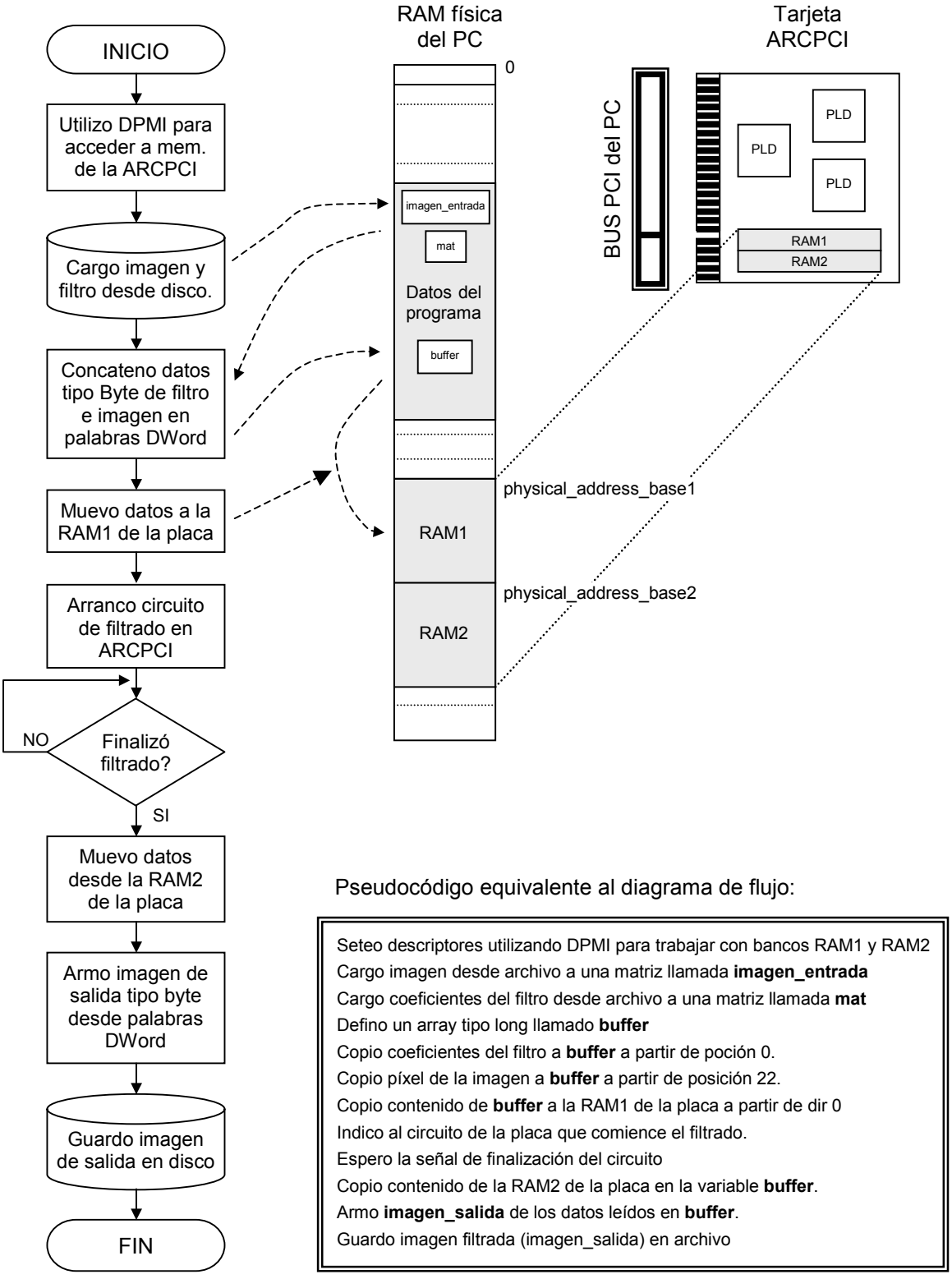
- Debe funcionar en la placa ARC-PCI, en uno sólo de sus PLD, y por tanto podrá utilizar alguna de las velocidades de reloj que están disponibles, ellas son 16.5, 33 o 66 MHz.
- Debe trabajar en un sistema con buses donde comparten el uso de los bancos de memoria los tres PLD.
- La imagen y el filtro deben ser cargados desde los bancos RAM1 y/o RAM2 donde también se guardará la imagen filtrada.
- Debe fácilmente ser actualizable a cualquier tamaño de imagen y de filtro.
- El arranque del circuito se realiza desde el PC.
- El diseño deberá indicar al PC que terminó el filtrado.

Opciones de diseño:

- Tanto la imagen como los coeficientes del filtro se leerán solamente del banco RAM1.
- Se trabajará con imágenes en tono de grises de 8 bits no mayores a 724x724 (restricción impuesta por el banco de memoria de 128Kx32bits) y además el número de columnas deberá ser múltiplo de 4.
- La imagen de salida será escrita en el banco RAM2.
- La actualización del tamaño del la imagen y el filtro se realizará en tiempo de compilación.

## 6.2 Código C del programa compilado a software

Teniendo presente las opciones de diseño, comentaremos inicialmente el programa en C realizado para correr en el PC, que para su desarrollo nos basamos en parte del código del Filtro Original. Veamos su diagrama de flujo y un esquema del sistema completo.




Pseudocódigo equivalente al diagrama de flujo:

```
Seteo descriptores utilizando DPMI para trabajar con bancos RAM1 y RAM2
Cargo imagen desde archivo a una matriz llamada imagen_entrada
Cargo coeficientes del filtro desde archivo a una matriz llamada mat
Defino un array tipo long llamado buffer
Copio coeficientes del filtro a buffer a partir de poción 0.
Copio píxel de la imagen a buffer a partir de posición 22.
Copio contenido de buffer a la RAM1 de la placa a partir de dir 0
Indico al circuito de la placa que comience el filtrado.
Espero la señal de finalización del circuito
Copio contenido de la RAM2 de la placa en la variable buffer.
Armo imagen_salida de los datos leídos en buffer.
Guardo imagen filtrada (imagen_salida) en archivo
```

**Figura 6-3** Diagrama de flujo del programa que lanza el filtrado de imagen.

El código completo de este programa se puede ver en el Apéndice F pág. 252, aquí solamente mencionaremos algunas partes que nos parecen relevantes.

La placa ARC-PCI sólo realiza lecturas y escrituras de palabras dword alineadas por tanto el mayor cuidado está en como armamos los datos (de ancho 32 bits), que van a ser transferidos a sus bancos de RAM, a partir de los coeficientes del filtro y píxel de imagen que son de ancho 8 bits.
En el programa definimos un array, llamado *buffer*, tipo long (o sea 32 bits) en el cual vamos a ir armando las palabras dword a partir de los datos tipo byte. Veamos por ejemplo como fue resuelta la copia de la imagen de entrada hacia *buffer*.

```
n = 88; // posición inicial de la imagen en el buffer
// ancho representa el número total de columnas de la imagen
// alto representa el número total de filas de la imagen
 for(pixel=0; pixel<(ancho*alto); ++pixel)
  {
    x = pixel % ancho;
    y = pixel / ancho;
    buffer[n/4] = buffer[n/4] | (imagen_entrada[y][x] << 8*(n % 4));
    n=n+1;
  }
```

**Figura 6-4** Código en C de empaquetado de imagen para RAM de placa ARC-PCI.

Luego que los datos están correctamente cargados en el array *buffer* se copian a la RAM de la placa utilizando una función de bajo nivel que mueve bloques de memoria (llamada movedatal()).
Veamos a continuación como se desarma del array *buffer* la imagen de salida.

```
n = 0; // posición inicial de la imagen en el buffer
// ancho representa el número total de columnas de la imagen
// alto representa el número total de filas de la imagen
 for(pixel=0; pixel<(ancho*alto); ++pixel)
  {
    x = pixel % ancho;
    y = pixel / ancho;
    imagen_salida[y][x]=  (buffer[n/4]  >> 8*(n % 4)) & 0xFF;
    n=n+1;
  }
```

**Figura 6-5** Código en C de desempaquetado de imagen desde RAM de placa ARC-PCI.

Las funciones de lectura de coeficientes del filtro y lectura y grabación de imagen en formato PGM se mantuvieron iguales que en su versión original.

Subrayamos que mientras la placa realiza el filtrado el procesador del PC puede trabajar en otras tareas y periódicamente verificar si la placa finalizó. Pueden utilizarse otros mecanismos donde, por ejemplo, el circuito en la placa genere una interrupción al finalizar.

## 6.3 Código C del programa compilado a hardware

Teniendo en cuenta los requerimientos comenzamos realizando una primera aproximación al diseño final que nos servirá para ir verificando un punto clave que es la frecuencia máxima de trabajo. Luego de ser necesario iremos refinando el código para obtener una mejor performance.

Realizamos un diseño paramétrico en tamaño de imagen y tamaño de filtro en tiempo de compilación, que consta esencialmente de los siguientes bloques:

```
while(1){
        Espero señal de disparo
        Configuro coeficientes del filtro
        Filtrado de imagen
        Doy señal de finalización
}
```

**Figura 6-6** Esquema del programa principal para HandelC de la aplicación final.

Como mencionamos en las restricciones, nuestro diseño comparte bancos de memoria con otros dispositivos en un bus. Por esta razón en Handel C debemos pedir y esperar que nos otorguen el bus antes de realizar cualquier operación sobre las RAM. A modo de ejemplo el código necesario para llevar a cabo este control para el banco 1 es:

| | |
|---|---|
| ...<br>main ( ..... )<br>{<br>  par{<br>  while(1) {<br>  // saco por el puerto la variable pedido1<br>  pido_bus1 ! pedido1;<br>  };<br>  while(1){<br>  // resto del programa principal<br>  ....<br>  };<br>  };<br>}; | Programa Principal |
| ...<br>pedido1 = 0; // pido el bus 1<br>recibo_bus1 ? aux; // leo el Puerto de resp. del árbitro<br>while (aux) { recibo_bus1 ? aux; }; // espero que aux=0<br>// ya tengo el bus, continuo con las operaciones<br>...<br>pedido1 = 1; // devuelvo bus 1<br>... | En alguna parte del código donde quiera trabajar con el bus 1 |

**Figura 6-7** Ejemplo de pedido de bus para Handel C en aplicación final.

Hacemos notar que si bien en el ejemplo anterior el pedido del bus se realiza con una señal que es activa por cero (al igual que lo maneja el árbitro), en la realización de la aplicación final utilizamos una señal que es activa por uno y luego la negamos antes de interconectarla con el árbitro. La razón se halla en que el pedido realizado por nuestro diseño es salida de un flip-flop que en el momento del reset es llevado a cero, por lo tanto estaríamos pidiendo el bus por algunos períodos de reloj (1 o 2) antes de que pudiéramos desde el programa en HCC cambiar su valor, este comportamiento lo consideramos como un efecto secundario no deseado que resolvimos fácilmente trabajando con la señal negada.

A continuación describiremos cada uno de los bloques mencionados en la Figura 6-6.

## Configuración del filtro

Los coeficientes del filtro, enteros con signo de 8 bits, se leen de la memoria, en particular del banco 1, a partir de la dirección 0 y hasta como máximo la dirección 20 pues fijamos 9x9 como tamaño máximo razonable para el filtro. Los coeficientes deben aparecer listados por filas comenzando por la fila 1 columna 1. Estos coeficientes son guardados en la memoria interna del chip (utilizando la megafunción lpm_ram_dq según se vio en el capítulo 5). La siguiente figura muestra el código para el HCC que se encarga de cargar estos coeficientes.

```
void configurar_filtro(){
// Los elementos del filtro estan ordenados
// por filas en la ram

    l, suma_coef, pedido1=0, 0, 1; // pido bus 1
    // espero me den el bus 1
    recibo_bus1 ? aux;
    while(aux){recibo_bus1 ? aux;};

    // escribo palabra que indica al PC "en proceso"
    mem_dat[0x3ffff]=0xcccccccc;

    while(l.<.Felementos){
       // para todos los elementos del filtro (Felementos)
       if (l.(0..1)==0) // leo de la RAM cada 4 elementos
             var32=mem_dat[0 @ l.(2..(ind_ram-1))];
       // extraigo de var32 a tmp2 el byte correspondiente
       // al elemento "l"
            case ((l).(0..1)){
                  0: {tmp2 = var32.(0..7);}
                  1: {tmp2 = var32.(8..15);}
                  2: {tmp2 = var32.(16..23);}
                  3: {tmp2 = var32.(24..31);}
            };
            par{
                  // escribo en RAM interna el coeficiente leido
                  ram_coef_din ! tmp2; // saco dato
                  ram_coef_we ! 1;  // doy pulso de escritura
                  // calculo suma de coeficientes
                  suma_coef = suma_coef + tmp2;
                  l = l+1;
            }
       };
       pedido1= 0; // devuelvo bus 1
}
```

**Figura 6-8** Código para Handel C de lectura de coeficientes de filtro.

Veamos un ejemplo de cómo se deben ordenar en la memoria de la placa los coeficientes del filtro, recordemos que ésta RAM tiene un ancho de 32 bits.

| Filtro | | |
|---|---|---|
| 1 | 2 | 3 |
| 4 | 5 | 6 |
| 7 | 8 | 9 |

| Dir. RAM | Contenido RAM |
|---|---|
| 0 | 04030201h |
| 1 | 08070605h |
| 2 | 00000009h |

## Filtrado de imagen

La estructura general de esta rutina es la vista en el pseudocódigo del Filtro Original, pero como no trabajamos con matrices sino con memorias debemos realizar cálculos que nos permitan determinar la dirección en la RAM del píxel deseado. El nuevo pseudocódigo entonces es:

```
Para cada fila de la imagen ( = j) {
        Para cada columna de la imagen ( = i) {
                Sea Suma = 0
                Pido el bus 1
                Para cada fila del filtro ( = k) {
                    Para cada columna del filtro ( = m) {
                        // Calculo direccion del punto de la imagen (píxel)
                        dir_ram = ...  // ver fórmula mas adelante.
                        Si no tengo el dato de la RAM leido entonces {
                            Esperar que me den el bus 1
                            var32 = contenido de la RAM en dir_ram
                        }
                        tmp2 =  un byte de var32 correspondiente al píxel(j+k, i+m)
                        tmp = coeficiente del filtro para (k,m)
                        Suma = Suma + tmp * tmp2
                    }
                }
                Calculo el cociente de Suma / suma coef. del filtro
                Devuelvo bus 1
                Armo dword llamado tmp32 que contiene en posición correcta el nuevo pixel
                Si termine una fila de la imagen o si complete un dword entonces {
                        Pido bus 2
                        Espero me den el bus 2
                        Escribo en RAM el dato tmp32
                        Devuelvo bus 2
                }
        }
}
Pedir bus 1
Esperar que me den bus 1
Escribir en 0x3ffff palabra de "Fin de filtrado"
Devuelvo bus 1
```

**Figura 6-9** Pseudocódigo del filtro de imagen para Handel C.

Es importante destacar que la opción de diseño que restringe el número de columnas totales de la imagen a múltiplos de 4, facilita el cálculo de la dirección y posición del píxel en la RAM y además de esta forma el comienzo de las filas de la imagen está alineado en doubleword.

La dirección de la RAM para la lectura de un píxel la calculamos como:

$$\text{dir\_ram} = \text{dir\_base} + \left[\left(\text{filaImag} + \text{filaFiltro} - \left(\text{FFtot} >> 1\right)\right) .* .\text{ICtot} + \right.$$
$$\left. + \text{colImag} + \text{colFiltro} - \left(\text{FCtot} >> 1\right) \right] >> 2$$

Donde:

dir_base = offset donde comienza la Imagen en memoria, en nuestro caso es 22
filaImag = fila de la Imagen que se esta analizando (sería la "j" de Figura 6-9)
colImag = ídem pero para columnas (sería la "i" de Figura 6-9)
IFtot = total de filas de la Imagen
ICtot = ídem pero para columnas
filaFiltro = fila del Filtro que se esta analizando ( sería la "k" de Figura 6-9)
colFiltro = ídem pero para columnas (sería la "m" de Figura 6-9).
FFtot = total de filas del Filtro
FCtot = ídem pero para columnas

El dato de la RAM de 32 bits es cargado en *var32* y luego determinamos el píxel de la siguiente forma:

```
case (pos) {
        0: { pixel = var32.(0..7);}
        1: { pixel = var32.(8..15);}
        2: { pixel = var32.(16..23);}
        3: { pixel = var32.(24..31);}
};
```

**Figura 6-10** Ejemplo de extracción de un byte de un palabra dword para Handel C.

Donde la posición del byte correspondiente al píxel deseado se determina como:

$$pos = (colImag + colFiltro - (FCtot >> 1)).(0..1); \quad // \textit{resto de dividir entre 4}$$

Otro aspecto que consideramos para el diseño fue tratar de minimizar los accesos a la RAM, para ello decidimos mantener un cache de los dword leídos, pero en esta primer versión el tamaño de ese cache es de 1. Si no utilizáramos este cache, por ejemplo en un filtro de 3x3, deberíamos acceder a la RAM nueve veces. Al utilizarlo este número se reduce a 3 para algunos puntos y para otros a 6 veces.

Como se pudo observar en el pseudocódigo de la Figura 6-9, el acceso al bus 1 se pide cuando comenzamos a calcular cada punto de la nueva imagen y se devuelve antes de realizar la división entre la suma de los coeficientes del filtro. La escritura del nuevo píxel se realiza en el bus 2 y aquí realizamos escrituras cada 4 pixels, en los bordes dependerá del tamaño del filtro, por ejemplo para 3x3 se escribirá luego de realizar tres cálculos de pixels nuevos.

### Divisor de 16 bits

Fue necesario crear una función adicional que realizara la división entera de 16 bits con signo, si bien ya habíamos analizado un algoritmo en el Capítulo 5 éste no cumplía con los requisitos de velocidad, por lo tanto implementamos el mismo método pero con otro algoritmo que trabajara a mayor velocidad. Su código es el siguiente:

```
// realiza la division entera de sum16 entre sum_coef
// el resultado div8 es recortado a 8 bits
void division8(){
  int ccnt0,ccnt1,ccnt2,ccnt3,ccnt4,ccnt5,ccnt6,ccnt7:1;
  int ccnt8,ccnt9,ccnt10,ccnt11,ccnt12,ccnt13:1;
  int d,z : 24;
  int s3,s4,s5,s6,s7:24;
  int s8,s9,s10,s11,s12,s13,s14,s15:24;

  if (suma_coef.(1..7) .>. 0){
    // divido si suma_coef es > 1
    // calculo valores absolutos y ajusto ancho de sum16
    // tambiendesplazo suma_coef 16 posiciones a la izq.
    z, d = 0 @ abs(sum16), abs(suma_coef) @ 0;
    ccnt13 =((((z<<3)-d)<0) ? 0 : 1);
    s3=(z<<3)-((ccnt13==1) ? d : 0);
    ccnt12 =((((s3<<1)-d)<0) ? 0 : 1);
    s4=(s3<<1)-((ccnt12==1) ? d : 0);
    ccnt11 =((((s4<<1)-d)<0) ? 0 : 1);
    s5=(s4<<1)-((ccnt11==1) ? d : 0);
    ccnt10 =((((s5<<1)-d)<0) ? 0 : 1);
    s6=(s5<<1)-((ccnt10==1) ? d : 0);
    ccnt9 =((((s6<<1)-d)<0) ? 0 : 1);
    s7=(s6<<1)-((ccnt9==1) ? d : 0);
    ccnt8 =((((s7<<1)-d)<0) ? 0 : 1);
    s8=(s7<<1)-((ccnt8==1) ? d : 0);
    ccnt7 =((((s8<<1)-d)<0) ? 0 : 1);
    s9=(s8<<1)-((ccnt7 ==1) ? d : 0);
    ccnt6 =((((s9<<1)-d)<0) ? 0 : 1);
    s10=(s9<<1)-((ccnt6==1) ? d : 0);
    ccnt5 =((((s10<<1)-d)<0) ? 0 : 1);
    s11=(s10<<1)-((ccnt5==1) ? d : 0);
    ccnt4 =((((s11<<1)-d)<0) ? 0 : 1);
    s12=(s11<<1)-((ccnt4==1) ? d : 0);
    ccnt3 =((((s12<<1)-d)<0) ? 0 : 1);
    s13=(s12<<1)-((ccnt3==1) ? d : 0);
    ccnt2 =((((s13<<1)-d)<0) ? 0 : 1);
    s14=(s13<<1)-((ccnt2==1) ? d : 0);
    ccnt1 =((((s14<<1)-d)<0) ? 0 : 1);
    s15=(s14<<1)-((ccnt1==1) ? d : 0);
    ccnt0 =((((s15<<1)-d)<0) ? 0 : 1);

    // armo resultado y ajusto el signo
    div8= (((sum16.15 ^ suma_coef.7)==0) ? (ccnt7 @
ccnt6 @ ccnt5 @ ccnt4 @ ccnt3 @ ccnt2 @ ccnt1 @
ccnt0) : -(ccnt7 @ ccnt6 @ ccnt5 @ ccnt4 @ ccnt3 @ ccnt2
@ ccnt1 @ ccnt0));
  }
  else div8= sum16.(0..7); // si no divido solo recorto dato
  }
```

**Figura 6-11** Código de un divisor de 16 bits para Handel C.

El cuerpo principal es la división de un número de 16 bits entre otro de 8 bits, ambos sin signo, luego de realizado el cálculo se ajusta el signo del resultado.

Para el uso en el algoritmo de filtrado de imágenes recortamos el resultado a los 8 bits menos significativos al igual que se realiza en el código C de la versión Filtro Original.

Destacamos que en el caso de divisores que sean 0 o 1 no se realiza la división y el resultado es igual al dividendo (recortado a 8 bits).

Resultados
Chip EPF10K50RC240-3 (ALTERA)

| | FF | Celdas Lógicas | Frec. Max. [ MHz ] | Tclk |
|---|---|---|---|---|
| **division8()** | 284 | 604 | 27.85 | 1 ó 29 |

**Señal de finalización**

Mientras el circuito de la placa ARC-PCI está realizando el filtrado, el programa en C que corre en el PC puede ir realizando cualquier otro tipo de tareas. Para esta aplicación la única tarea que realiza este programa es esperar y verificar la finalización del filtrado.

Del conjunto de opciones que disponemos con la interfaz PCI optamos por la utilización de un flag (o bandera) en una posición determinada de la memoria de la placa, en particular RAM1. Por lo tanto desde el circuito indicamos “finalizado” con la escritura del dword 0xFFFFFFFF en la dirección 0x3FFFF. También indicamos “en proceso” escribiendo en la misma dirección el dword 0xCCCCCCCC.

## 6.4 Modificaciones a la primer versión

Cuando compilamos esta primer versión encontramos que la velocidad máxima no era suficiente, llegábamos tan solo a 10 MHz para imágenes de 12x12. Luego de examinar el camino critico observamos que el problema se encontraba en el producto del coeficiente del filtro por el píxel.

Para aumentar la velocidad del circuito sustituimos el operador producto (*) por un procedimiento que realiza la multiplicación en 4 etapas que opera a 23.5 MHz.

**Multiplicador de 8 bits**

El método que implementamos es el clásico de corrimientos y sumas pero modificado para que demore pocos ciclos de reloj. La primer etapa del multiplicador calcula el valor absoluto del multiplicando (coeficiente). En las siguientes dos etapas desplazamos y sumamos y en la última hacemos una suma y calculamos el valor ya con signo del resultado. A continuación se pueden ver las fuentes de dicho multiplicador y un esquema que muestra gráficamente las operaciones realizadas.

```
// realiza la operacion sum16 = tmp * tmp2 + sum16
// con tmp y tmp2 de 8 bits,sum16 de 16 bits
// el producto es con signo
void multip8(){
  int m41, m42: 14;
  int m0, m1, m2, m3: 10;
  int mr:8;
  par{   //calculo valor absoluto de tmp
     if (tmp.7==1)   mr= (~tmp)+1;
     else            mr= tmp;
  }
  par{
    // realizamos el producto parcial de tmp2 con grupos de a dos bits de mr
    m0=cond(mr.(0..1), 0 -> 0, 1 -> 0 @ tmp2, 2 -> (0 @ tmp2 @ cero1),
                       3 -> ((cero2 @ tmp2.(1..7))+ (cero1 @ tmp2.(0..7))) @ tmp2.0);
    m1=cond(mr.(2..3), 0 -> 0, 1 -> 0 @ tmp2, 2 -> (0 @ tmp2 @ cero1),
                       3 -> ((cero2 @ tmp2.(1..7))+ (cero1 @ tmp2.(0..7))) @ tmp2.0);
    m2=cond(mr.(4..5), 0 -> 0, 1 -> 0 @ tmp2, 2 -> (0 @ tmp2 @ cero1),
                       3 -> ((cero2 @ tmp2.(1..7))+ (cero1 @ tmp2.(0..7))) @ tmp2.0);
    m3=cond(mr.(6..7), 0 -> 0, 1 -> 0 @ tmp2, 2 -> (0 @ tmp2 @ cero1),
                       3 -> ((cero2 @ tmp2.(1..7))+ (cero1 @ tmp2.(0..7))) @ tmp2.0);
  }
  par{
    // realizamos suma parcial de los resultados temporales
    // estas sumas estan realizadas convenientemente para que sean de la menor cantidad de bits posibles
    m41 = ((0 @ m0.(4..9))+ m2.(0..9)) @ m0.(0..3);
    m42 = ((0 @ m1.(4..9))+ m3.(0..9)) @ m1.(0..3);
  }
  // completamos el calculo sumando los resultados anteriores y ajustando el signo del resultado
  mul16=((tmp.7==1) ? -(((0 @ m41.(2..13))+m42.(0..13)) @ m41.(0..1)) : ((0 @ m41.(2..13))+m42.(0..13)) @ m41.(0..1));
  sum16 = mul16 + sum16;
}
```

**Figura 6-12** Código de un multiplicador de 8 bits para Handel C.

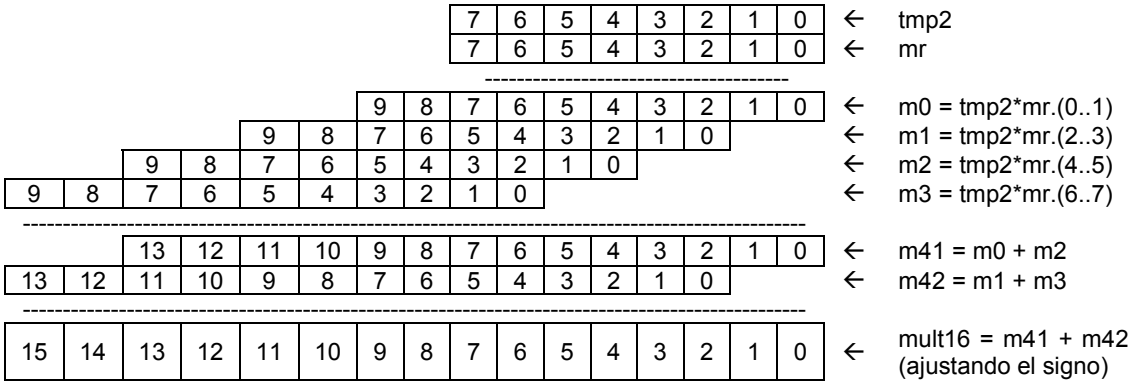



**Figura 6-13** Diagrama del cálculo realizado por el multiplicador.

Resultados

Chip EPF10K50RC240-3 (ALTERA)

| | FF | Celdas Lógicas | Frec. Max. [ MHz ] | Tclk |
|---|---|---|---|---|
| **multip16()** | 111 | 274 | 23.47 | 4 |

Con estos cambios obtuvimos, para el circuito completo del filtrado, una velocidad de 23.5 MHz para una imagen de 12x12 y un filtro de 3x3. Ahora el camino crítico está en la suma de 16 bits que se realiza al final de este procedimiento, donde se acumula el producto calculado.

Por otro lado, al crecer el tamaño de la imagen aumenta también el ancho de las variables relacionadas con el cálculo de la dirección del píxel. El camino crítico queda entonces en las operaciones de éste cálculo. Como siguiente paso decidimos optimizar el cálculo partiéndolo convenientemente en 4 etapas, dicho cambio se puede observar en la figura que está a continuación donde resaltamos las líneas afectadas.

**Extracto del código original**

```
...
j, primera, i= Ffilas_div2, 1, Fcolumnas_div2;
   while (j .<. (Ifilas-Ffilas_div2)){
      while (i .<. (Icolumnas-Fcolumnas_div2)){
         par{
            pedido1, l, dir_ram_ant, sum16, k =0, 0, 0, 0, -Ffilas_div2;
            flag = ((i.(0..1)==3 | (i==(Icolumnas-Fcolumnas_div2-1))) ? 0 : 1);
         }
         while(k!=(Ffilas_div2+1)){
            m = -Fcolumnas_div2;
            while(m!=(Fcolumnas_div2+1)){
               dir_ram_ant =  dir_ram;
               dir_ram = Idir_base + 0 @ ((j + k).*.IcolW).(2..(ind_ram_ext-1)) + (0 @ ((i + m).(2..(tam_ind_Icol-1))));
               if ((dir_ram_ant!=dir_ram) | (primera==1)) {
                  recibo_bus1 ? aux;
                  while(aux){recibo_bus1 ? aux;};
                  var32, primera=mem_dat[0 @ dir_ram],0;
               };
               ...
```

**Extracto del código en su versión final**

```
...
j, primera, i= Ffilas_div2, 1, Fcolumnas_div2;
   while (j .<. (Ifilas-Ffilas_div2)){
      while (i .<. (Icolumnas-Fcolumnas_div2)){
         par{
            pedido1, l, dir_ram_ant, sum16, k =0, 0, 0, 0, -Ffilas_div2;
            flag = ((i.(0..1)==3 | (i==(Icolumnas-Fcolumnas_div2-1))) ? 0 : 1);
         }
         while(k!=(Ffilas_div2+1)){
            tmp3 = 0 @ (j + k);
            tmp1, m= (tmp3.*.(IcolW_div4)).(0..ind_ram_ext), -Fcolumnas_div2;
            while(m!=(Fcolumnas_div2+1)){
               dir_ram, dir_ram_ant=tmp1 + (0 @ ((i + m).(2..(tam_ind_Icol-1)))), dir_ram;
               if ((dir_ram_ant!=dir_ram) | (primera==1)) {
                  par{ // si no tengo el bus espero a recibirlo, si ya
                  // lo tengo el par demora un solo ciclo de reloj
                     {
                      recibo_bus1 ? aux;
                      while(aux){recibo_bus1 ? aux;};
                     }
                     dir_ram2 = Idir_base + dir_ram;
                  }
                  var32, primera=mem_dat[0 @ dir_ram2],0;
               };
               ...
```

**Figura 6-14** Modificación del cálculo de la dirección del píxel para filtro de imagen.

Con estas mejoras el diseño, para una imagen de 511x511, alcanza una velocidad de 17 MHz. Al aumentar el tamaño de la imagen en un punto la velocidad aumenta a 23 MHz esto ocurre pues el camino crítico es en la siguiente instrucción:

tmp1 = (tmp3.*.(IcolW_div4)).(0..ind_ram_ext)

Esta sentencia se usa para calcular la dirección del píxel que queremos leer, en la RAM. La variable *tmp3* representa la fila en la que estamos leyendo, a ésta la multiplicamos por el número de columnas divido cuatro. Pese a que ahora *tmp3* tiene un bit más

IcolW_div4 pasa de 0111 1111 (511 / 4) a 1000 0000 (512 / 4) por lo que la multiplicación se convierte en un desplazamiento de 7 bits. Si se sigue aumentando el tamaño de la imagen, la velocidad desciende lentamente hasta 17 MHz con una imagen de 724x724 (máximo permitido para una imagen cuadrada en la RAM de la ARC-PCI).

En la siguiente tabla se puede apreciar la relación entre el tamaño de la imagen y la velocidad obtenida para el chip Flex 10K50.

**Tabla 6-1** Cuadro comparativo entre tamaño de imagen y velocidad.

| Filtro 3x3. Imagen | FF | Celdas Lógicas | Frec. Max. [ MHz ] |
|---|---|---|---|
| 500 x 500 | 649 | 1955 | 17.92 |
| 511 x 511 | 649 | 1969 | 16.92 |
| 512 x 512 | 651 | 1744 | 23.25 |
| 516 x 516 | 659 | 1802 | 22.37 |
| 724 x 724 | 659 | 1932 | 17.15 |

La versión completa del código final se puede apreciar en Apéndice F pág. **254**.

## 6.5 Performance del diseño

Vamos a estimar el número total de períodos de reloj que necesitará el diseño para filtrar una imagen.
De examinar el código en Handel C se obtiene la siguiente formula.

$$Tclks = 5 + (IFtot - FFtot + 1)\cdot\left((ICtot - FCtot + 1)\cdot\left(1 + FFtot\cdot\left(2 + FCtot\cdot\left(1 + \frac{2\cdot C1}{FFtot} + 6\right)\right) + 2 + C2 + \frac{1}{4}\right) + 1\right)$$

$$C1 = \begin{Bmatrix} 1.5 \rightarrow FCtot = 3 \\ 2 \rightarrow FCtot = 5 \\ 2.5 \rightarrow FCtot = 7 \\ 3 \rightarrow FCtot = 9 \end{Bmatrix} \qquad C2 = \begin{Bmatrix} 1 \rightarrow \left|\sum coef\right| < 2 \\ 29 \rightarrow \left|\sum coef\right| \geq 2 \end{Bmatrix}$$

Asumimos para simplificar que al diseño se le otorga el bus al siguiente período en que se lo solicita.

Verificamos los tiempos calculados teóricamente insertando en el código C que corre en el PC una función que nos permite medir el tiempo entre que lanzamos el diseño y que detectamos la finalización del mismo. La función es uclock(), ésta nos brinda una granularidad de aproximadamente 840 ns, que si bien es demasiada precisión para el circuito de la ARC-PCI, no lo es para realizar comparaciones con el Filtro Original corriendo en PC actuales. Las medidas de tiempos para éste último las insertamos antes y después de la función que realiza el filtrado de la imagen.
Todas las pruebas se corrieron iniciando el equipo en MS-DOS y fueron realizadas para una Imagen de 500x500 y un filtro de 3x3.

**Tabla 6-2** Cuadro comparativo entre Filtro Original y diseño en Handel C.

| | CPU | Frecuencias [ MHz ] | | Memoria [ MB ] | Proceso de Fabricación del Procesador [ µm ] | Tiempo de cálculo [ ms ] | |
|---|---|---|---|---|---|---|---|
| | | Core µP | System bus | | | Sin división | Con división |
| **HandelC** | --- | --- | --- | --- | --- | 1236(*) | 1657(*) |
| | en ARC-PCI | --- | 16.5 | --- | .42 | 1359 | 1767 |
| **Filtro Original** | Pentium-s | 90 | 60 | 24 | .6 | 2979 | 3122 |
| | Pentium-s | 120 | 60 | 64 | .6 | 2226 | 2330 |
| | Pentium MMX | 200 | 66 | 64 | .35 | 1340 | 1403 |
| | Pentium Celeron | 266 | 66 | 64 | .25 | 535 | 551 |
| | Pentium III | 550 | 100 | 256 | .25 | 259 | 266 |
| | Pentium 4 | 1600 | 400 | 512 | .18 | 107 | 119 |

(*) Tiempo calculado con la formula teórica para frecuencia de trabajo de 16.5 MHz.

**Tabla 6-3** Cuadro comparativo para distintas familias de Altera y Xilinx.

| | Familia | Chip | Proceso de fabricación [ µm ] | FF (%) | LC (%) | Frec. Max. [ MHz ] | Costo (1) (U$S) | $\frac{Costo}{MHz}$ | Tiempo de cálculo (6) [ ms ] |
|---|---|---|---|---|---|---|---|---|---|
| ALTERA | Stratix | EP1S10B672C6 (3) (7) | 0.13 | 5 | 14 | 124.86 | 360 | 2.9 | 163 |
| | " | EP1S10F780C5 (3) (8) | 0.13 | 5 | 14 | 140.79 | 520 | 3.7 | **145** |
| | APEX II | EP2A15B724C7 (2) (7) | 0.15 | 3 | 9 | 83.68 | 1210 | 14.5 | 244 |
| | APEX 20KC | EP20K200CQ208C7 (3) | 0.15 | 6 | 19 | 88.56 | 520 | 5.9 | 230 |
| | APEX 20KE | EP20K60EQC208-1 (3) | 0.18 | 20 | 64 | 75.48 | 112 | 1.5 | 270 |
| | APEX 20K | EP20K100QC240-1 (3) | 0.22 | 16 | 39 | 73.6 | 144 | 1.9 | 277 |
| | Mercury | EP1M120F484C5 (3) (8) | 0.15 | 13 | 27 | 113 | 640 | 5.7 | 181 |
| | Cyclone | EP1C6Q240C6 (2) | 0.13 | 9 | 21 | 99.3 | 59.7 | **0.6** | 205 |
| | ACEX 1K | EP1K30QC208-1 (2) | 0.22 | 28 | 88 | 45.87 | 33.5 | 0.7 | 445 |
| | FLEX 10KE | EPF10K30EQC208-1 (2) | 0.22 | 28 | 89 | 45.87 | 112 | 2.4 | 445 |
| | FLEX 10KA | EPF10K50VRC240-1 (4) | 0.3 | 21 | 62 | 24.39 | 302 | 12.4 | 836 |
| | FLEX 10K | EPF10K50RC240-3 (4) | 0.42 | 22 | 65 | 17.6 | 316 | 17.95 | 1553 |
| | | | | | | | | | |
| XILINX | Spartan XL | XCS40XL-4PQ240 (5) | 0.22 | 47 | 83 | 26.26 | 40 | 1.5 | 777 |
| | Spartan II | XC2S100-5PQ208 (5) | 0.18 | 31 | 58 | 62.54 | 19.55 | **0.3** | 326 |
| | Spartan IIE | XC2S100E-6PQ208 (5) | 0.18 | 31 | 57 | 45.27 | 19.25 | 0.4 | 451 |
| | Virtex | XCV100-4PQ240 (5) | 0.22 | 31 | 57 | 69.2 | 90.3 | 1.3 | 295 |
| | Virtex E | XCV100E-6PQ240 (5) | 0.18 | 31 | 57 | 49.02 | 55.4 | 1.1 | 416 |
| | Virtex II | XC2V250-4FG256 (5) (9) | 0.15 | 24 | 44 | 98 | 89 | 0.9 | **208** |
| | Virtex II PRO | XC2VP2-6FG256 (5) (9) | 0.13 | 26 | 49 | 94.4 | 125 | 1.3 | 216 |

(1) Precios obtenidos de Arrow Electronics[4], Insight Electronics[5] y Avnet Electronics Marketing[6]
(2) Compilado con Quartus Web versión 2.2 SP1 de Altera
(3) Compilado con Quartus II versión 2.1 de Altera
(4) Compilado con Max+Plus II versión 10.1 de Altera
(5) Compilado con ISE versión 4.2i de Xilinx
(6) Calculado con la formula teórica para la frecuencia máxima. Sin realizar divisiones.
(7) Encapsulado BGA (Ball-grid array)
(8) Encapsulado FBGA (FineLine BGA)
(9) Encapsulado Fine pitch BGA

4 Se puede acceder desde la dirección http://www.arrow.com/

5 Se puede acceder desde la dirección http://www.insight-electronics.com/

Decidimos comparar la frecuencia máxima alcanzada por nuestro diseño con otras familias de FPGA más recientes que el Flex 10K50 de la ARC-PCI. Además comparamos los costos de los chip para encapsulados PQFP (Plastic quad flat pack) o RQFP (Power quad flat pack) de aproximadamente 240 patas, salvo casos indicados. Éstos resultados se pueden apreciar en la Tabla 6-3 donde además destacamos los chips para los cuales el filtrado se realiza más rápido, son Stratix de Altera y Virtex II de Xilinx, y también los que producen una mejor relación Costo-MHz y en este caso son Cyclone de Altera y Spartan II de Xilinx.

## 6.6 Entidad interfaz en VHDL

Como vimos en el capítulo 3 el Handel C no genera circuitos que se puedan conectar directamente a un bus, para que este diseño funcionara con la placa ARC-PCI fue necesario generar una entidad en VHDL que nos permitiera colocar buffers triestado en las señales. Para este caso en particular utilizamos como señal de habilitación de éstos la combinación del pedido del bus y la respuesta del árbitro.

Un problema que ya habíamos comentado y que dificultó el trabajar con buffers triestado en éste interfaz es que las señales de datos de la RAM que genera el HCC ya poseen buffers triestado y éstos por defecto están siempre habilitados salvo para las lecturas.

Para solucionar este problema creamos un programa en AWK (llamado BUFT) que modifica las señales de habilitación para los datos de las RAM externas, como nueva señal se utiliza su pulso de escritura (wb), por más detalles de este corrector ver Apéndice B .

El código VHDL de este diseño se puede ver en Apéndice F pág. 258.

## 6.7 Ejemplos de imágenes filtradas

A continuación mostramos unas imágenes de 500x500 filtradas por el diseño en la placa ARC-PCI con filtros de 3x3.

Los filtros utilizados son los siguientes:

**Tabla 6-4** Filtros utilizados en las pruebas con la placa ARC-PCI.

| F1 | | |
|---|---|---|
| -1 | 0 | -1 |
| 0 | 10 | 0 |
| -1 | 0 | -1 |

| F2 | | |
|---|---|---|
| 0 | 0 | 0 |
| 1 | 0 | -1 |
| 0 | 0 | 0 |

| F3 | | |
|---|---|---|
| 0 | 1 | 0 |
| 0 | 0 | 0 |
| 0 | -1 | 0 |

[6] Se puede acceder desde la dirección http://www.avnet.com/em/

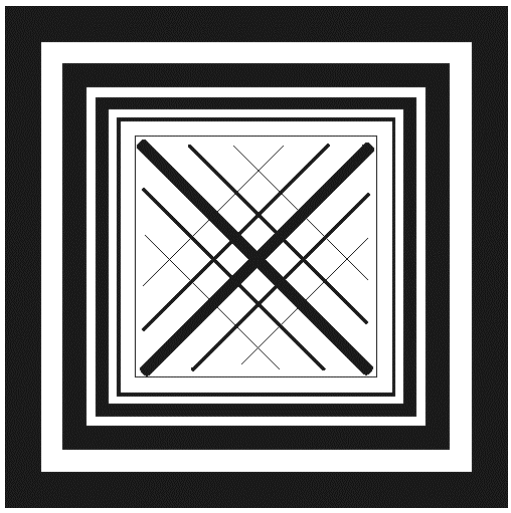


**Figura 6-15** Ejemplo de imágenes a filtrar con la placa ARC-PCI

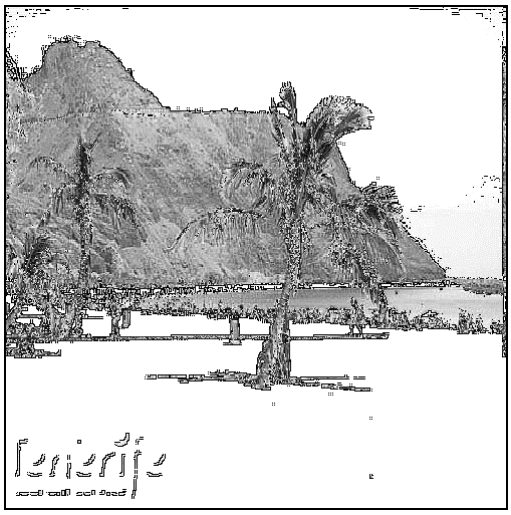


**Figura 6-16** Imagen filtrada con filtro F1

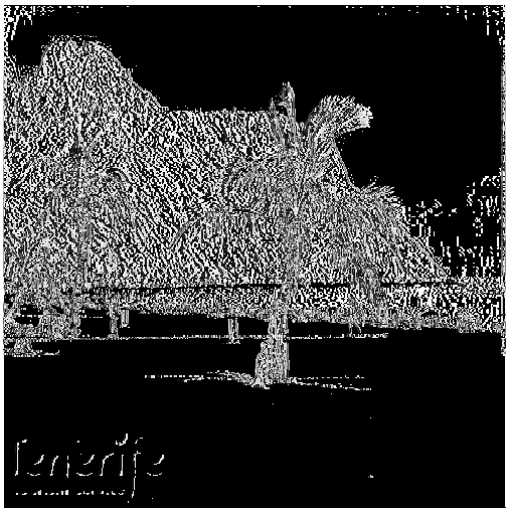


**Figura 6-17** Imagen filtrada con filtro F2

Las imágenes que aparecen a continuación son representadas en negativo.

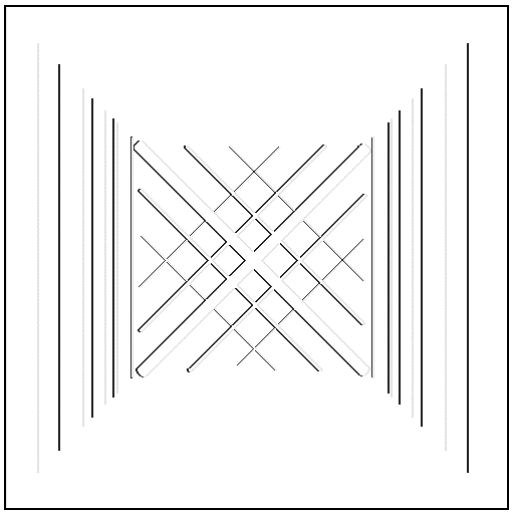


**Figura 6-18** Imagen filtrada con filtro F2

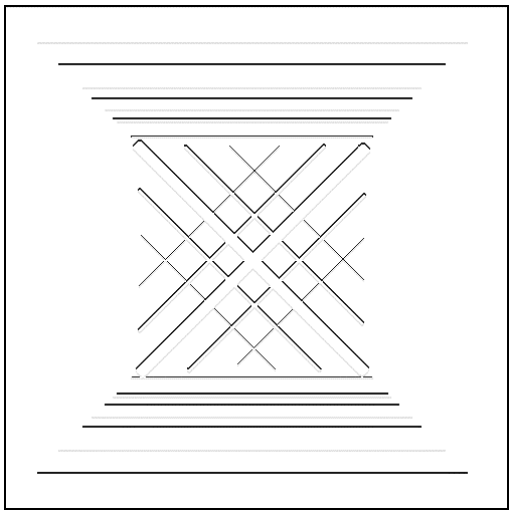


**Figura 6-19** Imagen filtrada con filtro F3

## 6.8 Conclusiones

El filtro funciona satisfactoriamente, obteniéndose iguales resultados que con su par escrito en C.
Si analizamos la Tabla 6-2 vemos que nuestro diseño no es muy rápido comparado con los PC con Pentium 4 o sin ir tan lejos con los Pentium Celeron pero no es muy justo sacar conclusiones comparando el chip de la placa que es de hace 10 años con los últimos procesadores de Intel. Pero si miramos la Tabla 6-3 donde estimamos la velocidad de nuestro diseño para un PLD de última generación vemos que los tiempos se acercan bastante, Stratix contra P4. De ésta tabla además podemos ver que es una opción muy viable tanto en velocidad como en costos la utilización del chip Cyclone de Altera o el chip Spartan II de Xilinx.

Opinamos que respetando la estructura del algoritmo original, y sin agregar otros módulos mediante el traductor, aunque perfeccionáramos más el diseño para Handel C no podríamos obtener resultados mejores a un 15% o 20%. Si cambiamos el algoritmo sí se podría obtener una performance mejor pero dicho cambio escapa al objetivo de esta aplicación final que es la traducción más fiel y directa posible de un algoritmo escrito en lenguaje C.

Un aspecto en el cual debimos ser muy cuidadosos en este diseño fue en la determinación de los anchos de las variables pues los fijamos en función de las dimensiones de la matriz y del filtro.

También queremos destacar que el circuito generado no necesariamente tiene que trabajar con imágenes cuadradas, y que su tamaño (medido en FF y celdas lógicas) crece en forma logarítmica con el tamaño de la imagen y el filtro. De examinar el código en Handel C y los valores obtenidos de la Tabla 6-1 podemos decir que por cada bit de ancho que aumente en el número de filas de la imagen el circuito crecerá entre 4 y 12 flip-flop, por cada bit de las columnas también crecerá entre 4 y 12 y las dimensiones del filtro aportarán 1 o 2 flip-flop. El número de celdas lógicas no crecerá más de un 10% del valor obtenido para el peor caso con un bit menos de ancho, es decir el tamaño para 1023 estará ente 1 y 1.1 veces el tamaño para 511 (en filas y/o columnas)[7].

## 6.9 Posibles mejoras

En lo que respecta a la adaptabilidad del diseño realizado debemos decir que resultaría sencillo hacer que el circuito funcione con tamaños de Imagen y Filtro variables, determinados en tiempo de ejecución como un parámetro más a pasar. No se realizó en este proyecto por no contar con el tiempo suficiente para testear y documentar los resultados.

Se podría trabajar con un cache de varias líneas de la imagen pero no creemos que se mejore demasiado la performance del diseño pues las lecturas a memoria externa y memoria interna llevan los mismos ciclos, nos podríamos ahorrar algunos ciclos del

[7] Todos estos valores dependerán de las optimizaciones que puedan realizar los sintetizadores y en particular del número de filas y/o columnas.

pedido y espera del bus, pero hay un overhead por el trabajo con el cache que hay que tener en cuenta. Esto es si suponemos que el bus no es utilizado por otros dispositivos y que la velocidad del diseño no la fijan los retardos de la RAM donde se almacena la imagen.

Una posible mejora que disminuiría el tiempo de cálculo del circuito es la utilización de módulos rápidos y combinatorios que sustituyan al producto, división y suma. Por ejemplo con la utilización de la megafunción lpm_mult (a través del traductor de XNF a VHDL) en lugar del procedimiento multip8() logramos reducir en un 40% el número de ciclos totales requeridos para el cálculo, la frecuencia máxima aumentará pero no llegará a los 33 MHz pues los sumadores de 16 bits serán una limitante.

# 7 Conclusiones

En este proyecto se estudió la utilización de lenguaje C con la finalidad de diseñar aplicaciones en hardware, en particular sobre lógica reprogramable. Las conclusiones se basan en la experiencia adquirida durante el desarrollo del mismo y en supuestos sobre resultados más generales.

Primero realizamos una búsqueda de herramientas que realizaran la conversión de C a un lenguaje del tipo HDL. Del conjunto que encontramos, se seleccionaron Handel C y Transmogrifier C de acuerdo a los criterios: **(**i**)** precio de las licencias dividido años de validez, **(**ii**)** utilidad o funcionalidad, **(**iii**)** soporte técnico o estado del paquete y **(**iv**)** plataforma.

Para evaluar el funcionamiento de estos compiladores usamos ejemplos típicos de desarrollos en Hardware (árbitro de bus, memoria FIFO) y otros de Software (raíz cuadrada, producto de matrices), los cuales fueron presentados en el capítulo 5. En algunos de estos casos se realizó también una solución en VHDL para comparar el tiempo de diseño y características como recursos del chip y frecuencia máxima.

De esta evaluación observamos que en general el diseño se realizó más rápido y fácil que por los métodos tradicionales, permitiéndonos de esta forma ciclos de prototipación rápidos. También creemos que la curva de aprendizaje de estos compiladores es mayor que para VHDL, sobre todo por Ingenieros no Eléctricos, pues se necesita solamente saber programar en C, aunque para obtener óptimos resultados además es necesario conocer la estructura de los circuitos generados y tener nociones de diseño de circuitos (por ejemplo circuitos secuenciales modo reloj).

Lograr sintetizar un diseño generado con estas herramientas requirió un trabajo extra no previsto. Encontramos que el Transmogrifier C tiene problemas de sintaxis en los archivos XNF generados y no tiene previsto el trabajo con memorias internas así como tampoco operadores de multiplicación o división. Resultó conveniente utilizar un traductor del formato XNF al VHDL que sumado a una serie de correctores (indicados en el Apéndice B) solucionan gran parte de los problemas.

Para Handel C se encontró que tiene problemas en la generación de los archivos VHDL y EDIF, no utiliza las celdas de memoria propias del PLD en la implementación de las RAM internas y no existe la posibilidad de definir señales que tengan salidas en tercer estado. Aquí también gran parte de los problemas y limitaciones fueron solucionados usando correctores y el traductor de XNF a VHDL.

Los ejemplos mencionados anteriormente permitieron evaluar dichas herramientas en los siguientes aspectos:

- Eficacia: si se logran resolver los problemas planteados.
- Eficiencia: si la solución obtenida es más o menos rápida, o consume más o menos recursos del chip que un desarrollo tradicional.
- Flexibilidad: facilidades para resolver problemas de diferentes características.
- Limitaciones: si se encontraron dificultades para la realización de los diseños.

**Eficacia**

Se pudieron resolver los problemas planteados cumpliendo todos sus requerimientos.

Pensando más en general opinamos que especificaciones, por ejemplo, relacionadas a retardos de respuestas de señales no son simples de cumplir. Desde el C no se tiene un control fino del circuito como se tiene en VHDL, por lo tanto no podemos alterar el número de niveles de lógica combinatoria que atraviese una señal. Este tipo de requerimiento deberán solucionarse en la herramienta de síntesis del VHDL o de Place & Route, otra posible solución es trabajar con chips más rápidos.

Especificaciones relacionadas al área ocupada en el chip, son más difíciles de cumplir que las anteriores pues los circuitos generados en general resultan de mayor tamaño que los diseñados por métodos tradicionales. Aquí nuevamente una solución puede ser optar por un chip de mayor capacidad al previsto originalmente.

Especificaciones que estén relacionadas con acciones que haya que tomar dentro del período de reloj son muy complicadas y podrían llegar a ser imposibles de realizar. Por ejemplo si los flip-flop del circuito generado son sensibles al flanco de subida del reloj no puedo realizar acciones para los flancos de bajada del mismo. Otro ejemplo son salidas que tengan que cambiar inmediatamente (antes del próximo ciclo) en función de valores de las entradas, aquí se podría realizar un circuito pero no se puede garantizar que dicha salida este libre de glitches.

Podemos resumir diciendo que los requerimientos que impliquen tiempos de respuesta rápidos (0, 1 o 2 clocks) son más complicados de especificar en C y podría caber la posibilidad de que no se puedan implementar.

**Eficiencia**

Como el lenguaje C (aceptado por los compiladores) es de mayor nivel de abstracción que el VHDL sintetizable es razonable esperar que los tamaños de los circuitos aumenten y que disminuyan los tiempos de diseño.

De la Tabla 7-1 se concluye que tanto los diseños implementados en Transmogrifier C como los implementados con Handel C son de tamaño superior, consumen mayor cantidad de FF y celdas lógicas, que sus pares en VHDL. Éstos compiladores también producen diseños más lentos (hasta 50% menos).

**Tabla 7-1** Cuadro comparativo entre diseños en Transmogrifier C, Handel C y VHDL.

| ALTERA - Chip EPF10K20RC240-4 | | | | |
|---|---|---|---|---|
| | | **FF** | **Celdas Lógicas** | **Frec. Max. [ MHz ]** |
| **Árbitro v. 1** | **VHDL** | 10 | 36 | 49.75 |
| | **TMCC** | 120% | 139% | 86% |
| | **HCC** | 120% | 94% | 93% |
| **Árbitro v. 2** | **VHDL** | 8 | 10 | 99.00 |
| | **TMCC** | 125% | 200% | 65% |
| | **HCC** | 150% | 260% | 62% |
| **Acceso a memoria externa y multiplicador** | **VHDL** | 56 | 276 | 15.43 |
| | **TMCC** | 148% | 142% | 77% |
| | **TMCC** lpm_mult | 146% | 108% | 89% |
| | **HCC** | 102% | 80% | 72% |
| **FIFO** | **VHDL** | 34 | 78 | 28.08 |
| | **TMCC** | 135% | 188% | 95% |
| | **HCC** | 118% | 150% | 100% |
| **Multiplicador sin pipeline** | **VHDL** | 32 | 185 | 17.60 |
| | **HCC** | 106% | 98% | 71% |
| **Multiplicador con 1 etapa de pipeline** | **VHDL** | 72 | 172 | 19.80 |
| | **HCC** | 136% | 123% | 67% |
| **Multiplicador con 3 etapas de pipeline** | **VHDL** | 160 | 201 | 32.57 |
| | **HCC** | 101% | 136% | 88% |
| **Idem opción Fast de compilación** | **VHDL** | 160 | 160 | 103.09 |
| | **HCC** | 101% | 181% | 53% |

| XILINX - Chip XC4010EPC84-3 | | | | | |
|---|---|---|---|---|---|
| | | **FF** | **3-LUT** | **4-LUT** | **Frec. Max. [ MHz ]** |
| **Árbitro v. 1** | **VHDL** | 8 | 4 | 38 | 39.98 |
| | **TMCC** | 125% | 125% | 145% | 91% |
| | **HCC** | 150% | 75% | 76% | 122% |
| **Árbitro v. 2** | **VHDL** | 11 | 1 | 15 | 73.23 |
| | **TMCC** | 72% | 100% | 113% | 126% |
| | **HCC** | 109% | 100% | 93% | 78% |
| **FIFO** | **VHDL** | 26 | 21 | 81 | 38.7 |
| | **HCC** | 150% | 148% | 196% | 69% |

Nota: Los porcentajes fueron calculados tomando como referencia de 100% al valor obtenido para el diseño en VHDL

En lo que respecta a los tiempos de diseño en todos los casos fueron menores, y en particular con Handel C fueron aún menores. Para los ejemplos que desarrollamos la diferencia de horas de diseño entre VHDL y C fue de un 25% (en general) pero recordemos que son ejemplos sencillos. Para grandes diseños o sin ir tan lejos la aplicación final realizada en este proyecto, creemos que ésta diferencia puede ser de más del 50%, pues por ejemplo realizar lazos de control en C es muy sencillo (poner While) mientras que en VHDL es una tarea muy tediosa.

Para los tres métodos de resolución de los ejemplos los tiempos de simulación fueron iguales debido a que siempre terminamos simulando diseños descritos en VHDL.

Si comparamos el desempeño del circuito generado como aplicación final, podemos observar en la Tabla 7-2 que su tiempo de cálculo es similar a un Pentium MMX de 200 MHz y si mejoramos los multiplicadores usando la función lpm_mult los tiempos de cálculo se reducen en un 40% acercándose a un Pentium Celeron de 266 MHz. Por otro lado si consideramos correr el diseño en un FPGA de última generación, como por ejemplo Cyclone, se puede comparar su desempeño con un Pentium III de 550 MHz y mejorando el circuito podemos trabajar con tiempos similares a un Pentium 4 de 1.6 GHz. Si consideramos una solución más costosa, la utilización del Stratix de Altera nos permite obtener tiempos de cálculo de 84 ms (considerando las mejoras al multiplicador)

**Tabla 7-2** Cuadro comparativo para aplicación final.

| | PC o CHIP | Frec. Max. [ MHz ] | Tiempo de cálculo [ ms ] | Tiempo de cálculo al 60% (3) [ ms ] |
|---|---|---|---|---|
| Circuito generado con Handel C | **FLEX10K** (EPF10K50RC240-3) | 16.5 (1) | 1359 (2) | 815 |
| | **Cyclone** (EP1C6Q240C6) | 99.3 | 205 (4) | 123 |
| Algoritmo en C corriendo en PC | **Pentium MMX** 200 MHz (bus 66 MHz) | 200 | 1340 | --- |
| | **Pentium Celeron** 266 MHz (bus 66 MHz) | 266 | 535 | --- |
| | **Pentium III** 550 MHz (bus 100 MHz) | 550 | 259 | --- |
| | **Pentium 4** 1.6GHz (bus 4*100MHz) | 1600 | 107 | --- |

(1) Frecuencia de trabajo en la placa ARC-PCI.
(2) Tiempo medido con la placa ARC-PCI sin realizar divisiones.
(3) Tiempo de cálculo supuesto en caso de sustitución de procedimiento multip8() por megafuncion lpm_mult.
(4) Tiempo obtenido con la formula teórica sin tener que realizar divisiones.

## Flexibilidad

Si pensamos en la codificación de un algoritmo dado, encontramos que es de similar complejidad su realización en lenguaje ANSI C que en las variantes de C definidas por Transmogrifier y Handel.

Por otro lado como el circuito generado está especificado en VHDL, podemos fácilmente integrarlo (o instanciarlo) en diseños más complejos. Por ejemplo puedo realizar las tareas tediosas de un diseño desde C y los detalles finos hacerlos en VHDL como ser el manejo de memorias dinámicas.

Con respecto al tipo de máquina generado podemos decir que ambos compiladores generan máquinas tipo Mealy pero realizando los programas en forma cuidadosa podemos simular el comportamiento de una máquina tipo Moore.

Por último queremos destacar algunas características particulares de cada compilador, podemos decir que Transmogrifier C tiene un mayor control de los puertos, de salida y

entrada, al poder definirles propiedades como por ejemplo que sean registrados o no. En cambio en Handel C para registrar los puertos de entrada debemos guardarlos específicamente en variables y para los puertos de salida mantiene su valor por un período de reloj obligándonos a realizar algunas "pisadas" en la programación para lograr registrar los datos. Este comportamiento de Handel C según el problema podría o no ser una dificultad.

También para Transmogrifier C vemos como un aspecto muy útil la compilación en hilos de control que permite separar un diseño en subcircuitos que corren en paralelo. Incluso permite integrar diseños en XNF generados por otras herramientas.

Otra característica importante a destacar es que Handel C ofrece la posibilidad de trabajar las variables con y sin signo obteniendo para este último caso una lógica más sencilla. Con el otro compilador solamente podemos utilizar números con signo, lo que implica que para trabajar sin signo se tengan que definir variables de un ancho mayor al necesario.

**Limitaciones**

No es muy sencillo adaptar un programa ya escrito en C a Transmogrifier C o Handel C, depende de la complejidad de éste (por ejemplo no se puede usar recursión) y sobre todo de los tipos utilizados (punteros, arrays, etc.). Otro factor importante para poder realizar las traducciones desde C son las operaciones realizadas pues no se cuenta con, por ejemplo, división, potencia, funciones trigonométricas, etc.

Estas herramientas sólo generan circuitos secuenciales modo reloj, con un único reloj global y todos los flip-flop sensibles al mismo flanco (de subida por defecto).

Para el Transmogrifier C existe una limitación en el ancho de las constantes (no tipo const) que deben ser menores a 32 bits. Y en Handel C, como mencionamos anteriormente, no permite la generación de puertos con salida en tercer estado dificultando así poder insertar directamente los circuitos generados en sistemas con buses. El único caso en que crea puertos bidireccionales es para las señales de datos de las memorias externas pero el manejo del buffer es poco habitual pues esta siempre habilitado y se deshabilita en los ciclos de lectura.

**Uso comparativo de las herramientas**

A continuación mencionaremos los puntos más relevantes de cada compilador que pudimos observar en el desarrollo de los distintos diseños.

TMCC

- Buen control del tipo de puerto generado (con o sin FF, etc.).
- Compilación en hilos de control
- Adaptación de las fuentes del compilador para realizar mejoras.

HCC

- Operadores con y sin signo. Manipulación de los bits de las variables.
- Manejo de memorias nativo.
- Mayor claridad en el código del programa facilitando su depuración.
- Resulta más sencilla la programación de los algoritmos.

**Comentarios finales**

Destacamos a continuación las principales ideas vertidas en estas conclusiones:

- Rápido desarrollo de algoritmos.
- Facilidad para modificar el diseño.
- Ciclos de prototipación rápidos.
- Fácil programación o descripción de los circuitos desde C
- Buena integración con otros circuitos
- Amplio espectro de requerimientos cubiertos

Queremos mencionar además que Handel C continúa hoy en día desarrollándose y la información comercial de sus versiones más modernas prometen superar muchas de las limitaciones encontradas en este proyecto. También mejoran el ambiente de desarrollo y simulación.

**Trabajos futuros**
Posibles líneas de trabajo o estudio serían los siguientes:

- Análisis de herramientas que acepten SystemC
- Mejoras al Transmogrifier C
- Mejoras al traductor de XNF a VHDL

Describiremos brevemente cada uno de ellos.

Análisis de herramientas que acepten SystemC
Es importante no dejar de lado la tendencia del mercado actual que es la utilización de SystemC. ¿Qué es SystemC? Como mencionamos en el Apéndice D es un conjunto de librerías para C++ que permiten la especificación de múltiples procesos concurrentes descritos tanto a nivel funcional como a nivel de registros (RTL).
Creemos que resulta muy interesante la idea planteada por el grupo que lleva adelante ésta iniciativa, OSCI, lamentablemente al momento del desarrollo de este proyecto no tuvimos acceso a ninguna de las herramientas que sintetizarán lenguaje C con esta extensión de SystemC.

Mejoras al Transmogrifier C
Como están disponibles las fuentes es interesante la posibilidad de ampliar y corregir el funcionamiento de este compilador. Si bien durante el desarrollo del proyecto modificamos las fuentes, solamente nos limitamos a la corrección de problemas sintácticos en la generación de los símbolos del netlist en XNF.
Es factible la realización de un módulo que genere VHDL directamente y podría también analizarse la posibilidad de ampliar el subset del ANSI C que soporta.

Mejoras al traductor de XNF a VHDL
El aumentar las funcionalidades que brinda el traductor es una tarea muy viable. A lo largo del proyecto incluimos la posibilidad de trabajar con RAM internas, multiplicadores como lpm_mult, entre otras. El agregar otras funciones especiales u operadores no soportados como la división convertirían al conjunto compilador-traductor en un paquete más atractivo.

# A Algoritmos de operaciones básicas

En este apéndice se detallan los algoritmos descritos en distintos capítulos y además algunos otros que complementan el marco teórico.

## A.1 Sumadores

### A.1.1 Sumadores de 1 bit

Referencias [8] [9]

**Semisumador H.A. (Half Adder)**

Es un sumador que admite 2 entradas binarias **A** y **B** en función de las cuales genera 2 salidas también binarias **S** (Suma) y **C** (Carry). Decimos que es medio sumador ya que no tiene entrada para el bit de carry de una etapa anterior.

Inputs: $A, B$

Outputs[8]: $S = A \oplus B = /A \cdot B + A \cdot /B$

$C = A \cdot B$

| Tabla de verdad | | | |
|---|---|---|---|
| **A** | **B** | **S** | **C** |
| 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |

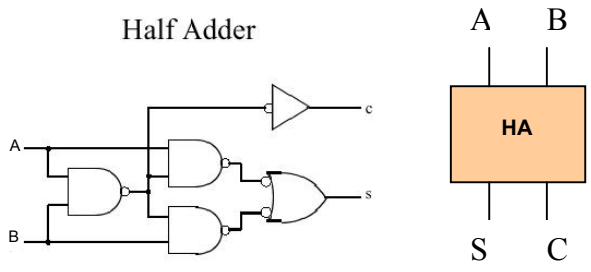



**Figura A-1** Tabla de verdad y circuito del sumador "Half Adder".

**Sumador completo F.A. (Full Adder)**

Este sumador considera además de los 2 sumandos, el carry de la etapa precedente (que llamaremos $C_{i-1}$). En las figuras siguientes se muestran el diagrama en bloques y la tabla de verdad para la etapa i-esima.

[8] La barra "/" indica negación.

Inputs: $A, B, C_{i-1}$

Outputs: $S = A \oplus B \oplus C_{i-1}$

$S = A./B./C_{i-1} + /A.B./C_{i-1} + /A./B.C_{i-1} + A.B.C_{i-1}$

$C = A . B + ( A + B ) C_{i-1}$

| Tabla de Verdad | | | | |
|---|---|---|---|---|
| A | B | $C_{i-1}$ | S | C |
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 |

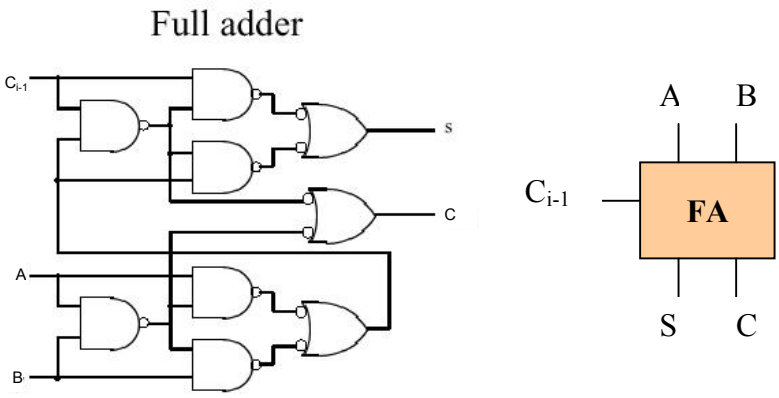



**Figura A-2** Tabla de verdad y circuito del sumador "Full Adder".

Una posible implementación de un sumador full adder se obtiene tratando los bits de a 2, con semisumadores en cascada, utilizando dos sumadores half adder como se ilustra debajo.

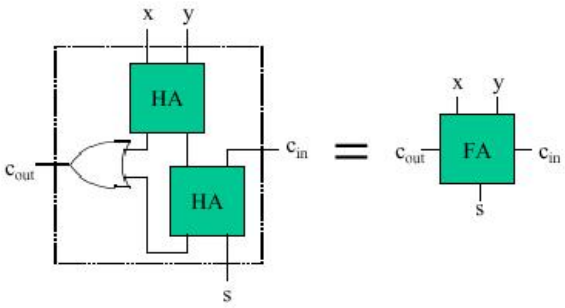



**Figura A-3** Implementación de Full-Adder con cascada de Half-Adder.

## A.2 Sumadores de n-bits

### A.2.1 Con propagación del Carry o "Ripple Carry"

Referencias [8] [10] [9]

En los circuitos de suma con propagación de carry, tenemos que la etapa n-ésima no tiene el resultado correcto de la suma, hasta que no recibe el carry de la previa, la que a su vez lo necesita de la anterior a ella, etc. Entonces la suma se demora hasta que el carry se propaga a lo largo de toda la cadena de n bits.

En la Figura A-4, la línea punteada indica el camino crítico que determina la velocidad máxima que puede tener un sumador de este tipo. El tiempo total de cálculo de la suma su pude escribir como:

$$T_{suma\ total} = (n-1) * T_{carry} + máximo\ (T_{suma}, T_{carry})$$

Donde $T_{suma}$ es el tiempo de propagación de cada F.A. para la suma y $T_{carry}$ es el tiempo de propagación de cada F.A. para el carry.

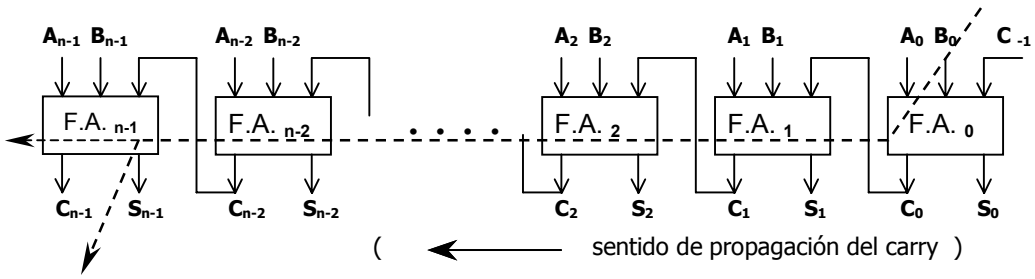



**Figura A-4** Sumador con Propagación de Carry.

## A.2.2 Sumadores rápidos

En estos sumadores de varios bits se trabajan algo más las ecuaciones (la suma y el carry), principalmente para disminuir el tiempo de generación del carry (función de los niveles de compuerta o sea de los caminos en serie de la señal).
Existen sumadores (sumadores rápidos), que generan el carry directamente (sumador de carry anticipado "look-ahead carry", sumador por "bypass" y "carry select"), con lo que acortan el tiempo requerido por la suma, por supuesto a costa de mayor "hardware" o circuitería. Para el desarrollo de estos sumadores se trabaja sobre la cadena de acarreo para eliminarla o limitarla.

### A.2.2.1 Sumador rápido por bypass
Referencia [10]

En las siguientes figuras se esquematiza un sumador de 12 bits con acarreo anticipado o desvío de acarreo cada 4 etapas.
Si el carry proveniente de etapas anteriores vale "1" ($C_{entrada}$ = 1) y los sumandos son tales que este acarreo debe propagarse hasta la salida, lo cual ocurre si en cada posición al menos uno de los sumandos tiene un 1. Esta propagación se hace a través de las puertas G (3 niveles) en lugar de tener que atravesar los 4 sumadores (8 a 12 niveles de compuerta en total) como se muestra en la Figura A-5.

Lo que se intenta evitar es la propagación del carry, logrando este por anticipado y enviándolo a la etapa siguiente, pero puede no poder anticiparse, entonces la propagación debe hacerse a través de la cadena de sumadores. En caso de poder anticipar el carry, el tiempo de propagación de éste es menor. En la Figura A-6 vemos los distintos bloques y como se recibe el carry del circuito bypass.

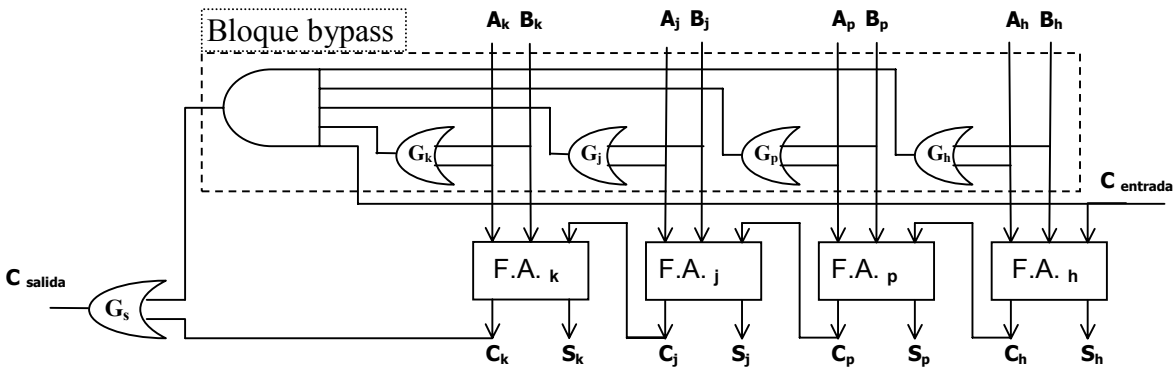



**Figura A-5** Una etapa del sumador por Bypass.

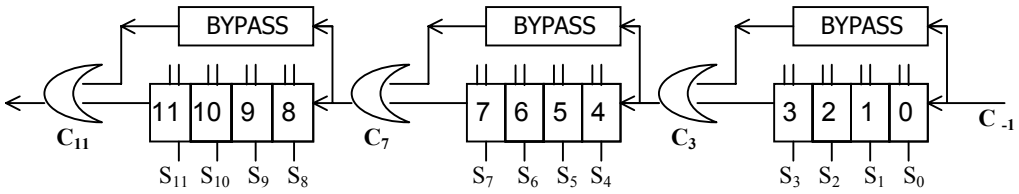



**Figura A-6** Sumador por Bypass (circuito total).

El tiempo máximo es tal que no pueda realizar el anticipo:

$$T_{suma\_max} = (n-1) * T_{carry} + máximo\ (T_{suma}, T_{carry})$$

Ahora el tiempo mínimo en que puedo obtener un resultado de suma válido es:

$$T_{suma\_min} = m * T_{suma} + (p-1) * 3 * T_{compuerta}$$

Siendo "m" el número de sumadores F.A. en cada bloque y "p" el número de bloques implementados con bypass.

### A.2.2.2 Sumador paralelo con acarreo paralelo o anticipado (Carry Look-Ahead)

Referencia [9]

Para solucionar el retardo, con este circuito obtengo de forma rápida el acarreo de entrada de todas las etapas del sumador (Figura A-4). Por tanto los bits de la suma se obtienen simultáneamente.

La idea consiste en obtener la generación de acarreo ( $G_n$ ) y la propagación de acarreo ( $P_n$ ) de una forma independiente de cada bit para la obtención del último acarreo mucho más rápido ( antes tenía que esperar a que se realizaran todos para obtenerlo ).

Se procede utilizando las formulas básicas de la siguiente forma:

$$S_n = C_{n-1} \oplus A_n \oplus B_n = C_{n-1} \oplus P_n$$
$$C_n = A_n \cdot B_n + C_{n-1} (A_n \oplus B_n) = G_n + C_{n-1} \cdot P_n$$

Siendo

$$P_n = A_n \oplus B_n$$
$$G_n = A_n \cdot B_n$$

Lo que se esta haciendo es basándose en hardware hacer más rápido el circuito, generando el carry y la suma simultáneamente.

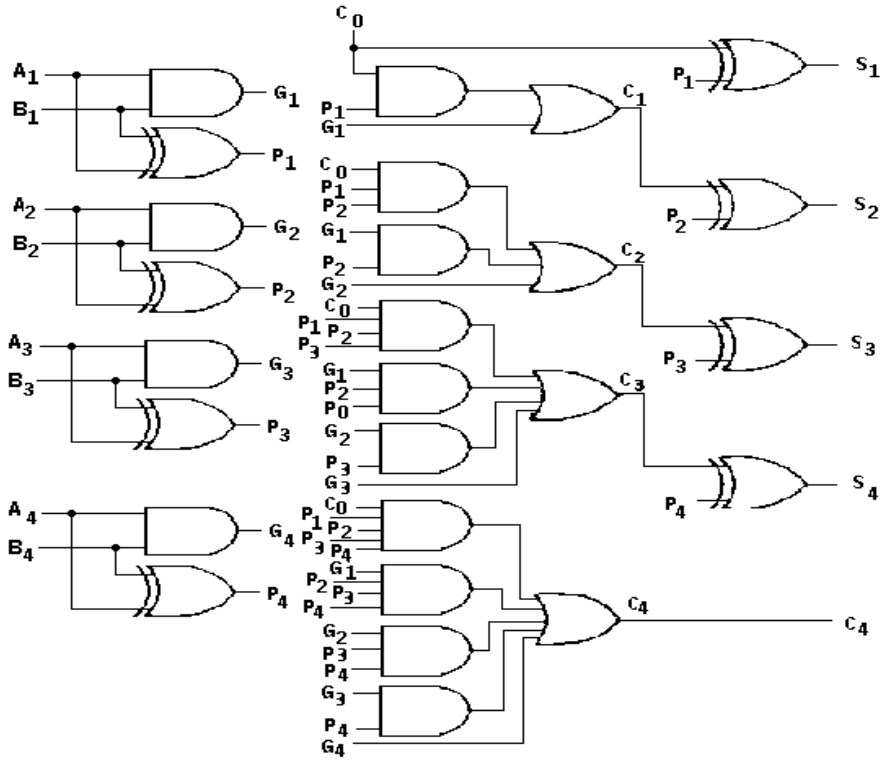


**Figura A-7** Sumador Carry Look-Ahead

### A.2.2.3 Sumador "Carry Select"

Esta suma esta basada en la generación de ambos resultados de las sumas parciales (bit a bit) para el caso de tener carry 0 o 1. Luego de obtener ambas sumas, la suma correcta es seleccionada según sea el carry en cuestión.

En la figura se ven los bloques 1-Carry Sum y 0-Carry Sum, estos dos bloques calculan el resultado de la suma para los dos carry. Luego ambos resultados entran al bloque Carry Determination and Sum Selection el cual distingue cual es el carry y extrae el resultado válido.

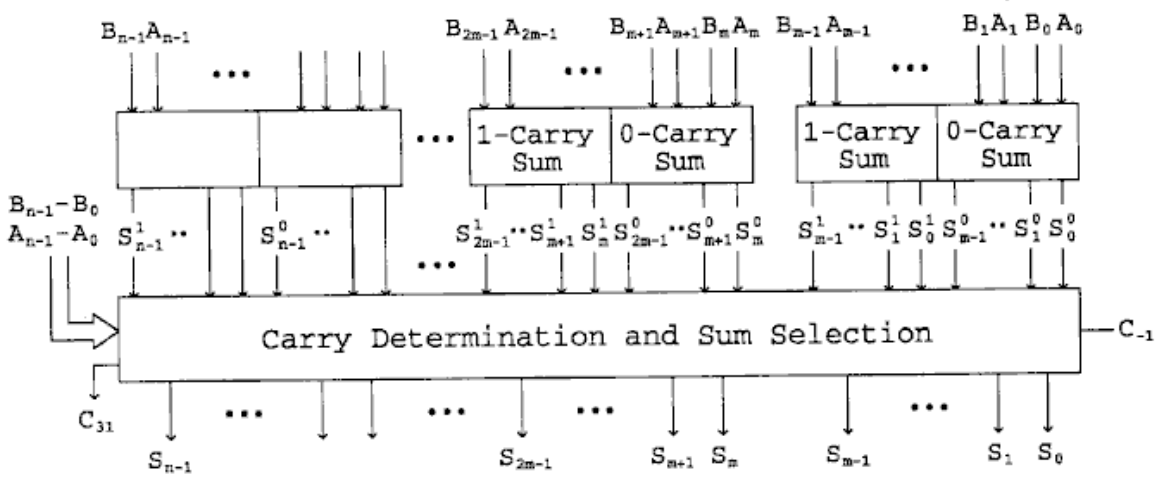



**Figura A-8** Sumador Carry Select

### A.2.3 Sumadores serial o secuencial

Referencia [10]

Los números a sumarse pueden guardarse en celdas de memoria y efectuarse la suma bit a bit de cada par de dígitos y el carry de la suma anterior, guardándose los resultados en otro registro. Este proceso serie requiere de sincronismo de las operaciones, lo que se logra mediante un reloj (clock) constituyendo así un sistema secuencial.

En la Figura A-9 se muestra el principio de operación que describiremos a continuación.

El contenido de los registros circula un lugar hacia la derecha en cada pulso de reloj. El proceso se inicia cargando **A** y **B** en los registros de **n** bits y poniendo a cero el flip-flop tipo D (DFF). La salida del sumador corresponderá a la suma del LSB de **A** y del LSB de B, dando el LSB del resultado (en **S**i) y el primer carry (en **C**i).

Con el primer pulso de reloj, el carry aparece en **Q** y el bit suma S**0** es registrado en el MSB del registro **S**; además los contenidos de los registros **A** y **B** circulan un lugar hacia la derecha. En esta situación **Si** será el bit suma del LSB de **A** y del LSB de **B** (que ahora son los segundos bits de los números iniciales) y del transporte de la suma anterior (que es Q). Al segundo pulso de reloj, esta suma se copia en **S** (MSB) que circula una posición hacia la derecha. Así sucesivamente hasta tener realizada toda la operación.

Como se observa, el proceso permitió sumar "**n"** bits con un único sumador completo. El precio que hubo que pagar fue el tiempo insumido, por cuanto se necesitaron "**n"** pulsos de reloj para completar la operación.

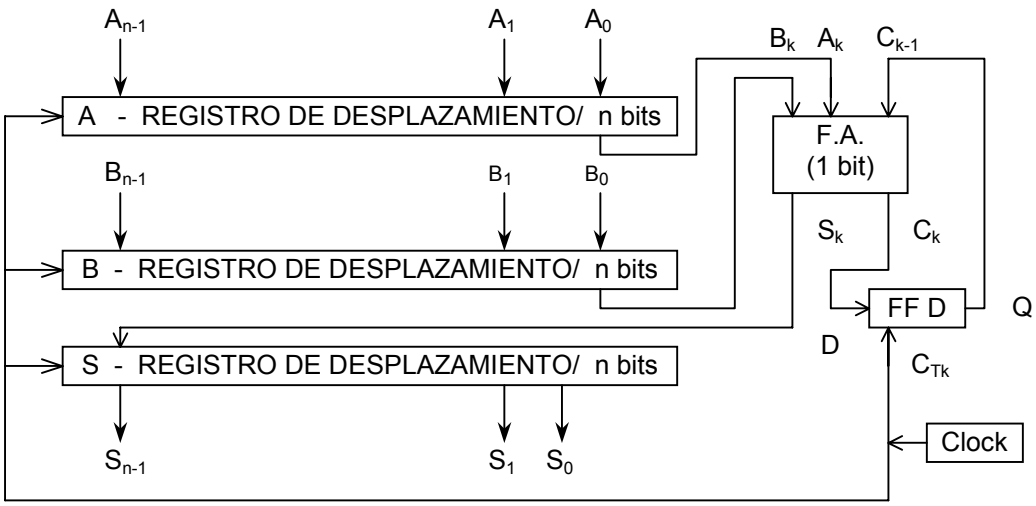



**Figura A-9** Sumador Serie o secuencial

### A.2.4 Over flow de números sin signo

Cuando un resultado no puede ser representado con la cantidad de bits de con la cual estamos trabajando se dice que existe desbordamiento (overflow). En la suma de naturales esto ocurre cuando el bit más significativo (n-1) produce un acarreo C. Entonces la condición de overflow (OV) en la suma de naturales es $C_{n-1}$ .

## A.3 Resta

### A.3.1 Restadores de 1 bit

Referencia [10]

#### A.3.1.1 Semi-Restador

De manera similar a lo efectuado con el sumador, podríamos construir un semi-restador tal que cumpla la siguiente tabla de verdad y cuya posible implementación se muestra en la figura siguiente.

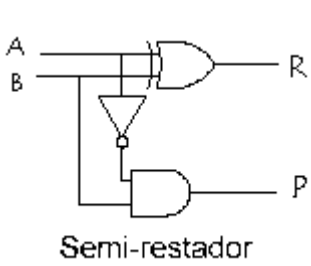
Restador de 2 bits. Admite 2 entradas binarias **A** y **B** en función de las cuales genera 2 salidas también binarias **R** (Resta) y **P** (Préstamo)

Inputs: $A, B$

Outputs: $R = A \oplus B = /A \,.\, B + A \,.\, /B$

$P = /A \,.\, B$

| Tabla de verdad | | | |
|---|---|---|---|
| **A** | **B** | **R** | **P** |
| 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 |



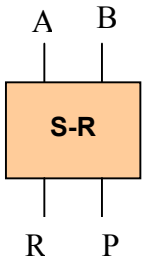




**Figura A-10** Tabla de verdad y circuito del Semi-restador.

Nótese que la expresión de la diferencia D coincide con la de la suma S.

#### A.3.1.2 Restador completo

Un restador de 3 bits, Considera además de los 2 bits a restar, el préstamo de la etapa precedente $P_{n-1}$. En las figuras siguientes se muestran el diagrama en bloques y la tabla de verdad para la etapa n-esima.

Inputs: $A, B , P_{n-1}$

Outputs: $S = A \oplus B \oplus C_{n-1}$

$S = A./B./C_{n-1} + /A.B./C_{n-1} + /A./B.C_{n-1} + A.B.C_{n-1}$

$P_n = P_{n-1}. /A_n + P_{n-1} \,.\, B_n + /A_n \,.\, B_n$

| Tabla de verdad | | | | |
|---|---|---|---|---|
| $A_n$ | $B_n$ | $P_{n-1}$ | $R_n$ | $P_n$ |
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 1 |
| 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 0 |
| 1 | 0 | 0 | 1 | 1 |
| 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 0 | 0 | 0 |
| 1 | 1 | 1 | 1 | 1 |

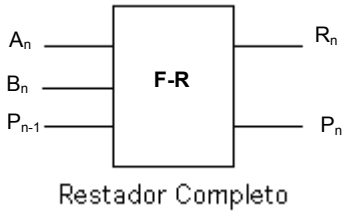



**Figura A-11** Tabla de verdad y circuito del restador completo

En la resta también se pueden implementar, como en la suma, la generación directa del Periodo, así de esta forma estamos disminuyendo el tiempo de propagación y obtenemos el resultado valido en menor tiempo.

## A.3.2 La aritmética binaria de números con signo.

La aritmética del sistema de numeración binario que hemos visto ha sido la de números sin signo, pero en este apartado vamos a estudiar el tratamiento especial que tiene la consideración del signo. Y esto nos dará lugar a diferentes representaciones.

### A.3.2.1 Representación magnitud y signo.

La representación en **magnitud-signo** de un número consiste en escribir el número en sistema binario natural y añadir un bit a la izquierda con valor " 0 " para números positivos y con valor " 1 " para números negativos.

### A.3.2.2 Representación mediante complementos.

El utilizar la representación **magnitud-signo** en un sistema digital significaría la utilización de un circuito para realizar las sumas y otro distinto para realizar restas. Por este motivo se utiliza **complemento a dos** que son representaciones para convertir restas en sumas (se resta sumando) y así simplificar la circuitería. Importante, de esta forma no tenemos que tomar en cuenta el bit de overflow.

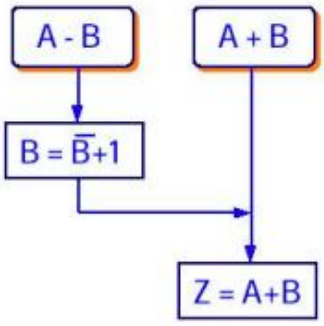



**Figura A-12** Proceso de conversión de resta en suma mediante complementos

### Complemento a uno ( C1 )

El complemento a uno es básicamente invertir los bits, los unos pasan a ser ceros y los ceros pasan a ser unos.

### Complemento a dos ( C2 )

El **complemento a dos** de un número binario es el que se obtiene sumando 1 al C1. Los pasos que se deben seguir para decodificar un número representado en C2 son los siguientes, teniendo en cuenta que el bit de más peso (MSB), representa el signo:

- Si el primer bit más significativos 0 el número es positivo. Entonces el número representado es el equivalente del número binario que forma el resto de bits.
- Si el bit más significativo es 1 el número es negativo. Entonces el número representado es el opuesto del equivalente decimal del número binario que forma su complemento a dos.

La operación aritmética de resta en C2 se realiza de la siguiente forma:

La representación binaria de +42 es 0101010 y poniendo el bit de signo será 00101010, mientras que el C2(-19) será tomando el binario de 19 que es 0010011 realizo el C1 obteniendo el 1101100 y su C2 será C1+1 = 1101101. Le añado un uno delante para indicar que es negativo y entonces realizo la resta (que se ha transformado en una suma).

```
  42          00101010          00101010
 -19    ⇨    -00010011    ⇨    11101101+
----          --------          --------
  23                            00010111
```

**Figura A-13** Ejemplo de conversión de resta a suma en complementos

Si la operación fuera una suma se realizaría la codificación en C2 de ambos números como se hizo para el número 42 del ejemplo anterior.

Hay que tener en cuenta que el bit más significativo vendría a ser como un bit de signo, y hay que tener cuidado con el rango de números que puedo representar. Por ejemplo, en caso de tener 8 bits el conjunto de números que puedo representar el [-128 .. 127].

## A.3.3 Overflow

Se produce resultado incorrecto (overflow), cuando el signo de ambos operandos es diferente del signo del resultado.
Ej.: si estamos trabajando con 8 bits podemos representar números entre –128 y 127, al hacer -100-120 = -220, para expresar el resultado correcto se precisan 9 bits (en C2), al tomar los 8 bits menos significativos obtenemos un resultado positivo.

```
 +  -100        +   1001 1000
    -120            1000 0000
    ----            ---------
    -220         1  0010 1000
```

Nótese que el Carry-out, no determina esta condición, ya que existen resultados correctos que tienen Carry-out igual a uno. Esto se debe a que, al sumar dos números de diferente signo, la magnitud del resultado es siempre menor o igual que la del mayor operando. Por lo tanto el resultado, en este caso, es siempre representable.

## A.4 Multiplicación binaria

Referencias [9] [11] [12]

### A.4.1 Multiplicación de enteros sin signo

La multiplicación binaria es igual a la multiplicación decimal, se realiza de la misma forma. A continuación describiremos dos algoritmos para poder llevar a cabo esta operación.

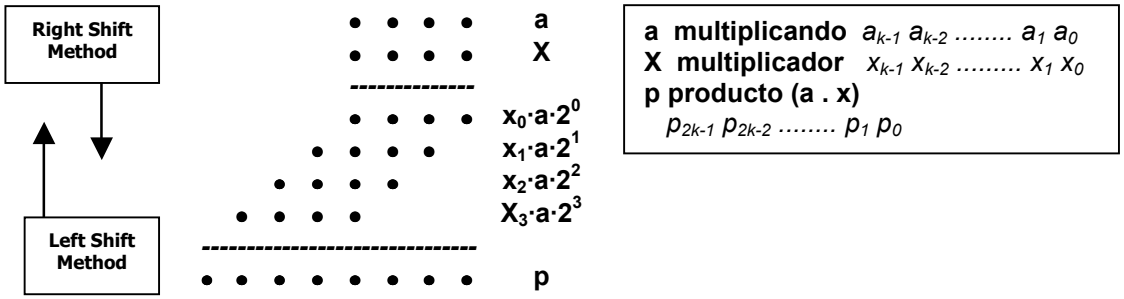



**Figura A-14** Esquema de una multiplicación.

#### A.4.1.1 Right shift algorithm (algoritmo de corrimiento hacia la derecha)

En la multiplicación de números binarios, al multiplicar **( a . $X_i$ )** se hace el *and* de cada elemento del multiplicando **( a )** con el elemento Xi del multiplicador. O como se puede ver en la Figura A-16, a través de un multiplexor, que según sea el elemento $X_i$, tengo el multiplicando o 0 a la salida del multiplexor. Los elementos del multiplicando Xi van a ir entrando al mutliplexor del menos significativo al más significativo. Luego realizo la suma de k bits con lo que obtengo del multiplexor y el producto parcial anterior, tomando de este los k bits más significativos, esto da como resultado el siguiente producto parcial. Posteriormente realizo el corrimiento de este producto parcial en un bit, en este caso es a la derecha, y repito el ciclo.

Luego de haber finalizado el ciclo, obtengo el resultado de la multiplicación, que tendrá un largo máximo de n+m (largo del multiplicando + largo del multiplicador)

**Right-shift Algorithm**

```
a          1 0 1 0
x          1 0 1 1
=====================
P(0)       0 0 0 0
+x0.a      1 0 1 0
---------------------
2p(1)    0 1 0 1 0
P(1)       0 1 0 1 0
+x1.a      1 0 1 0
---------------------
2p(2)    0 1 1 1 1 0
P(2)       0 1 1 1 1 0
+x2.a      0 0 0 0
---------------------
2p(3)    0 0 1 1 1 1 0
P(3)       0 0 1 1 1 1 0
+x3.a      1 0 1 0
---------------------
2p(4)    0 1 1 0 1 1 1 0
P(4)       0 1 1 0 1 1 1 0
=====================
```

**Figura A-15**
Ejemplo multiplicación con corrimiento hacia la derecha

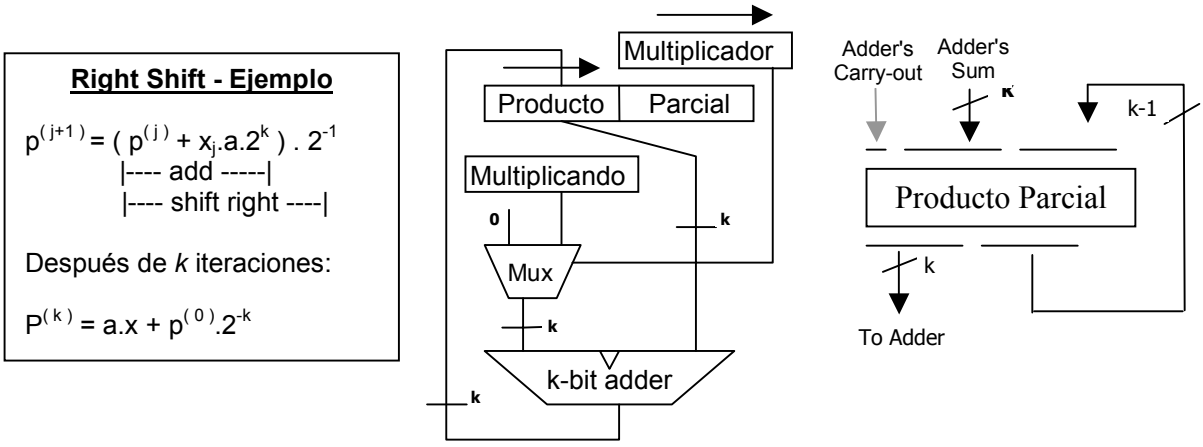



**Figura A-16** Circuito de multiplicación con corrimiento hacia la derecha

### A.4.1.2 Left shift algorithm (algoritmo de corrimiento hacia la izquierda)

Es otro método, que tiene como variante que el corrimiento en vez de hacerse hacia la derecha es a la izquierda. La variante es que al multiplexor van a entrar los bits del multiplicando del más significativo al menos significativo. El resto del esquema es similar.

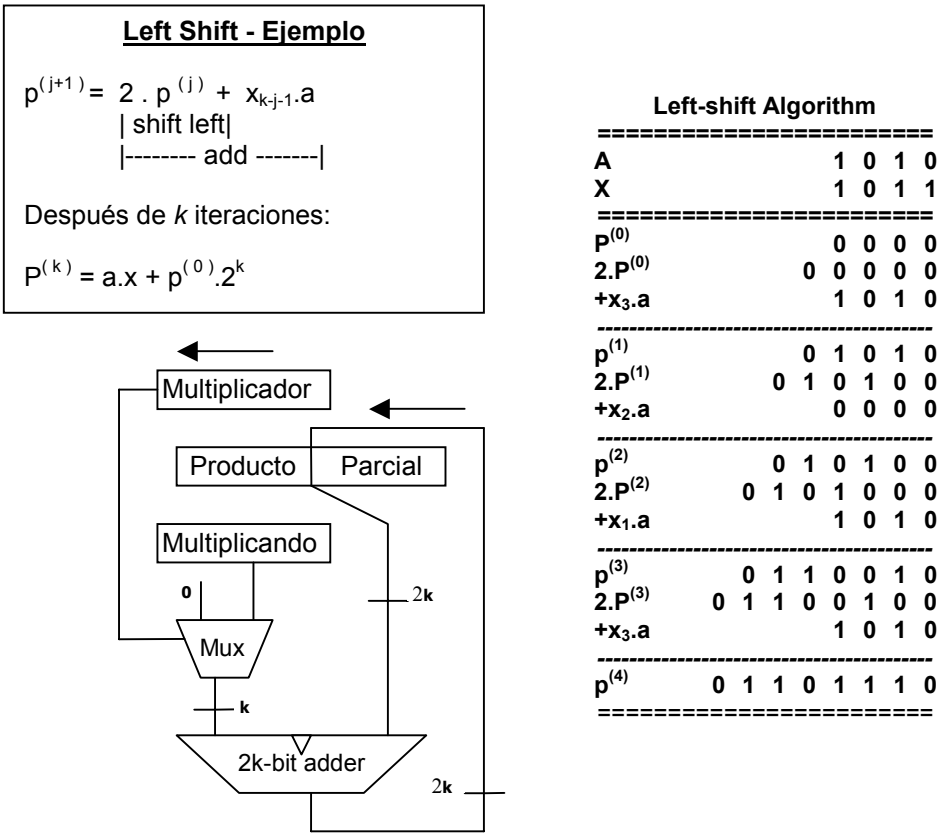



**Figura A-17** Ejemplo y circuito de multiplicación con corrimiento hacia la izquierda

## A.4.2 Multiplicación de enteros con signo

La multiplicación en complemento a 2 (C2) no es coherente con la multiplicación sin signo. Sin embargo, puede modificarse el algoritmo de suma y desplazamiento para que opere directamente en C2. Para ello hay que distinguir 3 posibles casos:

**Caso 1: Multiplicando positivo y multiplicador positivo**
No requiere modificación.

**Caso 2: Multiplicando negativo y multiplicador positivo**

Extender correctamente el signo del multiplicando.

Al sumar los productos parciales se asume implícitamente que los bits mas significativos de los sumandos son 0, esto solo es correcto si el multiplicando es positivo, si es negativo se esta extendiendo incorrectamente su signo.

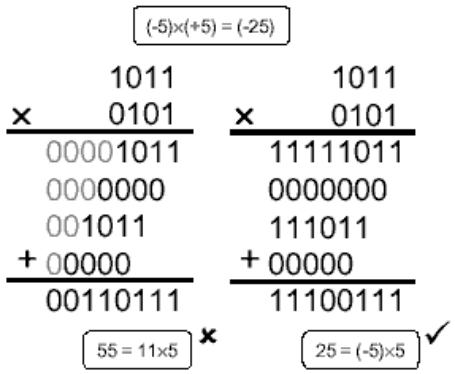



**Figura A-18** Ejemplo de división con signo con multiplicador positivo

**Caso 3: Multiplicador negativo**

En la ultima iteración sumar o restar el multiplicando en función del signo del multiplicador.

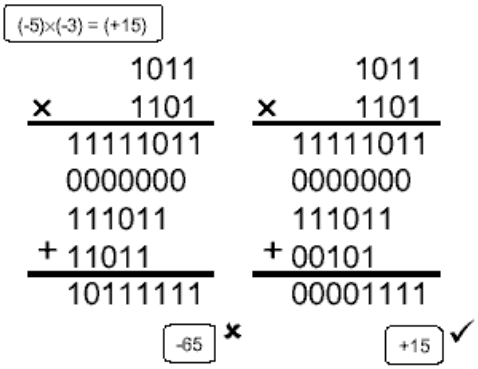



**Figura A-19** Ejemplo de división con signo con multiplicador negativo

## A.4.3 Multiplicación Combinacional

La idea es el diseño de una celda combinacional que toma un digito del multiplicando, otro del multiplicador y otro del producto parcial anterior y genere un bit del siguiente producto parcial.
Hay que repetir este procedimiento como bit parciales haya que generar.

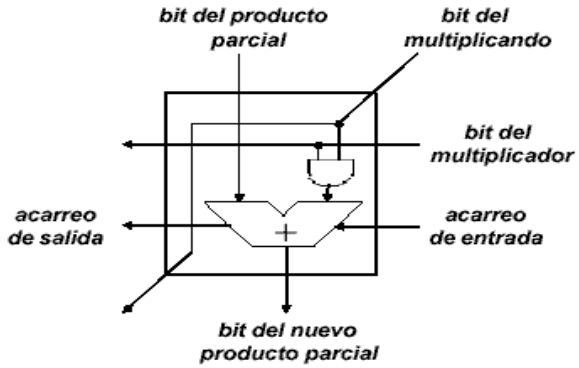



**Figura A-20** Diseño del bloque de un multiplicador combinacional

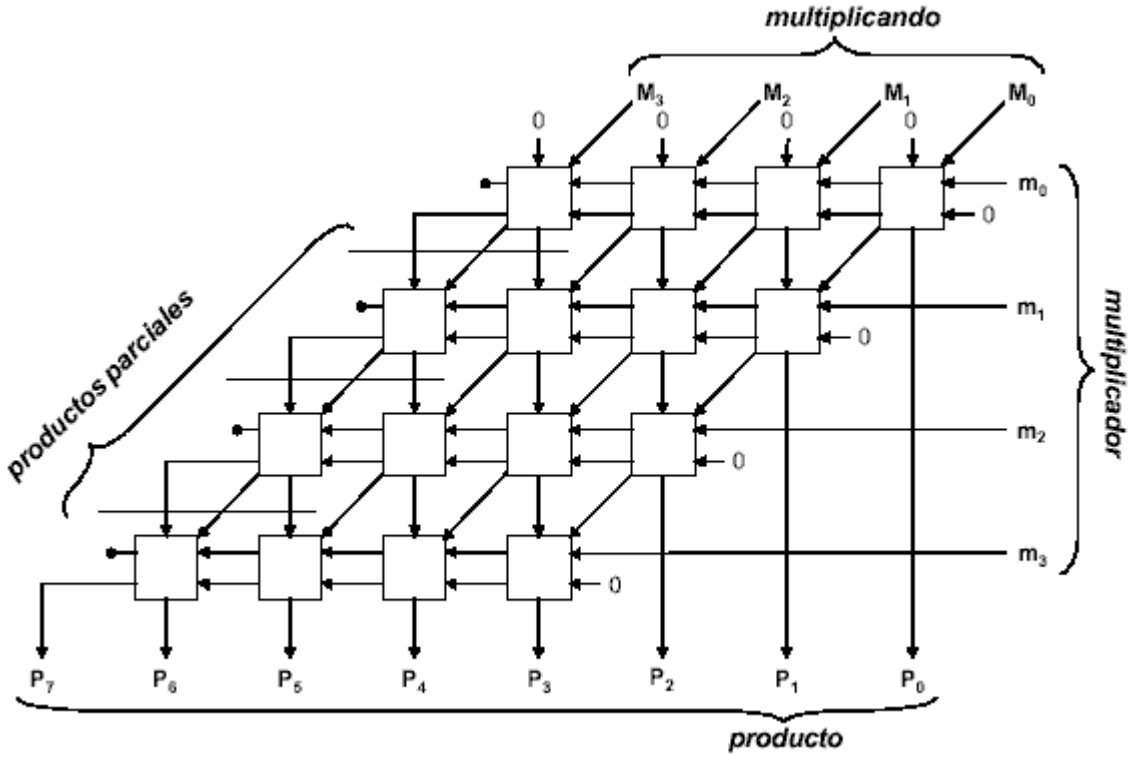



**Figura A-21** Diagrama de bloques de un multiplicador combinacional

## A.5 División de enteros sin signo

Referencias **[9]** [11]

Metodo tradicional de division:
Colocar uno en el cociente. Para la obtencion de los restos parciales hay que ir recorriendo el dividendo de izquierda a derecha. Si el resto parcial es mayor que el divisor, añadir un 1 en el cociente (a la derecha) y obtener el nuevo resto parcial. Si el resto parcial es menor que el divisor, añadir un 0 en el cociente (a la derecha) y ampliar el resto parcial con un bit más del dividendo. Para alinear correctamente los restos parciales y el divisor, en cada iteracion desplazarlo a la izquierda.

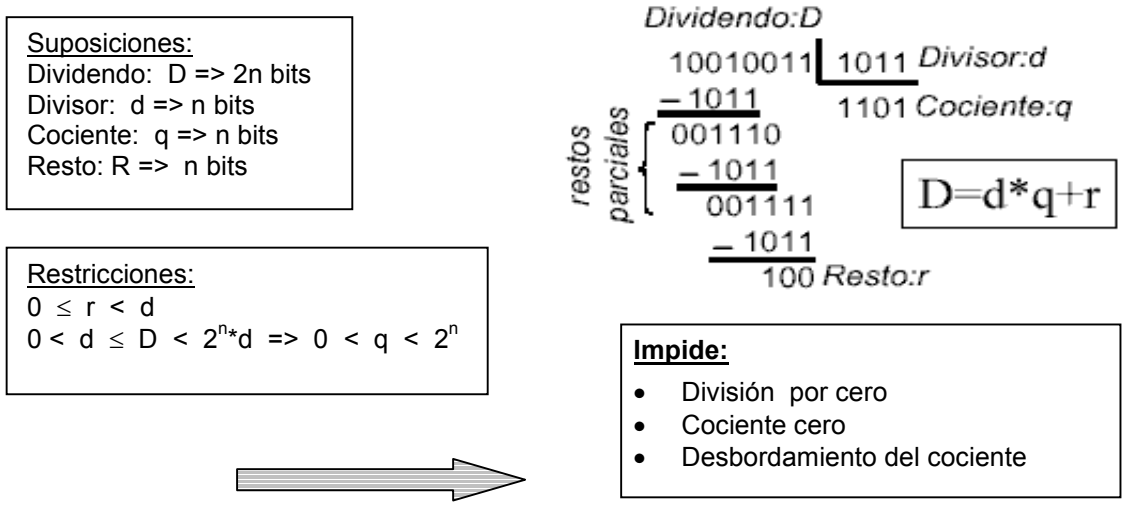



**Figura A-22** Ejemplo y requerimientos de la división de enteros sin signo

### A.5.1 Division de números enteros sin signo con restauracion:

Para efectuar la division se utilizan 3 registros: resto/dividendo, divisor y cociente.
Para tener alineados correctamente los restos parciales y el divisor, en cada iteracion hay que desplazar el divisor a la derecha. El cociente despues de cada iteracion hay que desplazarlo hacia la izquierda. El signo de la resta determinará si el resto parcial "cabe" en el divisor.

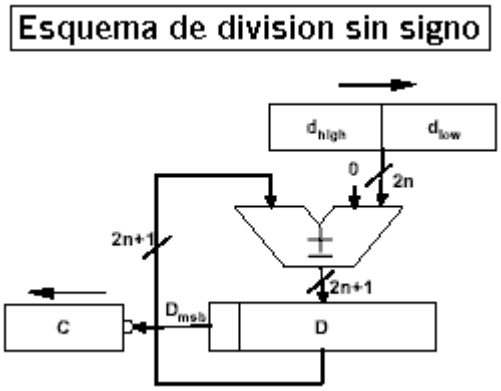



**Figura A-22** Esquema de la división de enteros sin signo con restauración

Los pasos son:

S0: Cargo (0, dividendo) en el registro D
Cargo el divisor en $d_{high}$ ; $d_{low} = 0$
C = 0

S1: D ← D - (0,d)

S2: Si $D_{msb} = 0$, se inserta un 1 en C y se desplaza a la izquierda
Si $D_{msb} = 1$, se inserta un 0 en C y se desplaza a la izquierda y D ← D + (0,d)
Desplazo D hacia la derecha
Si el loop S1 y S2 no se ha repetido n+1 veces, volver a S1

## A.5.2 Division de números enteros sin signo sin restauracion:

En este caso no tengo registro para el cociente, este se va formando según sea $D_{msb}$ y se va acumulando en $D_{low}$.

Los pasos son:

S0: Cargo (0, dividendo) en el registro D
Cargo el divisor en d

S1: Desplazo el registro D hacia la derecha

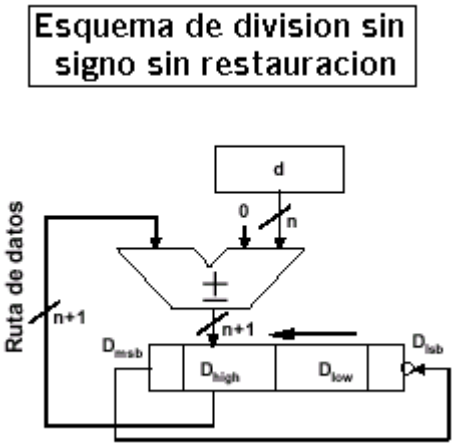
S2: $D_{high} \leftarrow D_{high}$ - (0,d)

S3: Si $D_{msb} = 0 \Rightarrow D_{high} \leftarrow 1$
Si $D_{msb} = 1 \Rightarrow D_{high} \leftarrow 0$

S4: Desplazar el registro D a la izquierda

S5: Si $D_{msb} = 0 \Rightarrow D_{high} \leftarrow D_{high}$ - (0,d)
Si $D_{msb} = 1 \Rightarrow D_{high} \leftarrow D_{high}$ + (0,d)
Si el loop S3-S5 no se ha repetido n-1 veces, volver a S3

S6: Si $D_{msb} = 0 \Rightarrow D_{lsb} \leftarrow 1$
Si $D_{msb} = 1 \Rightarrow D_{high} \leftarrow D_{high}$ + (0,d) ; $D_{lsb} \leftarrow 0$




| *tras* | *D* | *d* |
|---|---|---|
| *S0* | 010010011 | 1011 |
| *S1* | 100100110 | 1011 |
| *S2* | 001110110 | 1011 |
| *S3* | 001110111 | 1011 |
| *S4* | 011101110 | 1011 |
| *S5* | 000111110 | 1011 |
| *S3* | 000111111 | 1011 |
| *S4* | 001111110 | 1011 |
| *S5* | 111001110 | 1011 |
| *S3* | 111001110 | 1011 |
| *S4* | 110011100 | 1011 |
| *S5* | 001001100 | 1011 |
| *S6* | 001001101 | 1011 |

**Figura A-23** Esquema de la división de enteros sin signo sin restauración.

### A.5.3 División combinacional con restauración

La idea es que a través del bloque combinacional ir obteniendo los restos parciales de cada paso de la división, que entrarán al nuevo paso.

| | |
|---|---|
| B : bit del divisor | D : resultado de la comparación |
| A : bit del resto parcial | |
| C : carry in | S : bit del nuevo resto parcial |
| P : carry out | |

| Estamos suponiendo que | |
|---|---|
| $N = n_1\ n_2\ n_3\ n_4\ n_5\ n_6$ | Dividendo |
| $D = d_1\ d_2\ d_3$ | Divisor |
| $Q = q_1\ q_2\ q_3$ | Cociente |
| $R = 0\ 0\ 0\ r_4\ r_5\ r_6$ | Resto |
| $D > N$ | |

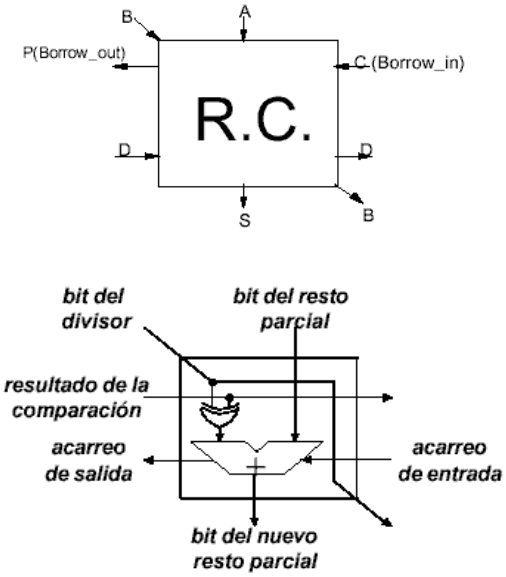



**Figura A-24** Esquema del bloque del divisor combinacional con restauración

Ecuaciones del Restador Controlado (R.C.) son:

$P = A'B + A'C + BC$
$S = AD + AB'C' + ABC + A'BC'D' + A'B'CD'$

De donde $S = A \oplus B \oplus C$ si $D = 0$
$S = A$ si $D = 1$

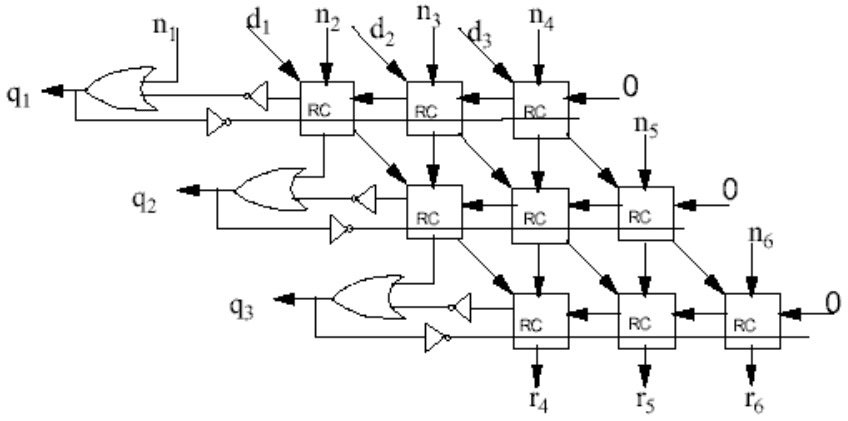



**Figura A-25** Esquema del divisor combinacional con restauración.

# B Correctores

Este apéndice se divide en cuatro secciones. En la primera hacemos una descripción de los correctores que usamos para solucionar problemas con los archivos generados por los compiladores (HCC, TMCC). Luego comentamos el funcionamiento del traductor de XNF a VHDL, las correcciones al código original y también como usar las extensiones que le hicimos. La tercer sección trata sobre las correcciones a las fuentes del Transmogrifier C, necesarias para no tener que usar correctores al trabajar con el, también mencionamos posibles mejoras al código para generar archivos más claros. Por último mostramos las fuentes de los correctores.

## B.1 Correctores

### B.1.1 LIBLOG

**Descripción del problema**
Cuando generamos un archivo de salida en VHDL con Handel C, éste declara una serie de componentes, como ser LIBAN2, LIBDFF, etc. y entre ellos se encuentra uno llamado LIBLOG2 (las librerías con la definición de estos componentes se pueden ver en el Apéndice F).
Los problemas con este último componente son dos, uno de ellos surge pues no esta correctamente declarado y contiene un error de sintaxis haciendo que los sintetizadores no compilen. El otro problema surge en la instanciación (o mapeo) dentro del cuerpo de la arquitectura pues se indica una etiqueta, el port map pero no el nombre del componente, que debería ser LIBLOG2.
Veamos un ejemplo de un código generado por el compilador donde destacamos los errores cometidos:

```
…
architecture NETLIST of matriz is
...
   component LIBLOG2
         port ( VDDLOG,GNDLOG : out std_ulogic;  ← El punto y coma
               );                                  esta demás.
...
begin
      CSupply :  [ ]  port map ( VDDLOG => VCC, GNDLOG => GND );
      …          ↑
                 Aquí falta el nombre del componente.
```

**Figura B-1** Declaración y mapeo de componente LIBLOG2 realizada por Handel C.

**Solución**
Los problemas son muy simples de solucionar, si bien se podría resolver manualmente, resulta tedioso estar retocando "a mano" cada vez las fuentes generadas y sobre todo en diseños grandes donde existe una larga lista de declaración de señales previa a la declaración del componente y de su instanciación.

Creamos un sencillo programa para el Awk[9] que soluciona estos problemas. Esencialmente lo que hace es buscar la declaración del componente LIBLOG2 y cuando lo encuentra setea una variable llamada "encontro". Al procesar la próxima línea elimina el último carácter (el punto y coma que sobra) y resetea la variable "encontro". Luego continua procesando el resto de las líneas en busca del string "CSupply" y una vez encontrado inserta luego del carácter ":" el string "LIBLOG2".

Las fuentes de este programa se pueden apreciar al final de éste apéndice (pág. 174).

## B.1.2 PUERTOS1

**Descripción del problema**
En las etapas iniciales del proyecto realizamos pruebas en el Handel C con su archivo de salida en VHDL y declarábamos las señales de I/O del circuito como CHANNELS, luego vimos otras formas mejores de trabajar.

Al compilar un programa en éstas condiciones se genera un código en VHDL que tiene una serie de problemas con relación a las señales TXRDY y RXRDY de los channels. Estos problemas son:

- En el cuerpo de la arquitectura se leen datos de señales que fueron declaradas como salidas de la entidad. En general los sintetizadores de VHDL no permiten utilizar las señales tipo OUT como entradas de otros componentes, hay que definir una señal interna y asignar ésta a la salida declarada.
- Utiliza en el cuerpo de la arquitectura señales que nunca son declaradas.

**Solución**
Creamos un programa para Awk que corrige estos problemas, en una primera fase se examina la declaración de entidad y se arman dos listas, de las señales tipo entrada y tipo salida, que contengan las palabras "TXRDY" y "RXRDY". Luego se arma otra lista de las señales que son declaradas en la arquitectura y que contengan estas mismas palabras.

Para el cuerpo de la arquitectura se examinan los nombres de todas las señales que intervienen (y que contengan las palabras antes mencionadas), se verifica que hayan sido declaradas en las listas generadas anteriormente y en caso de no aparecer son anotadas en una cuarta lista de variables para ser tratadas en la segunda fase.

Luego de terminada la etapa anterior se procede a escribir el archivo corregido declarando las señales faltantes y creando nuevas para trabajar con los puertos de salida.

[9] En particular utilizamos el GNU Awk 3.0.3

Las fuentes de este programa se encuentran al final de este apéndice (pág. 174). A continuación mostramos un ejemplo de un código con los problemas mencionados y la solución adoptada.

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
entity test3 is
  port (
    signal C_mp_TXRDY : in std_ulogic;
    signal C_mp_RXRDY : out std_ulogic;
    …
  );
end test3;
architecture NETLIST of test3 is
    signal C_res_TFER : std_ulogic;
    signal C_mp_TFER : std_ulogic;
    …
begin
    C0: LIBOR2 port map (Y => Opt_50,A => FINISH_OUT,B => C_res_TFER);
    C2: LIBOR2 port map (Y => Opt_51,A => C_mp_TFER,B => S_44);
    C3: LIBDFF port map (Q => C_res_TXRDY,CK => HCLK,D => Opt_51);
    C5: LIBDFF port map (Q => C_mp_RXRDY,CK => HCLK,D => Opt_52);
    C6: LIBAN2 port map (Y => C_mp_TFER,A => C_mp_RXRDY,B => C_mp_TXRDY);
    C7: LIBINV port map (Y => S_15,A => C_mp_TFER);
    C8: LIBAN2 port map (Y => C_res_TFER,A => C_res_RXRDY,B => C_res_TXRDY);
    …
```

**Figura B-2** Ejemplo de VHDL con errores generado por Handel C.

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
entity test3 is
  port (
    signal C_mp_TXRDY : in std_ulogic;
    signal C_mp_RXRDY : out std_ulogic;
    …
  );
end test3;
architecture NETLIST of test3 is
    signal C_res_TFER : std_ulogic;
    signal C_mp_TFER : std_ulogic;
    …
-- Señales generadas automaticamente por el corrector del codigo, version 1.0
    signal corrC_mp_RXRDY : std_ulogic;
    signal C_res_TXRDY : std_ulogic;
    signal C_res_RXRDY : std_ulogic;
begin
-- Asignaciones generadas automaticamente por el corrector del codigo, version 1.0
    C_mp_RXRDY <= corrC_mp_RXRDY;
    C0: LIBOR2 port map (Y => Opt_50,A => FINISH_OUT,B => C_res_TFER);
    C2: LIBOR2 port map (Y => Opt_51,A => C_mp_TFER,B => S_44);
    C3: LIBDFF port map (Q => C_res_TXRDY,CK => HCLK,D => Opt_51);
    C5: LIBDFF port map (Q => corrC_mp_RXRDY,CK => HCLK,D => Opt_52);
    C6: LIBAN2 port map (Y => C_mp_TFER,A => corrC_mp_RXRDY,B => C_mp_TXRDY);
    C7: LIBINV port map (Y => S_15,A => C_mp_TFER);
    C8: LIBAN2 port map (Y => C_res_TFER,A => C_res_RXRDY,B => C_res_TXRDY);
    ...
```

**Figura B-3** Ejemplo de VHDL generado por Handel C corregido con PUERTOS1.

## B.1.3 NOMBRES

**Descripción del problema**
Al compilar con el Transmogrifier C y luego utilizar el traductor de XNF a VHDL, se genera un problema con los nombres de las señales pues no cumplen la especificación de nombres de señal válida para VHDL. Hay señales que comienzan con números, otras contienen el signo de menos ("-") y otras con dos o más underscore ("_") consecutivos.

**Solución**
Generamos un simple programa para Awk para preprocesar el archivo XNF antes de utilizarlo con el traductor a VHDL y así corregir los nombres de las señales.

Este programa va examinando el archivo en XNF y verificando que los nombres de las señales sean válidos, en caso de no serlo se realizan los siguientes cambios:

- Nombres de señales que no comienzan con letras se les agrega una "a" delante.
- Nombres de señales que contengan por los menos un signo de menos ("-") se sustituyen por un underscore ("_")
- Nombres de señales que contengan dos o más underscores consecutivos se intercala a letra "a" entre ellos.

Las fuentes de este programa se encuentran al final de éste apéndice (pág. 175). A continuación mostraremos un ejemplo de la corrección de los nombres.

| Versión original | | Versión corregida |
|---|---|---|
| LCANET, 4 | | LCANET, 4 |
| PWR, 1, VCC | | PWR, 1, VCC |
| PWR, 0, GND | | PWR, 0, GND |
| PROG, tmcc, 3.1, "Tue Jun 18 12:34:09 2002" | | PROG, tmcc, 3.1, "Tue Jun 18 12:34:09 2002" |
| PART, 4003pc84 | | PART, 4003pc84 |
| SYM, FFin-0_1_0Running, BUF | → | SYM, FFin_0_1_0Running, BUF |
| PIN, I, I, VCC | | PIN, I, I, VCC |
| PIN, O, O, FFin-0_1_0Running | → | PIN, O, O, FFin_0_1_0Running |
| END | | END |
| SYM, 0_1_0Running, DFF | → | SYM, a0_1_0Running, DFF |
| PIN, D, I, FFin-0_1_0Running | → | PIN, D, I, FFin_0_1_0Running |
| PIN, C, I, CLK | | PIN, C, I, CLK |
| PIN, Q, O, 0_1_0Running | → | PIN, Q, O, a0_1_0Running |
| END | | END |
| SYM, 10T_SYM0_4_0__count, AND | → | SYM, a10T_SYM0_4_0_aount, AND |
| PIN, I2, I, 0_1_0Running,,INV | → | PIN, I2, I, a0_1_0Running, , INV |
| PIN, I1, I, 0_4_0__count,,INV | → | PIN, I1, I, a0_4_0_aount, , INV |
| PIN, I0, I, 0_5_0__entrada | → | PIN, I0, I, a0_5_0_antrada |
| PIN, O, O, 10T_0_4_0__count | → | PIN, O, O, a10T_0_4_0_aount |
| END | | END |
| … | | ... |

**Figura B-4** Ejemplo de corrección de nombres a XNF generado por Transmogrifier C.

## B.1.4 ANDS

**Descripción del problema**
Los archivos generados al compilar con el Transmogrifier C poseen un problema con la especificación de los nombres de los pines en el símbolo AND.
Los pines de un símbolo tipo OR, AND, BUF, etc., identifican el tipo con la letra "O" para indicar que es salida y con la letra "I" para indicar que es entrada y en el caso de existir más de una entrada la letra "I" es concatenada con un número (comenzando por 0) que identifica la entrada.
El compilador cuando genera los símbolos correspondientes a un AND de dos o más entradas no nombra correctamente el pin correspondiente a la primera, esto es en lugar de poner "I0" le pone "I".

**Solución**
Generamos un sencillo programa para Awk que corrige este problema. Se examina el archivo y se buscan los símbolos tipo AND, una vez encontrados se busca el PIN cuyo tipo sea declarado solamente con la letra "I" y ésta se sustituye por el string "I0".

Las fuentes de este programa se encuentran al final de este apéndice (pág. 175). A continuación mostramos una sección de código donde se realizaron las correcciones.

| Versión original | | Versión corregida |
|---|---|---|
| …<br>SYM, 8T_SYM0_9_3__Bit1, AND<br>PIN, I2, I, 0_4_L6init,,INV<br>PIN, I1, I, 0_9_3__Bit1<br>PIN, I, I, 0_16_L56looptop,,INV<br>PIN, O, O, 8T_0_9_3__Bit1<br>END<br>… | → | …<br>SYM, 8T_SYM0_9_3__Bit1, AND<br>PIN, I2, I, 0_4_L6init,,INV<br>PIN, I1, I, 0_9_3__Bit1<br>PIN, I0, I, 0_16_L56looptop,,INV<br>PIN, O, O, 8T_0_9_3__Bit1<br>END<br>... |

**Figura B-5** Ejemplo de corrección de pines a XNF generado por Transmogrifier C.

## B.1.5 RENAME2

**Descripción del problema**
Los archivos en XNF generados por el Handel C son procesados por el traductor a VHDL, es conveniente para que éste último reconozca correctamente los grupos de señales (o buses) que posean una nomenclatura estándar (respetada por el Transmogrifier C con fuentes modificadas).

Otro problema surge pues para algunas señales declaradas externas el compilador no respeta el nombre original dificultado los posteriores trabajos con el VHDL generado, como ser simulación, asignación de pines, etc.

Recordamos que en la especificación del XNF las señales externas son declaradas de la siguiente forma:

```
EXT, <nombre de señal>, <dirección>, <pin number>, <tipo>, LOC=<id. del pin para soft P&R>
```

Por ejemplo para una RAM externa de $(2^{16})$x8 llamada "mem_dat" se generan, entre otros, estos nombres de señales:

```
EXT, PADH_GND_9343, O, , LOC=mem_dat_Addr<7>
EXT, PADH_GND_9344, O, , LOC=mem_dat_Addr<8>
EXT, PADH_mem_dat_Data<1>_9353, B, , LOC=mem_dat_Data<1>
EXT, PADH_mem_dat_Data<2>_9354, B, , LOC=mem_dat_Data<2>
EXT, PADH_mem_dat_wb_481_9384, O, , LOC=mem_dat_wb
```

Aquí en particular definimos convenientemente el nombre del pin para cada señal como se puede observar en la etiqueta LOC (location).

Otro ejemplo que ocurre para puertos de 32 bits de ancho ocurre:

```
EXT, PADH_C_ram_salida_din_0_9257, O, , LOC=Z68_port_ram_salida_din_32
EXT, PADH_C_ram_salida_din_1_9258, O, , LOC=Z69_port_ram_salida_din_32
```

Y por último un ejemplo para un puerto de ancho 1 bit:

```
EXT, PADH_C_ram_fila2_din_TXRDY_9256, O, , LOC=Z67_port_ram_fila1_we_1
```

**Solución**

La solución a este problema es preprocesar el XNF generado con el compilador antes de utilizar el traductor a VHDL con el programa que describiremos a continuación. Además para el uso de RAM (y para algunos casos también de puertos) hay que ajustar las constantes de especificación en el código C antes de compilarlo para así poder definir convenientemente el LOC.

Para realizar el preprocesamiento que comentamos generamos un programa para Awk que examina el archivo en busca de los símbolos EXT, y una vez encontrados realizamos las siguientes consideraciones:

a) Si el nombre de la señal comienza con "PADH_C_" analizamos el resto del nombre partiéndolo en substring según el delimitador underscore.

a.1) Si el penúltimo substring es TXRDY entonces el nombre de la señal se obtendrá de la etiqueta LOC. Aquí también separamos el contenido de la etiqueta LOC en substring con el mismo delimitador y el nombre lo formamos con los substring del tercero al penúltimo. Ejemplo:

| **Versión original** |
|---|
| EXT, PADH_C_ram_fila2_din_TXRDY_9256, O, ,LOC=Z67_port_***ram_fila1_we***_1 |
| **Versión corregida** |
| EXT, ***ram_fila1_we***, O, , LOC=Z67_port_ram_fila1_we_1 |

a.2) Si no ocurre lo anterior entonces separamos en substring el contenido de la etiqueta LOC, también con el mismo delimitador, para determinar el ancho en bits de la señal que es indicado en el último substring. Formamos el nuevo nombre de la señal utilizando del tercero hasta el antepenúltimo substring del nombre original y luego si el ancho es mayor que uno agregamos entre "<" y ">" la posición dentro del bus que es indicada en el penúltimo substring. Ejemplo:

| **Versión original** |
|---|
| EXT, PADH_C_*ram_salida_din_1*_9258, O, , LOC=Z69_port_ram_salida_din_32 |
| **Versión corregida** |
| EXT, *ram_salida_din<1>*, O, , LOC=Z69_port_ram_salida_din_32 |

b) Si el anterior no se cumple y el nombre comienza con "PADH_" el nuevo nombre de la señal es el contenido de la etiqueta LOC. Ejemplo:

| **Versión original** |
|---|
| EXT, PADH_GND_9344, O, , LOC=*mem_dat_Addr<8>* |
| **Versión corregida** |
| EXT, *mem_dat_Addr<8>*, O, , LOC=mem_dat_Addr<8> |

Todos los renombres de las señales son guardados en una lista para luego reemplazarlos en el resto de los símbolos.

Las fuentes del programa se encuentran al final de este apéndice (pág. 176).

## B.1.6 UNDERSCORE

**Descripción del problema**
Un problema similar al mencionado con el nombre de las señales para el Transmogrifier C cuando se hablo del corrector NOMBRES, surge al utilizar Handel C con variables locales.

Este compilador en caso de que coincidan los nombres de las variables locales con otras variables de otras funciones o del programa principal, les agrega un underscore ("_") al comienzo del nombre para diferenciarla del resto, y agregará tantos como sean necesarios.

**Solución**
Si bien se podría utilizar el corrector NOMBRES, decidimos realizar una nueva implementación que sea especifica para este problema. El programa busca las ocurrencias de dos o más underscores consecutivos y los reemplaza por un underscore mas la primer letra que esta enseguida de los underscores mas el numero de underscores.

Las fuentes se encuentran al final de este apéndice (pág. 176). A continuación un ejemplo de las correcciones realizadas.

| Versión original | | Versión corregida |
|---|---|---|
| ... | | … |
| SYM, Q__a_0, DFF | → | SYM, Q_a2a_0, DFF |
| PIN, Q, O, Q__a_0 | → | PIN, Q, O, Q_a2a_0 |
| PIN, D, I, D__a_0 | → | PIN, D, I, D_a2a_0 |
| PIN, CE, I, CE____a | → | PIN, CE, I, CE_a4a |
| PIN, C, I, HCLK | | PIN, C, I, HCLK |
| END | | END |

**Figura B-6** Ejemplo de corrección de nombres a XNF generado por Handel C.

## B.1.7 BUFT

**Descripción del problema**
Cuando trabajamos con memorias externas, el Handel C genera automáticamente buffers triestado para las señales de datos de dicha RAM. El problema se genera pues la señal de habilitación de dichos buffers es poco convencional, están habilitados en todo momento y se deshabilitan al realizar lecturas. Esto complica la inserción de los circuitos en diseños basados en buses de recursos compartidos.

**Solución**
La forma de solucionar este problema es cambiar de señal de habilitación de los buffers, y optamos por la utilización de la señal de escritura (wb) de la memoria correspondiente.
Para ello realizamos un programa en Awk que se aplica a continuación del corrector llamado RENAME2 y realiza el trabajo en 2 etapas. En la primer etapa arma una lista de todas las señales de escritura para todas las RAM definidas, para ello busca en los símbolos OBUF (buffer de salida) que posean un nombre que termine con "_wb" y una vez encontrados registra el nombre de la señal de entrada a dicho buffer en una lista tipo hash que utiliza como índice el nombre de la RAM.

En la segunda etapa se comienza a generar el archivo de salida y se verifica la existencia de los símbolos OBUFT (buffer triestado), si se encuentra uno de ellos se busca el nombre de la RAM en la lista confeccionada anteriormente y sustituye el nombre de la señal declarado en el PIN llamado "T".
Las fuentes de este corrector se encuentran al final de éste apéndice (pág. 177). A continuación veamos un ejemplo de la corrección realizada.

| Versión Original. | | Versión Corregida |
|---|---|---|
| EXT, memA_Data<1>, B, , LOC=memA_Data<1> | | EXT, memA_Data<1>, B, , LOC=memA_Data<1> |
| SYM, memA_Data<2>, OBUFT, FAST | | SYM, memA_Data<2>, OBUFT, FAST |
| PIN, T, I, C_sal_TXRDY | → | PIN, T, I, memA_wb_34 |
| PIN, I, I, C_memA_Dout_2 | | PIN, I, I, C_memA_Dout_2 |
| PIN, O, O, memA_Data<2> | | PIN, O, O, memA_Data<2> |
| END | | END |
| SYM, memA_wb, OBUF,FAST | | SYM, memA_wb, OBUF,FAST |
| PIN, I, I, memA_wb_34 | | PIN, I, I, memA_wb_34 |
| PIN, O, O, memA_wb | | PIN, O, O, memA_wb |
| END | | END |
| EXT, memA_wb, O, , LOC=memA_wb | | EXT, memA_wb, O, , LOC=memA_wb |

**Figura B-7** Ejemplo de corrección de buffer triestado a XNF generado por Handel C.

# B.2 Traductor de XNF a VHDL

## B.2.1 Introducción

El formato de salida del TMCC es un netlist descrito en XNF, HCC ofrece la posibilidad de generar VHDL. Decidimos usar un traductor[10] de XNF a VHDL porque encontramos algunas desventajas/problemas en trabajar con los archivos XNF, estas son:

- El software Max+PlusII versión 10.1 se cuelga al intentar compilar dichos archivos.
- Este formato está siendo discontinuado, futuras versiones de ISE no lo incluirán. [11]
- Para chips de Xilinx hay que trabajar con 2 programas simultáneamente, el Design Manager genera archivos de simulación y configuración a partir del archivo .xnf y el Logic Simulator (integrado en el Project Manager) simula usando el archivo generado por el Design Manager.

## B.2.2 Funcionamiento

El traductor esta realizado en PERL, aquí describiremos la versión original luego mencionaremos los cambios realizados para su correcto funcionamiento.
El programa recorre línea por línea el archivo XNF y genera dos listas de señales (señales EXT y señales no EXT). Las señales EXT son las declaradas externas en el XNF.

Luego crea un archivo llamado *<nombre del archivo xnf>_netlist.vhd*, después imprime en ese archivo en el siguiente orden:

- *architecture netlist of <nombre del archivo xnf>*
- lista de señales no externas, cada línea es una de las siguientes posibilidades (según xxx sea un vector o no)

  *signal xxx: std_logic;*

  *signal xxx: std_logic_vector( N downto 0);*
- *begin*
- compuertas (AND, OR), flip-flop (DFF), buffers (INV, BUFGP, IBUF, OBUF, OBUFT)
- *end netlist;*

En particular para imprimir el cuerpo de la arquitectura lee nuevamente, línea por línea, el archivo XNF. Cuando encuentra alguno de los símbolos admitidos, imprime como se muestra en la Tabla B-1.

---

[10] Se puede bajar de http://www.arl.wustl.edu/~rex/CodeHenge/xnftovhdl.pl

[11] Según Xilinx en "**Answer Record # 12980: 4.2i ISE - Project Navigator does not allow me to bring in an XNF file; can I use an XNF file with the 4.2i ISE tools?"** accesible desde http://www.xilinx.com/company/search.htm

**Tabla B-1** Código VHDL impreso según símbolo especificado en netlist XNF.

| Símbolo | Se imprime |
|---|---|
| Una compuerta<br>(por ejemplo and de tres entradas) | z<= a and b and c; |
| Un inversor | z<=not a; |
| Un buffer tipo BUFGP, OBUF e IBUF | z<=a; |
| Un buffer con triestado (OBUFT) | z<=a when t='0' else 'z'; |
| Un flip-flop (DFF) | process (<reloj>)<br>begin<br>  if rising_edge(<reloj>) then<br>  <q><=<d>;<br>  end if;<br>end process; |

Otros símbolos son también reconocidos pero los compiladores que evaluamos no los utilizan y por tanto no verificamos si el traductor trabaja correctamente con ellos.

El traductor reconoce grupos (o bus) de señales, para ello busca en el nombre de la señal la ocurrencia de un número (N) entre los caracteres "<" y ">", va recordando el mayor "N" para dicho nombre de señal que es utilizado para determinar el ancho de los buses. Los compiladores evaluados en este proyecto no generan los grupos de señales con esta notación para solucionar el problema preprocesamos el archivo antes de ingresarlo al traductor, para ampliar sobre el procesamiento ver el corrector RENAME2 (punto B.1.5).

### B.2.3 Errores

El traductor en su versión original tiene varios errores y cosas que faltan:

- No imprime la declaración de entidad aunque genera la lista de señales externas.
- El nombre del archivo es diferente al nombre de la entidad (que supone al declarar la arquitectura), esto da problemas con algunos sintetizadores por ejemplo con el M+PII.
- Las compuertas OR las imprime como si fueran ANDs.
- Pese a que reconoce los flip-flop con señal de CE (clock enable), cuando imprime el proceso lo hace como si no estuviera (imprime un DFF común).
- No reconoce XOR, BUF, BUFG.

Estos errores y carencias fueron corregidos por nosotros en la versión final que utilizamos en este proyecto.

## B.2.4 Extensiones

Reset Global

Como se pudo observar anteriormente los flip-flop inferidos no poseen una señal de reset esto hace imposible poder llevar todos los flip-flop de los circuitos generados a su estado inicial, por tanto agregamos una señal de reset global para cada uno de estos.

Módulos de multiplicación

Como el TMCC no tiene incorporada la multiplicación de enteros decidimos agregarle funcionalidad al traductor para que sea posible usar la megafunción "lpm_mult" de Altera o el operador "*" de Xilinx.

Las señales entre las cuales queremos insertar un multiplicador deben ser declaradas como puertos del circuito, el resultado debe ser una entrada al circuito y los multiplicandos salidas del mismo. Para que el traductor reconozca que se desea usar un multiplicador y haga los cambios necesarios se debe usar la siguiente nomenclatura con las señales "mul_<id_mul>_<señal>" donde "id_mul" es un identificador del multiplicador, pues puede haber más de uno en un diseño, y "señal" puede ser "mr", "mn" o "res". Donde mul_<id_mul>_mn y mul_<id_mul>_mr son las señales que se desean multiplicar y mul_<id_mul>_res corresponde al resultado.

Si se quiere compilar un diseño para chips de Altera hay que pasarle el parámetro "Alt" al traductor en su línea de comandos, en este caso al reconocer un multiplicador este agrega y mapea el componente lpm_mult. Esta componente la configuramos para realizar productos con signo solamente, si se desea utilizar productos sin signo puede cambiar "a mano" las fuentes de VHDL generadas o sino declarar las variables "mul_<id_mul>_<señal>" con un bit más del necesario y forzar el más significativo a 0.

Si se desea compilar para chips de Xilinx no hay que agregarle ningún parámetro al traductor, en este caso al reconocer un multiplicador no se genera ningún componente y, en VHDL, se conectan las señales de la siguiente forma:

mul_<id_mul>_res <= mul_<id_mul>_mn * mul_<id_mul>_mr;

Por defecto se agrega el package "std_logic_signed" (librería IEEE), por lo que si se desea realizar multiplicación sin signo se debe modificar ésto (cuarta línea del archivo VHDL generado) y usar el package "std_logic_unsigned". Otra solución es al igual que para el caso de Altera, agrandar las señales en un bit. Cabe destacar que en un mismo archivo no se pueden usar multiplicadores diferentes (con signo y sin signo) esta es una limitación del Foundation.
El software Xilinx Foundation reconoce el operador producto "*" e implementa el multiplicador como módulos optimizados específicos a la arquitectura basados en el ancho de los operandos.

Al trabajar desde C hay que tener presente que lo que se inserta es un multiplicador combinatorio. Para mostrar como trabajar en C y lo que genera el traductor vemos un ejemplo en el que se desea multiplicar "a" por "b" (8 bits c/u) y guardar el resultado en "z".

```
main(){
#pragma intbits 8
        int mul_mult1_mn, mul_mult1_mr, a, b;
#pragma intbits 16
        int mul_mult1_res, z;

        inputport(a); inputport(b); outputport(z);
        inputport(mul_mult1_res); outputport(mul_mult1_mr);
        outputport(mul_mult1_mn);

        portflags(a, 0x3); portflags(b, 0x3); // port_registered
        portflags(mul_mult1_mr,0x1); // port_pin
        portflags(mul_mult1_mn,0x1);

        while(1){
                mul_mult1_mr=a;
                mul_mult1_mn=b;
                z=mul_mult1_res;
        };
}
```

**Figura B-8** Ejemplo de uso de producto para Transmogrifier C.

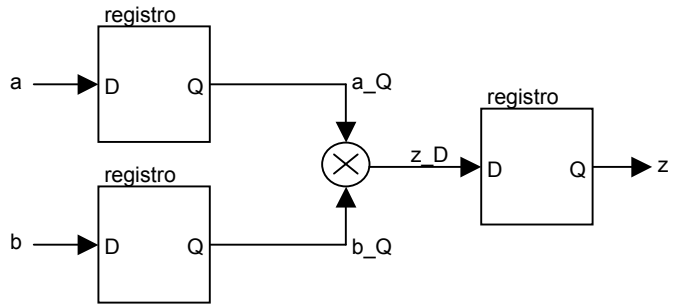



**Figura B-9** Simplificación del circuito generado para ejemplo de uso de producto.

Compilación:

```
tmcc –S –c multip.c myclock.xnf
x2v.pl multip.xnf Alt      -- En caso de compilar para
                              Xilinx no hay que agregar "Alt".
```

En la Figura B-10 podemos ver una forma resumida del archivo VHDL (multip.vhd) generado por el traductor.

```
entity multip is
...          // declaraciones de puertos
end multip;
architecture netlist of multip is
…          // declaraciones de señales
COMPONENT lpm_mult
             GENERIC (
                          LPM_WIDTHA: POSITIVE;
                          LPM_WIDTHB: POSITIVE;
                          LPM_WIDTHS: NATURAL:=1;
                          LPM_REPRESENTATION: STRING := "SIGNED";
                          LPM_WIDTHP: POSITIVE
              );
              PORT(
                       dataa:IN STD_LOGIC_VECTOR(LPM_WIDTHA-1 DOWNTO 0);
                       datab:IN STD_LOGIC_VECTOR(LPM_WIDTHB-1 DOWNTO 0);
                       result:OUT STD_LOGIC_VECTOR(LPM_WIDTHP-1 DOWNTO 0)
               );
END COMPONENT;
begin
  mult1:  lpm_mult
              generic map(LPM_WIDTHA=>8,
                            LPM_WIDTHB=>8,
                            LPM_WIDTHP=>16)
              port map(dataa=>mul_mult1_mn,
                          datab=>mul_mult1_mr,
                          result=>mul_mult1_res);

            mul_mult1_mr<= a_Q;
            mul_mult1_mn<=b_Q;
            z_D<=mul_mult1_res;
            process (CLK, globalReset)
            begin
                    if globalReset='1' then
                              a_Q <= "00000000";
                              b_Q <= "00000000";
                              z_D <= "0000000000000000";
                    elsif rising_edge(CLK) then
                              a_Q <= a;
                              b_Q <= b;
                              z_D <= mul_m1_res;
                    end if;
            end process;
end netlist;
```

**Figura B-10** Código VHDL generado para el ejemplo de uso de producto.

Aunque esta funcionalidad fue pensada para usarla con el Transmogrifier C también se puede usar con Handel C. La forma de trabajar es la misma así como el circuito generado.

Veamos, en la Figura B-11, como queda el ejemplo anterior al generarlo para Handel C, también realizamos el mismo circuito pero usando el operador * de Handel C.

**HCC con lpm_mult**

```
const spec harp = {
        fpga_type = "Xilinx4000",
        clock_pad = "P160"};
void main( target = harp,
        port (in) a:8,
        port (in) b:8,
        port (out) z:16,
        port (in) mul_m1_res:16,
        port (out) mul_m1_mn:8,
        port (out) mul_m1_mr:8)
{
        int aa, bb: 8;
        int c: 16;

        while(1) par{
                        a ? aa;
                        b ? bb;
                        mul_m1_mr ! aa;
                        mul_m1_mn ! bb;
                        mul_m1_res ? c;
                        z ! c;
                };
}
```

**HCC con operador producto nativo**

```
const spec harp = {
        fpga_type = "Xilinx4000",
        clock_pad = "P160"};
void main( target = harp,
        port (in) a:8,
        port (in) b:8,
        port (out) z:16)
{
        int aa,bb:8;
        int c:16;

        while(1) par{
                        a ? aa;
                        b ? bb;
                        c=aa*bb;
                        z ! c;
                };
}
```

**Figura B-11** Ejemplo de uso de producto para Handel C

Recursos ocupados en los ejemplos:
Chip 10K20RC240-4

**Tabla B-2** Comparativa de ejemplos para HCC y TMCC

| | **LC** | **Frecuencia Máx.** [ MHz ] |
|---|---|---|
| Transmogrifier C | 234 (20%) | 14.85 |
| HandelC lpm_mult | 256 (22%) | 13.71 |
| HandelC Operador prod. | 233 (20%) | 10.14 |

Podemos ver la conveniencia de usar la megafunción lpm_mult frente al operador nativo de Handel C ya que obtenemos circuitos de mayor velocidad con prácticamente la misma cantidad de celdas lógicas ocupadas.

Módulos de memoria interna

Se modificó el traductor con el objetivo de poder usar celdas de memoria al trabajar con RAM/ROM internas en los diseños, tanto del TMCC como del HCC. Cuando se desee usar dichas memorias lo que hay que hacer es, en el código C, trabajar con ellas como si fueran externas o sea las señales desde y hacia las RAMs deben ser entradas y salidas del circuito. Para que el traductor reconozca que se desea usar memoria interna y haga los cambios necesarios, hay que usar una nomenclatura especial para dichas señales que depende del tipo de memoria a utilizar. Los tipos disponibles son:

- lpm_ram_dq de Altera
- lpm_rom de Altera
- módulos sync_ram para Xilinx

Se debe indicar desde la línea de comandos con el parámetro “Alt” si se desea implementar memorias para chips de Altera o nada si se desea implementar para Xilinx.

Lpm_ram_dq

Elegimos esta megafunción (y no lpm_ram_io, lpm_ram_dp o csdpram) para implementar la RAM porque es más usual. Además la usamos con entrada síncrona ya que así es más fácil para escribir de forma consecutiva, por ejemplo se puede usar un contador para las direcciones, en cada pulso de reloj se carga un dato y se incrementa el contador. No hay que generar pulsos en WE para cada dato, se deja habilitada mientras dura la carga.

La notación de las señales es ***ram_xxx_yyy*** donde:

- xxx es el identificador de la RAM, todas las señales de una RAM deben usar el mismo identificador. No se debe usar el carácter "_" en dicho identificador.
- yyy identifica la señal de la RAM, puede ser: dir, din, dot, we que corresponden con : direcciones, dato in, dato out y write enable

Cuando el traductor encuentra alguna RAM, agrega el componente lpm_ram_dq a la arquitectura y luego mapea el componente ( para cada RAM a utilizar) como vemos a continuación:

```
<id_mem> : lpm_ram_dq
                generic map(    LPM_WIDTH=><data_width>,
                                LPM_WIDTHAD=><add_width>)
                port map(       data=>ram_<id_mem>_din,
                                address=>ram_<id_mem>_dir,
                                 we=>ram_<id_mem>_we,
                                 inclock=>hclk,
                                 q=>ram_<id_mem>_dot);
```

**Figura B-12** Mapeado en VHDL de componente lpm_ram_dq.

Lpm_rom

En este caso la notación de las señales debe ser ***rom_xxx_yyy*** donde:

- xxx es el identificador de la ROM, todas las señales de una ROM deben usar el mismo identificador. No se debe usar el carácter "_" en dicho identificador.
- yyy identifica la señal de la ROM, puede ser: dir, dot que corresponden con: direcciones y dato out.

Cuando el traductor encuentra alguna ROM, agrega el componente lpm_rom a la arquitectura y luego mapea el componente (para cada ROM a utilizar) como vemos a continuación:

```
<id_mem> : lpm_rom
                generic map(    LPM_WIDTH=><q_width>,
                                LPM_FILE=>rom_<id_mem>.mif
                                LPM_WIDTHAD=><addwidth>)
                port map(       address=>rom_<id_mem>_dir,
                                q=>rom_<id_mem>_dot);
```

**Figura B-13** Mapeado en VHDL de componente lpm_rom.

El diseñador debe crear además el archivo para la inicialización de la memoria ROM con el nombre rom_<id_mem>.mif, como puede aprecia en el Apéndice C Figura C-8.

Sync_ram

Se eligió este módulo (y no ram o dp_ram) por las mismas razones que se eligió la lpm_ram_dq con entradas síncronas.
La notación de las señales en este caso es ***ram_www_xxx_yyy*** donde:

- www identifica el tipo de RAM para el caso en que se quieran usar varias memorias y no todas iguales. Esto para la memoria de Altera no es necesario pues cada vez que se mapea el componente hay que especificar (lo hace el traductor) el valor de los genéricos LPM_WIDTH y LPM_WIDTHAD.
- xxx es el identificador de la RAM, todas las señales de una RAM deben usar el mismo identificador. No se debe usar el carácter "_" en dicho identificador.
- yyy identifica la señal de la RAM, puede ser: dir, din, dot, we que corresponden con : direcciones, dato in, dato out y write enable

Para cada tipo de RAM el traductor va a declarar un componente. Para asignar los bits de datos y direcciones el traductor se queda con el valor más alto de ram_tipo_mem_dir<n> y ram_tipo_mem_dot<n>. Luego mapea el componente para cada RAM de ese tipo encontrada:

```
mem : www_ram
        port map (      DI=>ram_tipo_mem_din,
                        A=>ram_tipo_mem_dir,
                        WR_EN=>ram_tipo_mem_we,
                        WR_CLK=>hclk,
                        DO=>ram_tipo_mem_dot);
```

**Figura B-14** Mapeado en VHDL de modulo sync_ram.

Este componente (www_ram) hay que crearlo usando la herramienta LOGIBLOX del paquete Foundation (o ISE).

# B.3 Correcciones a las fuentes del Transmogrifier C

## B.3.1 Introducción

Al trabajar con el TMCC encontramos algunos problemas de notación del archivo XNF generado. Cuando encontrábamos un problema la primer solución fue crear archivos que procesaran el XNF y corrigieran ese problema en particular. Luego investigamos las fuentes del TMCC para ver si podíamos encontrar de donde surgía el problema. Afortunadamente la mayoría de los cambios hay que hacerlos en el mismo archivo, output_xilinx.c que es el que genera los XNF.

Al estudiar las fuentes para solucionar los problemas también se nos ocurrieron algunos cambios para mejorar el trabajo en los archivos de salida con los grupos de señales (simulación, visualización, etc.).

## B.3.2 Problemas y soluciones

**1)** El símbolo AND no se imprime correctamente, el TMCC genera lo siguiente:

```
SYM, nombre_simbolo, AND
PIN, I2, I, entrada2
PIN, I1, I, entrada1
PIN, I, I, entrada0        ← Incorrecto, debería ser "PIN, I0, I, entrada0"
PIN, O, O, salida
END
```

**Figura B-15** Ejemplo de símbolo AND mal generado por Transmogrifier C

El problema aparece cuando se trabaja directamente con el archivo XNF, las herramientas de síntesis no aceptan esta notación. Si se usa el traductor que vimos para pasar de XNF a VHDL este problema desaparece.

Para solucionarlo hay que modificar el procedimiento "printAND( )" del archivo "output_xilinx.c". La versión modificada es la siguiente, donde resaltamos las líneas afectadas:

```
Static
printAND(i, table, count, b)
     int i;
     QMtab table[ ];
     int count;
     struct bit *b;
{
   int bitCnt, j, bitCnt_aux;
   struct bitlist *bl;
   bitCnt = QMtermBits(table[i], count+1);
   bitCnt_aux= bitCnt;          // Agregado
   fprintf (outputfile, "SYM, a%dT_SYM%s, %s\n", i,
            bitname(b), (bitCnt == 1) ? "BUF" : "AND");
   for(bl = b->primaries, j=count; bl; bl=bl->next, j--) {
     if(!(table[i].dc & (1<<j))) {
        if(bitCnt == 1)
            fprintf (outputfile, "PIN, I%s, I, a%s%s\n",
               (bitCnt_aux==1) ? "" : "0",bitname(bl->bit),
               (table[i].value & (1<<j)) ? "" :",,INV");
        else
            fprintf (outputfile, "PIN, I%d, I, a%s%s\n", --bitCnt,
                  bitname(bl->bit),
                  (table[i].value & (1<<j)) ? "" : ",,INV");
     }
   }
   fprintf (outputfile, "PIN, O, O, a%dT_%s\n", i, bitname(b));
   fprintf (outputfile, "END\n");
}
```

**Figura B-16** Modificación a procedimiento del módulo output_xilinx.c del Transmogrifier C

**2)** El nombre de algunos símbolos y variables contiene el carácter "-" (guión), esto no lo aceptan las herramientas de síntesis.
Para solucionar esto lo que hay que hacer es buscar en el archivo "output_xilinx.c" las secuencias "%s-" y "FFin-" y reemplazarlas por "%s_" y "FFin_" . El detalle de las ocurrencias es el siguiente:

| | | |
|---|---|---|
| %s- | 7 ocurrencias en las líneas: | 123, 138, 160, 170, 183, 202, 208 |
| FFin- | 11 ocurrencias en las líneas: | 125, 129, 140, 143, 226, 263, 274 |
| | | 303, 338, 409, 434 |

**3)** El nombre de algunas variables contiene el carácter doble "__" (doble underscore), esto tampoco lo aceptan las herramientas de síntesis.
Para solucionar esto uno de los cambios que hay que hacer es modificar el procedimiento "bitname( )" del archivo "tmcc.y". En la siguiente figura se resaltaron las líneas donde se realizaron los cambios.

```
char *
bitname(b)
    struct bit *b;
    {
    if(b->name)
        return(b->name);
    b->name = (char *) malloc(MAXNAMELEN);
    if(b->variable->width == 1) {
        if(b->flags & SYM_KEEPSNAME)
            strncpy(b->name, b->variable->name+1, MAXNAMELEN);
        else
            sprintf(b->name, "%d_%d_%s", thread,
                b->variable->lineno, b->variable->name+1);
        }
        else {
            sprintf(b->name, "%d_%d_%d_%s", thread,
                b->variable->lineno,  b->bitnumber,
                b->variable->name+1);
        }
        return(b->name);
}
```

**Figura B-17** Modificación a procedimiento del módulo tmcc.y del Transmogrifier C

Además hay que cambiar la siguiente línea del procedimiento init( ).

| ANTES | DESPUES |
|---|---|
| ...<br>vcc = findvariable("VCC", MUSTNOTEXIST, 1); | ...<br>vcc = findvariable("_VCC", MUSTNOTEXIST, 1); |

También hay que cambiar unas líneas en el archivo "output_xilinx.c":

```
        .....
        for(n=0; n<nbits; n++) {   // línea 103
                b = &bits[n];
                if(b->variable && !strcmp(b->variable->name, "_VCC"))   continue;
        .....
```

**4)** El nombre de algunas variables comienza con un digito, esto tampoco lo aceptan las herramientas de síntesis de VHDL. Para solucionar esto en cada línea en que se imprime el nombre de alguna variable o símbolo, que pueda comenzar con un digito, le agregamos una letra "a" al nombre. Para ver en que casos es necesario hacer el cambio presentamos unos ejemplos:

```
OUT "SYM, a%s, IBUF\n", bitname(b));   /* OUT es un macro que se sustituye
                                          por fprintf(ouputfile,  */
OUT "SYM, a%s_IBUF, IBUF\n", bitname(b)); /* %s se sustituye por valor de
                                             bitname(b),  por lo que es ahí donde
                                             debemos agregar la letra "a". */
OUT "PIN, O, O, a%s\n", bitname(b));
OUT "PIN, Q, O, a%s\n", bitname(b));
OUT "PIN, I, I, a%s\n",bitname(b->primaries->bit));
fprintf (outputfile, "PIN, I%d, I, a%s\n",  count--, bitname(bl->bit));
fprintf (outputfile, "PIN, I, I, a%dT_%s\n",i, bitname(b));
```

**Figura B-18** Ejemplos de cambios en el módulo output_xilinx.c del Transmogrifier C

También presentamos unos ejemplos de casos en que no es necesario agregarle la letra "a" a los nombres de variables.

```
fprintf(outputfile, "EXT, %s, %s,, LOC=%s\n", extname, type, pin);  /* las variables externas
                                                                       no tienen problema */
OUT "PIN, O, O, FFin_%s\n", bitname(b));  // El nombre ya comienza con letra ("F")
OUT "PIN, I, I, out%s\n", bitname(b));    // El nombre ya comienza con letra ("o")
```

**Figura B-19** Ejemplos donde no se hacen cambios al módulo output_xilinx.c del Transmogrifier C

En total hay que hacer estos cambios en 31 líneas, los números de éstas en el archivo original son: 114, 116, 123, 128, 131, 138, 142, 145, 151, 153, 160, 161, 170, 171, 179, 181, 183, 184, 202, 208, 229, 235, 267, 273, 277, 282, 305, 313, 371, 475, 481.

Para que estos cambios no afecten a la señal VCC hay que hacer las siguientes modificaciones:

```
...
if(count == 0){   // línea 234 en el original
    if (!strcmp(bitname(b->primaries->bit), "VCC")){    // Agregado
        OUT "PIN, I, I, %s\n", bitname(b->primaries->bit));
    }
    else{    // Agregado
        OUT "PIN, I, I, a%s\n", bitname(b->primaries->bit));   // Agregado
    }
}
else  OUT "PIN, I, I, GND\n");
```

**Figura B-20** Modificación a procedimiento del módulo output_xilinx.c del Transmogrifier C

### B.3.3 Mejoras al código

La especificación del XNF dice que para identificar a una señal como parte de un bus hay que usar la siguiente notación: "nombreBus<numeroDeBit>". El TMCC no usa esta notación, en las señales externas usa la siguiente: "nombreBus_numeroDeBit" y en las señales internas usa la siguiente: "unNumero_otroNumero_numeroDeBit_nombreBus". Estas notaciones dificultan la visualización de los archivos generados.

Para solucionar esto en el caso de señales externas alcanza con modificar el procedimiento "externalname( )" del archivo "tmcc.y". La versión modificada es la siguiente (incluye modificaciones vistas anteriormente):

```
char *
externalname(b)
    struct bit *b;
    {
     static char name[MAXNAMELEN];

     if(b->variable->width == 1)
          strncpy(name, b->variable->name + 1, MAXNAMELEN);
     else  /* Cuando el ancho es mayor  que uno el "_" por "< >"  */
          sprintf(name, "%s<%d>", b->variable->name + 1, b->bitnumber);
     return(name);
}
```

**Figura B-21** Modificación a procedimiento del módulo output_xilinx.c del Transmogrifier C

Para solucionar el problema con las señales internas hay que modificar el procedimiento "bitname( )" del archivo "tmcc.y". La versión modificada es la siguiente (incluye modificaciones vistas anteriormente):

```
char *
bitname(b)
struct bit *b;
{
 if(b->name)   return(b->name);
 b->name = (char *) malloc(MAXNAMELEN);
 if(b->variable->width == 1) {
   if(b->flags & SYM_KEEPSNAME)
          strncpy(b->name, b->variable->name+1, MAXNAMELEN);
   else
          sprintf(b->name, "%s_%d_%d", b->variable->name+1, thread,
                  b->variable->lineno);  /* Cambio el orden de los parámetros
                                            para que el nombre este primero. */
 } else{
     sprintf(b->name, "%s_%d_%d<%d>", b->variable->name+1, thread,
             b->variable->lineno, b->bitnumber );  /* Cambio el orden de los
                                  parámetros para que el nombre este primero
                                y el nº de bits último, ademas le agrego "< >". */
 }
 return(b->name);
}
```

**Figura B-22** Modificación a procedimiento del módulo tmcc.y del Transmogrifier C

En este caso no alcanza con hacer estas modificaciones ya que en el archivo XNF generado por el TMCC aparecen buses a los que les faltan señales y también aparecen señales de ancho uno a las que le pone los corchetes ("< 0 >") como si fuera un bus. Con estos problemas el archivo XNF, o el VHDL generado usando el traductor, no compila. La solución que encontramos fue crear un programa escrito en lenguaje Perl, llamado FIX_XNF.pl, que procese el archivo XNF generado por el TMCC y corrija esas señales, la fuentes se encuentra al final de éste apéndice (pág. 177).

Dicho programa consta básicamente de 3 procedimientos. El primero recorre el archivo XNF y crea dos hash (listas indexadas por strings) "%global_list" y "%namearray". En la primer lista se asocia cada nombre de variable encontrado (con corchetes y todo) con un uno. En la segunda lista se asocia cada nombre de variable (sin corchetes) con el máximo número de bit encontrado para esa variable. Estas listas quedan de la siguiente forma:

| global_list: | namearray: |
|---|---|
| sum<0> → 1 | sum → 3 |
| sum<3> → 1 | tmp → 0 |
| tmp<0> → 1 | aux → 4 |
| aux<0> → 1 | |
| aux<2> → 1 | |
| aux<4> → 1 | |

El segundo procedimiento procesa estas listas y crea una tercera ("%global_list2") en la que se asocia un valor de variable con el valor que debería tener. Siguiendo el ejemplo anterior esta lista queda así:

| global_list2: |
|---|
| sum<0> → sum<0> |
| sum<3> → sum<1> |
| tmp<0> → tmp |
| aux<0> → aux<0> |
| aux<2> → aux<1> |
| aux<4> → aux<2> |

El tercer procedimiento recorre nuevamente el archivo XNF y crea un nuevo archivo XNF que se diferencia del anterior por los cambios indicados en la lista %global_list2. El nombre de este archivo es igual al del original agregando "_tmp" (ej. sqrt.xnf → sqrt_tmp.xnf ).

## B.4 Fuentes

### LIBLOG:

Programa para Awk.

```
# corrige los problemas del LIBLOG
BEGIN { encontro=0 }
{
  if (encontro==1) {
      print substr($0,1,(length($0)-1))
      encontro=0 }
  else {
          if ($0 ~ /^CSupply/) {
              split($0,a,":")
              print a[1],": LIBLOG2",a[2]
          }
          else { print $0 }
        }
  if ($0 ~ /component LIBLOG2/)
      encontro=1
}
```

### PUERTOS1:

Programa para Awk.

```
# hace una lista de las señales con TXRDY y RXRDY
# para luego corregir vhdl
BEGIN {
  VERS = "1.0"  # indica version del codigo del corrector
  enEntity=0   # 0 no se llego, 1 en entity y 2 ya se termino
  enArchi=0    # 0 no se llego, 1 en achitecture y 2 ya se
termino
}
{ # cuerpo del proceso
    lineas[++cont]=$0
    if (enEntity==0) {
        if ($1 ~ /entity/) {
            nomEntity=$2
            enEntity=1
        }
    } else if (enEntity==1) {
        if (($1 ~ /signal/) && ($2 ~/(TXRDY)|(RXRDY)/)) {
            if ($4 ~ /in/) { sigIN[$2]=1 }
            if ($4 ~ /out/) { sigOUT[$2]=1 }
        } else if (($1 ~ /end/) && (index($2,nomEntity)!=0 )) {
            enEntity=2
        }
    }
    if ((enEntity==2) && (enArchi==0)) {
        if ($1 ~ /architecture/) {
            enArchi=1
        }
    } else if (enArchi==1) {
        if (($1 ~ /signal/) && ($2 ~/(TXRDY)|(RXRDY)/)) {
            sigACH[$2]=1
        } else if ($1 ~ /begin/) {
            enArchi=2
        }
    }
    if ((enEntity==2) && (enArchi==2)) {
        gsub(/[^A-Za-z_0-9]/," ",$0)
        n=split($0,a," ")
        for (i=1; i<=n; i++) {
            if ((a[i] ~/(TXRDY)|(RXRDY)/) && ( ! ((a[i] in sigIN) ||
              (a[i] in sigOUT) || (a[i] in sigACH)))) {
                    bad[a[i]]=a[i]
            }
        }
    }
} # fin del cuerpo del proceso
END {
enArchi=0
for (i=1;i<=NR;i++) {
    n=split(lineas[i],a)
    if (enArchi==0) {
        print lineas[i]
        if (a[1] ~ /architecture/) {
            enArchi=1
        }
    } else if (enArchi==1) {
        if (a[1] ~ /begin/) {
            enArchi=2
            # defino las señales de OUT
            print "-- Señales generadas automaticamente por
                    el corrector del codigo, version " VERS
            for (seniales in sigOUT) {
                print "signal corr" seniales " : std_ulogic;"
            }
            # defino las señales que no esten definidas
            # en ningun lado
            for (seniales in bad) {
                print "signal " seniales " : std_ulogic;"
            }
            print lineas[i]
            # ahora pongo las asignaciones para las
            # señales de OUT
            print "-- Asignaciones generadas automaticamente
                    por el corrector del codigo, version " VERS
            for (seniales in sigOUT) {
                print seniales " <= corr" seniales ";"
            }
            continue
        } else print lineas[i]
    }
    if (enArchi==2) {
        if (lineas[i] ~ /(TXRDY)|(RXRDY)/) {
            # sustituyo los nombres de las señales que agrege
            for (seniales in sigOUT) {
                sigcorr= "corr" seniales
                gsub(seniales,sigcorr,lineas[i])
            }
        }
        print lineas[i]
    }
} #del for
} # del END
```

**NOMBRES:**
Programa para Awk.

```
# para el tmcc, corrige nombres de señales para que sean validos en VHDL
BEGIN {
        VERS = "1.0"  # indica version del codigo del corrector
        FS = ","
        prop = 0
}
function cambiar (nom)
{
        gsub(/ +/,"",nom)   # elimino espacios
        sub(/^[^A-Za-z]+/,"a&",nom)  # corrigo para que comiencen con letra
        gsub(/-+/,"_",nom)  # reemplazo - por _
        gsub(/__./,"_a",nom)  # reemplazo repeticiones consecutivas de _
        return nom
}
{
 if ((prop==0) && ($1 ~ /^PART?/))  prop = 1   # paso propaganda :)
 else if (prop==1) {
     print "USER,SRC, Modificado automaticamente por corrector NOMBRES version " VERS
     prop = 2
  }
 # continuo con el programa normalmente...
 if ($1 ~ /^SYM?/) { # corrijo igual los nombres de los simbolos, total no cuesta nada...
     $2 = cambiar($2)
 }
 if ($1 ~ /^PIN+/) { # corrijo los nombres de los pines
     $4 = cambiar($4)
 }
 if ($1 ~ /^EXT+/) { # corrijo los nombres de las pata de IO
     $2 = cambiar($2)
 }
 for (i = 1; i < NF; i++) { # imprimo la salida
     gsub(/ +/,"",$i)   # elimino espacios
     printf "%s, ",$i
 }
 printf "%s \n",$NF
}
```

**ANDS:**
Programa para Awk.

```
# corrige problemas con el XNF del TMCC
# en los nombres de los simbolos del AND
BEGIN {  VERS = "1.0"  # indica version del codigo del corrector
         enSym=0  } #  1 en descripcion de simbolo
{
 if (enSym==0) {
    if (($1 ~ /SYM/) && ($3 ~ /AND/)) {
        enSym=1
        print "USER,SRC, Modificado automaticamente por corrector ANDS version " VERS
    }
 } else if (enSym==1) {
        if (($1 ~ /PIN/) && ($2 == "I,")) {
            n=split($0,a,",")
            a[2]="I0"
            $0=""
            for (i=1; i<n; i++) {
                $0 = $0 a[i] ", "
            }
            $0 = $0 a[n]
        } else if ($1 ~ /END/) {
            enSym=0
        }
 }
 print $0
}
```

## RENAME2:

Programa para Awk.

```
# renombra la se;ales externas generadas por el hcc
# especialmente indicado para utilizar con RAM int. y ext.
BEGIN {
   VERS = "1.0"  # indica version del codigo
   FS = "[ ,]*" # altero el parser de campos
   j=1
}
{
   if ($1 ~ /^PROG/) { # mando propaganda...
        printf "PROG, renombre de señales, %s, Proyecto
                HandelC, \"%s\" \n",VERS,strftime()
   } else if ($1 ~ /^EXT/) { # señal externa..
       nombre=""
       if ($2 ~ /PADH_C_/) { # caso 1
           n=split($2,a,"_")
           if (a[n-1] ~ /TXRDY/) { # tengo que ver el location
               n=split($4,a,"_")
               nombre=""
               for (i = 3; i <= n-2; i++)
                     nombre = nombre a[i] "_"
               nombre = nombre a[n-1]
           } else {
               m=split($4,b,"_")
               nombre=""
               for (i = 3; i <= n-3; i++)
                     nombre = nombre a[i] "_"
               if (b[m] == 1)
                     nombre = nombre a[n-2]
                else
                     nombre = nombre a[n-2] "<" a[n-1] ">"
               }
           } else if ($2 ~ /PADH_/) { # caso 2
               n=split($4,a,"=")
               nombre=a[2]
           }
           if (nombre != "") {
               tabla[$2]=nombre
           }
       }
       lin[j]=$0
       j=j+1
}
END {
       for (i=1;i<j;i++){
           salida=lin[i]
           for (nombre in tabla) {
               pos=index(lin[i],nombre)
               if (pos != 0) {
                   salida=substr(lin[i],1,pos-1) tabla[nombre]
                          substr(lin[i],pos+length(nombre))
                   break
               }
           }
           print salida
       }
}
```

## UNDERSCORES:

Programa para Awk.

```
# busca y sustituye los dobles o mas underscores que aparaezcan, pues dan problemas en VHDL,
#  y los sustituye por un underscore + primer letra del nombre + numero de underscores encontrados
# para el XNF a utilizar antes del RENAME2
BEGIN {
        VERS = "1.0"  # indica version del codigo
}
{
  do {
        i = match($0,/__+/)
        if (RLENGTH !=-1) {
                $0 = substr($0,1,i) substr($0,i+RLENGTH,1) RLENGTH substr($0,i+RLENGTH)
        }
  }
  while (i !=0)
  print $0
}
```

## BUFT:
Programa para Awk.

```
# corrige problemas de habilitaion de buffer triestado para
# utilizar con RAM internas y externas con HCC
BEGIN {  VERS = "1.0"  # indica version del codigo
  FS = "[ ,]*" # altero el parser de campos
  j=1
  enSYMo=0  }
{
  if ($1 ~ /^PART/) { # mando propaganda...
       $0=$0 sprintf("PROG, corrector de buffer triestado,
            %s, Proyecto HandelC, \"%s\" \n",VERS,strftime())
  } else if (($1 ~ /^PIN/)&&(enSYMo)&&($2 ~ /^I/)) {
               # registro nombre de senial de hab. nueva
               ShabNEW[nomRAM] = $4
               enSYMo=0
  } else if (($1 ~ /^SYM/)&&($3 ~ /^OBUF/)) {
               # simbolo de bufer de salida....
               n=split($2,a,"_")
               if (a[n] ~ /^wb/) { # guardo nombre de RAM
                    nombre=""
                    for (i = 1; i <= n-2; i++)
                         nombre = nombre a[i] "_"
                    nomRAM = nombre a[n-1]
                   enSYMo=1
               }
  }
  lin[j]=$0
  j=j+1
}
END {
   for (i=1;i<j;i++){
        salida=lin[i]
        n=split(salida,a,"[ ,]*")
        if ((a[1] ~ /^SYM/)&&(a[3] ~ /^OBUFT/)) {
             n=split(a[2],b,"_")
             nombre=""
             for (k = 1; k <= n-2; k++)
                  nombre = nombre b[k] "_"
             nomRAM = nombre b[n-1]
             enSYMt=1
        } else if ((a[1] ~ /^PIN/) && (a[2] ~ /^T/)&&(enSYMt)) {
                   enSYMo=0
                   a[4]=ShabNEW[nomRAM]
                   salida=""
                   for (k = 1; k <= n-1; k++)
                        salida = salida a[k] ", "
                   salida = salida a[n]
        }
        print salida
   }
}
```

## FIX_XNF:
Programa en Perl.

```
#!/usr/bin/perl
$xnffilename = $ARGV[0];
&BuildSignalList($xnffilename);
&ProcesoLista();
&EscriboArchivo($xnffilename);

sub BuildSignalList {
     local($xnffilename) = @_;
     local @line;
     open(XNFFILE,$xnffilename);
     while($theline = <XNFFILE>) {
        if($theline =~ /^PIN/) {
             @line = split(/,/,$theline);
             $name = $line[3];
             $name =~ s/[ \n]+//g;
             if($name =~ /\</) {
                  $global_list{$name}=1;
                  ($name,$size) = split(/</,$name);
                  $size =~ s/>//;
                  if($namearray{$name} eq "") {
                       $namearray{$name} = $size;
                  } else {
                       if($namearray{$name} < $size) {
                            $namearray{$name} = $size;
                       }
                  }
             }
        }
     }  # del while
     close(XNFFILE); }

sub ProcesoLista{
    local @a, $b, @c, $num_bit;
    foreach $name (sort(keys(%namearray))) {
        $num_bit=0;
        for($i=0; $i<=$namearray{$name}; $i=$i+1) {
             @a=($name,"<",$i,">");
             $b=join("",@a);
             if ($global_list{$b}==1){
                  @c=($name,"<",$num_bit,">");
                  $global_list2{$b}=join("",@c);
                  $num_bit=$num_bit+1;
                  $aux1=$b;
                  $aux2=$name;
             }
        }  # del for
        if ($num_bit==1){
             $global_list2{$aux1}=$aux2;
        }
    }
}

sub EscriboArchivo{
     local($xnffilename) = @_;
     open(XNFFILE,$xnffilename);
     $newfilename = $xnffilename;
     $newfilename =~ s/\.xnf/_tmp.xnf/;
     open(NEWFILE,"> ".$newfilename);
     while($theline = <XNFFILE>) {
          if(($theline =~ /^PIN/)&&($theline =~/</)) {
             @line = split(/,/,$theline);
             $name = $line[3];
             $name =~ s/[ \n]+//g;
             $name2=$global_list2{$name};
             if ($#line==3){
                 $name2=~ s/$/\n/;
             }
             $line[3]=$name2;
             $theline=join(",",@line);
             printf(NEWFILE $theline);
          }  else { printf(NEWFILE $theline); }
     }  # del while
}
```

# C Guía de compilación

En este apéndice se encuentran sugerencias de cómo armar los programas para Handel C y Transmogrifier C, además se explica en forma práctica el uso de los diferentes correctores y de la utilización del traductor de XNF a VHDL.

## C.1 Como compilar con Hcc

### C.1.1 Estructura del programa

```
const spec <nombre del target>={ especificación del target};

const spec <nombre de especificación de RAM externa>={especificación de RAM};

main( target = <nombre del target>,
port (<dirección>) <nombre puerto> : <ancho de bits>,
eram <nombre de RAM externa>[ <cantidad de datos en la RAM> ]=<nombre de
especificación de RAM externa>: <ancho en bits>;
        ... // otros puertos y/o erams )
{
        int <nombre variable global> : <ancho en bits>;
        ram <nombre RAM interna>[ <cantidad de datos en la RAM> ] : <ancho en bits>
        rom <nombre ROM interna>[ <cantidad de datos en la ROM> ] : <ancho en bits>
        ... // otras variables globales

        void <nombre procedimiento>( ){
        int <nombres variables locales> : <ancho de bits>;
        ... // cuerpo del procedimiento
        }

        ... // otros procedimientos

        ... // cuerpo del programa principal
}
```

**Figura C-1** Estructura de los programas en Handel C

### C.1.2 Especificación del target

El siguiente es un ejemplo de las opciones que ofrece la especificación del target.

```
fpga type = "Xilinx3000",
fpga chip = "3195PQ160-3",
clock pad = "P160",
not error pad = "P55",
finish pad = "P44",
clock divider = "1",
carry weight = "50",
critical weight = "100"
```

Sin embargo nosotros compilamos nuestros programas usando la siguiente especificación:

```
const spec tg = {
        fpga_type = “Xilinx4000”,
        clock_pad = “P10”
};
```

La primera opción es para que se genere un archivo XNF, no generamos VHDL para poder aplicar el corrector RENAME2, ya que éste procesa archivos en formato XNF. La segunda opción es para que no use oscilador interno, sino que tome la señal de reloj externa.

### C.1.3 Especificación de las RAM externas

La especificación de las memorias RAM externas la usamos de la siguiente forma:

```
Const spec tipo_mem={
        Addr = {"mem_Addr<0>", "mem_Addr<1>","mem_Addr<2>",
                "mem_Addr<3>",  ...  , "mem_Addr<14>","mem_Addr<15>"},
        data ={"mem_Data<0>","mem_Data<1>","mem_Data<2>",
                "mem_Data<3>",  ...  , "mem_Data<14>","mem_Data<15>"},
        ce   = "mem_ce",
        wb   = "mem_wb",
        en   = "mem_en"
};
```

**Figura C-2** Especificación para las memorias RAM externas

En lugar de los números de pin ponemos el nombre que queremos que tenga la señal en el archivo XNF, en este caso queríamos que nuestras señales se llamaran: mem_Addr, mem_Data, mem_ce, mem_wb, y mem_en.

## C.1.4 RAM interna

En el siguiente cuadro mostramos la declaración y formas de uso de la RAM nativa.

```
... // declaración de puertos y variables
ram <nombre>[<tamaño>]: <ancho en bits>;  // declaro la ram
...
     <nombre>[<posición>]=<valor>;    // escribo en la ram una constante
     <nombre>[<posición>]=<variable>; // escribo el valor de una variable
     <nombre>[<posición>]=<expresión>; // escribo el resultado de una expresión
     <variable>= <nombre>[<posición>]  // leo en una variable
     <variable>= expresión( <nombre>[<posición>] );  /* uso el valor de la ram en
                                 una expresión y el resultado lo cargo en una variable */
```

**Figura C-3** Declaración y formas de uso de la memoria RAM interna nativa de Handel C.

Si se desea usar la megafunción lpm_ram_dq de Altera se tienen los siguientes casos:

```
Void main(... // declaro target y otros puertos
          port (out) ram_<nombre>_we:1,
          port (out) ram_<nombre>_din: <ancho de datos>,
          port (out) ram_<nombre>_dir: <ancho de direcciones>,
          port (in)  ram_<nombre>_dot: <ancho de datos>){
...
// lectura de la ram
ram_<nombre>_dir ! <expresión>; /* en lugar de una expr. puede ser variable o constante */
<variable> = expresión(ram_<nombre>_dot);  /* en un ciclo (estado) pongo la
                                              dirección y en el siguiente uso el dato*/
// escritura en la ram, recordar que we, din y dir se registran
par{
          ram_<nombre>_we ! 1;  // en lugar de sacar un uno se puede usar una var.
          ram_<nombre>_din ! <expresión>; /* en lugar de una expresión puede ser
                                             una constante o una variable */
          ram_<nombre>_dir ! <expresión>; // idem
}
```

**Figura C-4** Forma de usar la megafunción Lpm_ram_dq de Altera.

En este caso hay que pasarle el parámetro "Alt" al traductor para que haga los cambios apropiados.

En la Figura C-5 vemos como usar módulos Sync_ram de Xilinx. Para sintetizar un diseño con estos primero hay que crearlos en el proyecto, esto se hace con la herramienta Logiblox de Foundation o ISE. El nombre del módulo debe ser "<tipo>_ram" y se debe crear un módulo por cada tipo usado en el diseño.

```
void main(... //declaro target y otros puertos
port (out) ram_<tipo>_<nombre>_we:1,
port (out) ram_<tipo>_<nombre>_din: <ancho de datos>,
port (out) ram_<tipo>_<nombre>_dir: <ancho de direcciones>,
port (in)  ram_<tipo>_<nombre>_dot: <ancho de datos>){
...
// lectura de la ram en este caso no se registran las direcciones
par{
ram_<tipo>_<nombre>_dir ! <expresión>; /* en lugar de una expresión puede
                                          ser una variable o una constante */
<variable> = expresión(ram_<tipo>_<nombre>_dot); /* en un ciclo (estado)
                                  pongo la dirección y en el siguiente uso el dato */
}

// escritura en la ram, sí se registran  we, dir y  din
par{
ram_<tipo>_<nombre>_we ! 1;  // en lugar de sacar un uno se puede usar  una variable
ram_<tipo>_<nombre>_din ! <expresión>; /* en lugar de una expresión
                                    puede ser una constante o una variable */
ram_<tipo>_<nombre>_dir ! <expresión>; // idem
}
```

**Figura C-5** Forma de usar los módulos SYNC_RAM de Xilinx.

## C.1.5 ROM interna

La memoria ROM provista por HandelC se usa de la siguiente forma:

```
… // declaración de puertos y variables
rom <nombre>={<pos. 0>, <pos.1>, …,<pos. N>}: <ancho en bits>; /* pos. X representa
                                                        el  contenido de la posición X */
…
        <variable>= <nombre>[<posición>];  // leo en una variable
        <variable>= expresión( <nombre>[<posición>] );  /* uso el valor de la rom en una
                                    expresión y el resultado lo cargo en una variable */
```

**Figura C-6** Forma de usar la memoria ROM nativa de Handel C.

Cabe recordar que esta memoria no usa flip-flops.

Si se desea usar celdas de memoria en chips de Altera hay que usar la megafunción lpm_rom. Esta se usa de la siguiente forma:

```
void main(... // declaro target y otros puertos
port (out) rom_<nombre>_dir: <ancho de direcciones>,
port (in)  rom_<nombre>_dot: <ancho de datos>){
...
// lectura de la rom
par{
rom_<nombre>_dir ! <expresión>; /* en lugar de una expresión puede ser
                                    una variable o una constante */
<variable> = expresión(rom_<nombre>_dot); /* en lugar de cargar en una
                variable, la expresión puede estar en un if(),  while(), .. */
}
```

**Figura C-7** Forma de usar la megafunción Lpm_rom de Altera.

La inicialización de la memoria se hace mediante un archivo de texto que debe crear el diseñador. El nombre debe ser "rom_<nombre>.mif" y el formato es el siguiente:

```
DEPTH = 32;       % Ancho y profundidad de la memoria, en decimal    %
WIDTH = 14;

ADDRESS_RADIX = HEX;      % radixes de direcciones y datos, por defecto son HEX %
DATA_RADIX = HEX;   % Pueden ser BIN, DEC, HEX, or OCT              %

CONTENT
BEGIN
% Ejemplo, {3fff, 3fff, 3fff, 3fff, 3fff, 3fff, 000f, 000a, 000e, 0005, f000, ffff, 0001, 0002} %
        [0..5]    :         3FFF;   % rango- las direcciones de 0 a 5 = 3FFF          %
        6         :         000F;   % dirección simple—la dirección  6 = F  %
        7         :         000A 000E 0005 F000 FFFF 0001 0002; % rango comenzando por
                                    una dirección—addr[7]=A, addr[8]=E, addr[9]=5,...%
END ;
```

**Figura C-8** Formato MIF, para inicialización de memorias ROM.

## C.1.6 Módulos de multiplicación

Pese a que HandelC ya cuenta con un operador de multiplicación, los módulos de Altera (lpm_mult) y Xilinx (*) permiten mejores resultados.

Estos módulos se usan de la siguiente forma:

```
void main(... // declaración de target y otros puertos
port (out) mul_<nombre>_mn: <ancho en bits>,
port (out) mul_<nombre>_mr: <ancho en bits>,
port (in) mul_<nombre>_res: <ancho en bits>){

...
// multiplicación
par{
        mul_<nombre>_mn ! <expresión>;
        mul_<nombre>_mr ! <expresión>;
        <variable> = expresión(mul_<nombre>_res);
}
```

**Figura C-9** Uso de módulos de multiplicación.

Pese a que como vimos se pueden usar expresiones como entradas al multiplicador y a su vez usar la salida del multiplicador en una expresión, lo recomendable es usar variables como entradas y el resultado guardarlo en una variable. Esto es porque generalmente el camino crítico en un diseño está en los multiplicadores.

Si se desea usar la megafunción lpm_mult, se le debe pasar el parámetro "Alt" al traductor de XNF a VHDL, si se le pasa otro parámetro o nada el traductor inserta el operador *.

Los multiplicadores insertados son con signo, si se desea trabajar sin signo se puede modificar el VHDL resultante o agregarle un bit a las entradas al multiplicador forzando éste a 0.

### C.1.7 Subexpresiones

Permiten ahorrar hardware, cuando una expresión se usa más de una vez conviene declararla aparte. En la siguiente figura mostramos como se declara y usa una subexpresion, también incluimos un ejemplo de su uso.

```
... // declaro variables que uso en la subexpresion
int <nombre>( ) = <expresión>;     // declaro la subexpresion
... // sigue el programa
<variable>=<nombre>( );  // asigno a una variable
if (<nombre>( ) ){...};   // uso la subexpresion en un if
<variable>=<expresión(<nombre>( )) > // uso la subexpresion en otra expresión
... // fin del programa
```

**Ejemplo:**

```
...                  // declaraciones de target y puertos
int mr,mn:4;         // declaro las variables de la subexpresion..
...                  // declaro otras variables y rams (x, y)
int mul()=mn*mr;           // declaro la subexp.

      while(1){
            ...        // inicializo las ram x e y
            mn, mr=x[0], y[4]; /* cargo en las variables de la subexp. los primeros
                                  valores a usar */
            par{ z ! mul(); mn, mr=x[1], y[3];} /* uso la subexp. y simultáneamente
                                                   cargo valores para usarla nuevamente */
            par{ z ! mul(); mn, mr=x[2], y[7];}
            ...
            par{ z ! mul(); mn, mr=x[7], y[5];}
            z ! mul();
      };
}
```

**Figura C-10** Declaración y ejemplo de uso de las subexpresiones.

## C.1.8 Compilación de los archivos y correcciones

Luego de generado el archivo C con las opciones recomendadas, éste se compila de la siguiente forma:

```
Hcc –ns –cpp archivo.c
awk –f rename2 archivo.xnf > temp.xnf
del archivo.xnf
awk –f buft temp.xnf > archivo.xnf
perl – x2v.pl archivo.xnf {Alt}
```

**Figura C-11** Secuencia de comandos para compilar con Handel C.

El parámetro "Alt" (opcional) indica que la tecnología de destino es Altera, es necesario indicarlo en el caso de que se use alguna de las megafunciones que reconoce el traductor (lpm_mult, lpm_ram_dq, lpm_rom). En caso de no usar memorias RAM externas no es necesario usar el corrector "buft".

Es muy útil incluir todo esto en un archivo de proceso por lotes (.bat) que acepte como parámetros el nombre del archivo y la tecnología de destino. Este puede ser de la siguiente forma:

```
hcc –ns –cpp %1.c
awk –f rename2 %1.xnf > temp.xnf
del %1.xnf
awk –f buft temp.xnf > %1.xnf
perl – x2v.pl %1.xnf %2
```

**Figura C-12** Archivo de proceso por lotes para compilar con Handel C.

## C.2 Como compilar con el Tmcc

### C.2.1 Estructura de los programas

```
main(){
#pragma intbits N   // ancho de las variables que se declaren hasta el próximo intbits
int <nombre de variable>;
...   // otras variables

inputport(<nombre de puerto de entrada>);  /* los puertos y buses se deben declarar
                                              antes  como variables */
outputport(<nombre de puerto de salida>);
busport(<nombre de bus>);
...  // otros puertos

portflags(<nombre de puerto>, <atributos>);
... // otros portflags
...  // cuerpo del programa

}
#pragma intbits M    // ancho de bits del valor que retorna la función
<nombre función>(<parámetro 1>, <parámetro 2>, ...)
#pragma intbits m1  // ancho de bits del parámetro 1
int <parámetro 1>
#pragma intbits m2
int <parámetro 2>
....
{
int <variables>;  // variables locales
... // cuerpo de la función
}
... // otras funciones
```

**Figura C-13** Estructura de los programas en Transmogrifier C.

### C.2.2 Reloj

Para usar como reloj una señal externa, es necesario crear un archivo XNF en el que se declare esto, éste archivo puede ser así:

```
SYM, CLK-AA, BUFGP
PIN, I, I, HCLK
PIN, O, O, CLK
END
EXT, HCLK, I
```

**Figura C-14** Archivo XNF de señal de reloj para los programas en Transmogrifier C

## C.2.3 Hilos

Cuando se quieren realizar varios procesos en paralelo, en lugar de intentar hacerlos en un mismo programa el TMCC ofrece la posibilidad de compilarlos en archivos separados. Para comunicar los diferentes procesos se usan puertos, pero a estos hay que declararlos que no son "PORT_PIN" para que no sean entradas o salidas del circuito final (luego de juntar los hilos). A continuación vemos un ejemplo de declaraciones para dos hilos.

```
main( ){ // hilo1
... // declaración de variables
   int com1, com2; /* señales de
                comunicación con el otro hilo */
... // declaro otros puertos
   inputport(com1); outputport(com2);
... // ajusto propiedades de otros puertos
   portflags(com1, 0); // un cable
   portflags(com2, 0x2); // salida registrada
...
```

```
main( ){  // hilo2
... // declaración de variables
   int com1, com2; /* señales de
                comunicación con el otro hilo */
... // declaro otros puertos
   outputport(com1); inputport(com2);
... // ajusto propiedades de otros puertos
   portflags(com2, 0); // un cable
   portflags(com1, 0x2); // salida registrada
...
```

**Figura C-15** Ejemplo de comunicación entre dos hilos.

## C.2.4 Tipos de puertos

La sentencia portflags permite ajustar las propiedades de los puertos. Su forma de uso es la siguiente:

```
main(){
... // declaración de variables, incluida la variable del puerto que queremos ajustar
... // declaración de puertos, incluido el puerto que queremos ajustar
portflags(<nombre de puerto>, <valor> {| <valor> { ...}});  /* asignamos el or de las
                                        propiedades que queremos setearle al puerto */
... // sigue el programa
```

**Figura C-16** Ajuste de propiedades de los puertos.

En la siguiente tabla mostramos las diferentes propiedades

**Tabla C-1** Valores de las propiedades de un puerto.

| Valor | Descripción |
|---|---|
| 0x0 | El puerto es un cable. |
| 0x1 | Port_pin: indica que el puerto es externo |
| 0x2 | Port_registered: intercala un flip-flop entre el puerto y el resto del circuito. |
| 0x4 | Port_pullup: habilita una resist. pullup interna en un puerto bidireccional |
| 0x8 | Port_pulldown: habilita una resistencia pulldown interna en un puerto bidir. |

## C.2.5 Buses

El TMCC permite usar puertos bidireccionales, su forma de declararlos y usarlos es la siguiente:

```
... // otras declaraciones
#pragma intbits <ancho de bits del puerto>
int <nombre del puerto>;
bus_port( <nombre del puerto> );
...// sigue el progama, al comienzo el puerto esta en su modo de entrada (triestado)
        <variable> = <nombre del puerto>; /* lo uso como entrada, se puede usar también en
                                             expresiones como cualquier otra variable */
        <nombre del puerto> = <valor> /* al escribir en él estando en modo entrada, el
                                         puerto, pasa automáticamente al modo salida, "valor"
                                         puede ser una variable o el resultado de una expresión */
        bus_idle( <nombre del puerto> ); /* para usarlo como entrada cuando está en modo
                                            salida tengo que usar esta sentencia */
...
```

**Figura C-17** Declaración y uso de puertos bidireccionales.

## C.2.6 RAM interna

El TMCC no tiene previstas las memorias internas, por lo que para usar celdas de memoria se debe usar una de las funcionalidades que le agregamos al traductor.

Si se desea usar la megafunción lpm_ram_dq de Altera se tienen los siguientes casos:

```
main(){
  int  ram_<nombre>_we;
... //declaro otras variables de 1 bit
#pragma intbits <ancho de datos>
  int ram_<nombre>_din, ram_<nombre>_dot;
#pragma intbits <ancho de direcciones>
  int ram_<nombre>_dir;

    outputport(ram_<nombre>_we);
    outputport(ram_<nombre>_din);
    outputport(ram_<nombre>_dir);
    inputport(ram_<nombre>_dot);
... // declaro otros puertos y sus portflags de
ser necesario
    portflags(ram_<nombre>_we, 0x1);
    portflags(ram_<nombre>_din,0x1);
    portflags(ram_<nombre>_dir,0x1);  /* estos
puertos no es necesario registrarlos porque la
 lpm_ram_dq ya lo hace */
```

```
// lectura de la ram
   ram_<nombre>_dir = <expresión>;
/* en lugar de una expresión puede ser una
variable o una constante */
  while(0){}; // espero al siguiente ciclo
  <variable> = expresión(ram_<nombre>_dot);
/* en un ciclo (estado) pongo la dirección y en
el siguiente uso el dato */

// escritura en la ram, recordar que we, din y
dir se registran
  ram_<nombre>_we = 1;  // en lugar de sacar
un uno se puede usar una var.
  ram_<nombre>_din = <expresión>; /* en
lugar de una expresión puede ser una
constante o una variable */
  ram_<nombre>_dir = <expresión>; // idem
```

**Figura C-18** Declaración y formas de uso de la megafunción lpm_ram_dq.

En este caso hay que pasarle el parámetro "Alt" al traductor para que haga los cambios apropiados.

En la siguiente figura vemos como usar módulos Sync_ram de Xilinx.

```
main(){
... /* la parte de declaración de variables, puertos y portflags lo único que cambia es que las
señales de la ram pasan de llamarse ram_<nombre>_xx a ram_<tipo>_<nombre>_xx */
 ...
// lectura de la ram en este caso no se registran las direcciones
    ram_<tipo>_<nombre>_dir = <expresión>; /* en lugar de una expresión puede ser
                                              una variable o una constante */
    <variable> = expresión(ram_<tipo>_<nombre>_dot); /* en un ciclo (estado) pongo la
                                                        dirección y en el siguiente uso el dato */
// escritura en la ram, sí se registran  we, dir y  din
    ram_<tipo>_<nombre>_we = 1;  /* en lugar de sacar un uno se puede usar una variable */
    ram_<tipo>_<nombre>_din = <expresión>; /* en lugar de una expresión puede ser una
                                              constante o una variable */
    ram_<tipo>_<nombre>_dir = <expresión>; // idem
```

**Figura C-19** Declaración y formas de uso del módulo Sync_ram.

En este caso no es necesario pasarle un parámetro adicional al traductor.

Además hay que crear un módulo Sync_ram en el proyecto, esto se hace con la herramienta Logiblox de Foundation o ISE. El nombre del módulo debe ser: <tipo>_ram.

## C.2.7 ROM interna

Uno de los formatos de salida del tmcc es "xc4000roms", este formato lo que hace es, en lugar de usar compuertas AND y OR , usa el símbolo ROM del XNF. Si bien esto es más compacto, el circuito final, luego de compilado con alguna otra herramienta (Max+Plus II, Synplicity, etc.), es más grande y/o lento. Por esta razón si estamos realizando un circuito que usa una memoria ROM, conviene compilar ésta aparte (hilo 1) con el formato "xc4000roms" y luego compilar el resto del circuito (hilo 2) comunicándolo con el circuito de la ROM. Cabe destacar que el traductor (X2V) no reconoce el símbolo ROM por lo que no se debe usar este formato si se desea usar el traductor.

En la siguiente figura mostramos dos opciones de como generar una ROM, la primera de ellas produce un circuito más compacto que la otra.

| **Forma recomendada** | **Forma no recomendada** |
|---|---|
| ....<br>if (DIR == 0){ DATO == XXX0;} else {<br>if (DIR == 1){ DATO == XXX1;} else {<br>if (DIR == 2){ DATO == XXX2;} else {<br>if (DIR == 3){ DATO == XXX3;} else {<br>....<br>if (DIR == 14){ DATO == XXX14;} else {<br>if (DIR == 15){ DATO == XXX15;};};...};<br>… | ....<br>if (DIR == 0){ DATO == XXX0;};<br>if (DIR == 1){ DATO == XXX1;};<br>if (DIR == 2){ DATO == XXX2;};<br>if (DIR == 3){ DATO == XXX3;};<br>...<br>if (DIR == 14){ DATO == XXX14;};<br>if (DIR == 15){ DATO == XXX15;};<br>… |

**Figura C-20** Código que muestra como implementar una ROM usando el formato xc4000roms.

Si se desea usar celdas de memoria en chips de Altera hay que usar la megafunción lpm_rom. Esta se usa de la siguiente forma:

```
main(){
#pragma inbits <ancho de direcciones>
int rom_<nombre>_dir;
#pragma inbits <ancho de datos>
int rom_<nombre>_dot;
...
// lectura de la rom
          par{
rom_<nombre>_dir ! <expresión>; /* en lugar de una expresión puede ser
                                   una variable o una constante */
<variable> = expresión(rom_<nombre>_dot); /* en lugar de cargar en una
                  variable, la expresión puede estar  en un if(),  while(), etc. */
}
```

**Figura C-21** Declaración y forma de uso de la megafunción lpm_rom.

La inicialización de la memoria se hace mediante un archivo de texto que debe crear el diseñador. El nombre debe ser "rom_<nombre>.mif" y el formato es el de la Figura C-8.

## C.2.8 Módulos de multiplicación

El TMCC no cuenta con un operador de multiplicación, se puede implementar como una función pero es conveniente usar los módulos de Altera (lpm_mult) y Xilinx (*) ya que permiten mejores resultados.

Estos módulos se usan de la siguiente forma:

```
main( ){
#pragma intbits <ancho en bits del multiplicando>
   int mul_<nombre>_mn;
#pragma intbits <ancho en bits del multiplicador>
   int mul_<nombre>_mr;
#pragma intbits <ancho en bits del resultado>
   int mul_<nombre>_res;
...
// multiplicación
   mul_<nombre>_mn = <expresión>;
   mul_<nombre>_mr = <expresión>;
   <variable> = expresión(mul_<nombre>_res);
}
```

**Figura C-22** Declaración y uso de los módulos de multiplicación.

Pese a que como vimos se pueden usar expresiones como entradas al multiplicador y a su vez usar la salida del multiplicador en una expresión, lo recomendable es usar variables como entradas y el resultado guardarlo en una variable. Esto es porque generalmente el camino critico en un diseño está en los multiplicadores.

Si se desea usar la megafunción lpm_mult, se le debe pasar el parámetro "Alt" al traductor de XNF a VHDL, si se le pasa otro parámetro o nada el traductor inserta el operador *.

Los multiplicadores insertados son con signo, si se desea trabajar sin signo se puede modificar el VHDL resultante o agregarle un bit a las entradas al multiplicador forzando este a 0.

## C.2.9 Compilación de los archivos y correcciones

Luego de generado el archivo C, éste se compila de la siguiente forma:

```
tmcc –S archivo.c myclock.xnf
awk –f nombres archivo.xnf > tmp.xnf
rm archivo.xnf
mv tmp.xnf archivo.xnf
perl -- x2v.pl archivo.xnf {Alt}
```

**Figura C-23** Secuencia de comandos para compilar con Transmogrifier C.

En caso de que se corrijan las fuentes del TMCC la compilación es de la siguiente forma:

```
tmcc –S archivo.c myclock.xnf
perl -- x2v.pl archivo.xnf {Alt}
```

**Figura C-24** Secuencia de comandos para compilar con Transmogrifier C modificado.

En caso de que se haya compilado el diseño en hilos (3 en el ejemplo), este se compila de la siguiente forma:

```
tmcc –S –c –T1 <hilo1>.c
tmcc –S –c –T2 <hilo2>.c
tmcc –S –c <hilo3>.c <hilo1>.xnf <hilo2>.xnf myclock.xnf
perl – x2v.pl <hilo3>.xnf {Alt}
```

**Figura C-25** Secuencia de comandos para compilar en hilos con Transmogrifier C modificado.

El archivo "myclock.xnf" contiene la señal de reloj externa.
El parámetro "Alt" (opcional) indica que la tecnología de destino es Altera, es necesario indicarlo en el caso de que se use alguna de las megafunciones que reconoce el traductor (lpm_mult, lpm_ram_dq, lpm_rom).

# D Estado del arte

En este apéndice encontraremos un resumen de la búsqueda de herramientas para la conversión del lenguaje de alto nivel (en este caso C o C++) a un lenguaje de descripción de hardware.

## D.1 OSCI (Open System C Initiative)

En septiembre de 1999, compañías lideres en EDA, IP, semiconductores, sistemas y software embebido, anunciaron la "Open SystemC Initiative" (OSCI) y la disponibilidad inmediata de una plataforma de modelado en C++ llamada SystemC para la descarga libre por Internet. Logrando un adelanto en cooperación industrial, SystemC es el primer resultado de la iniciativa, que promueve, permite y acelera el intercambio y codiseño de modelos IP a nivel de sistema, usando una plataforma de modelado común en C++. Mediante el modelo de licencia de la comunidad abierta, los diseñadores pueden crear, validar y compartir modelos con otras compañías usando SystemC y un compilador C++ estándar. En adición, vendedores EDA tienen acceso completo a la plataforma de modelado requerida para construir herramientas interoperables. El grupo líder incluye a ARM, CoWare Inc., Cygnus Solutions, Ericsson, Fujitsu, Infineon, Lucent Technologies, Sony Corporation, STMicroelectronics, Synopsys Inc. y Texas Instruments.

**SystemC**

En una descripción en SystemC, el bloque constructor fundamental es el proceso. Un proceso es como una función en C/C++ que implementa comportamiento. Una descripción completa del sistema consiste de múltiples procesos concurrentes. Los procesos se comunican unos con otros mediante señales, y relojes explícitos pueden ser usados para ordenar eventos y sincronizar procesos. Todos los bloques constructores son objetos (clases) que son parte de SystemC. Tipos de datos especiales requeridos para modelar hardware eficientemente son provistos como parte de la librería.

Usando la librería SystemC, un sistema puede ser especificado a varios niveles de abstracción. Al nivel más alto, solo la funcionalidad del sistema puede ser modelada. Para la implementación en hardware, los modelos pueden ser escritos en un estilo funcional o en un estilo de nivel de transferencia de registros (RTL). La parte de software del sistema puede ser naturalmente descrita en C/C++. Interfaces entre bloques software y hardware o entre bloques hardware puede ser fácilmente descrita al nivel de precisión de transacción (transaction-accurate) o al nivel de precisión de ciclo (cicle-accurate). Es

mas, diferentes partes del sistema pueden ser modeladas a diferentes niveles de abstracción y estos modelos pueden co-existir durante la simulación del sistema.

Las clases de SystemC y de C/C++ pueden ser usadas no solo para el desarrollo del sistema, sino que también para los bancos de prueba (testbench). La funcionalidad de las clases de SystemC junto con la naturaleza orientada a objetos de C++ proveen un poderoso mecanismo para desarrollar bancos de prueba compactos, eficientes y reusables. SystemC consiste de un conjunto de archivos de encabezado (header files) que describen las clases mas una librería de enlace que contiene el núcleo de simulación. El archivo de encabezado puede ser usado por el diseñador en su programa. Cualquier compilador conforme a ANSI C++ puede compilar SystemC, junto con el programa. Durante el enlazado (linking), es usada la librería SystemC, que contiene el núcleo de simulación. El ejecutable resultante sirve como simulador para el sistema descrito.

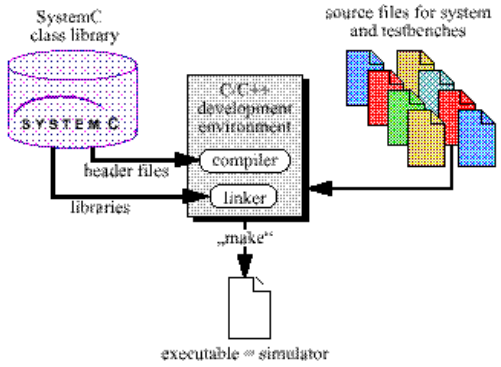



**Figura D-1** Esquema del proceso de compilación de un programa en C

En este capítulo analizamos algunas herramientas que permiten la implementación en hardware de código en SystemC:

- A|RT Designer y A|RT Builder de Frontier Designs
- Sistemsim de CoDesign Automation
- CoCentric System Studio y CoCentric SystemC Compiler de Synopsys

## D.2 Frontier Design[12]

Las herramientas tipo EDA de esta empresa establece un puente entre sistemas complejos escritos en C y su implementación final en una arquitectura de hardware óptima. **[13]**
La empresa cuenta con 15 años de experiencia en el pasaje de algoritmos escritos en C para DSP (Digital Signal Processing) a chips, esto les lleva a crear una metodología llamada Algorithm to Register Transfer o A|RT.

Como podemos observar de la Figura D-2 el proceso de transformación es centralizado por el A|RT Designer el cual toma un código C y lo transforma en un código VHDL o Verilog HDL pronto para utilizar con alguna herramienta de síntesis de FPGA por ejemplo.

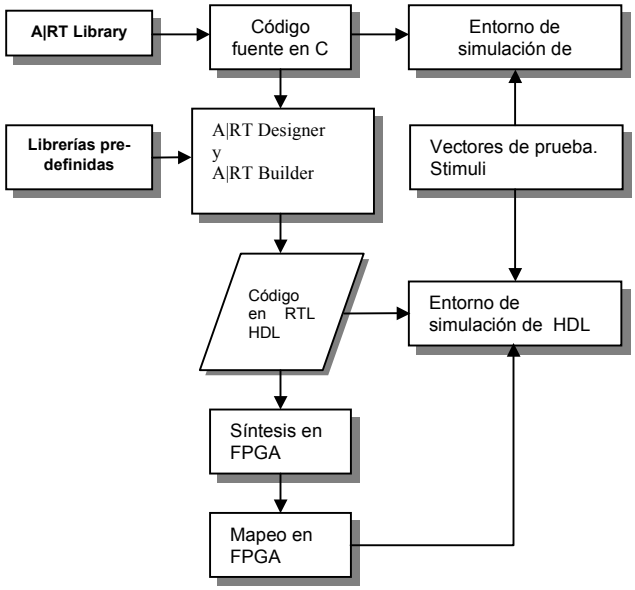



**Figura D-2** Proceso de diseño con A|RT Designer

Veamos a continuación las características de los tres principales productos desarrollados: A|RT Designer, A|RT Builder y A|RT Library

### D.2.1 A|RT Library

Una de las mayores dificultades al pasar de un algoritmo en C a una implementación en hardware es que es necesario crear un algoritmo que trabaje con punto fijo, tanto para la representación de los datos como para sus operaciones. A|RT Library provee una solución robusta y elegante que simplifica el agregar la aritmética de punto fijo a los programas. Esto se presenta como un subconjunto de clases de C++ que encapsula las características de los tipos de datos de punto fijo y sus operadores más comunes.

[12] En setiembre 2001 se fusionó con Philips Semiconductors DSP group y formaron una nueva empresa llamada Adelante Technologies (http://www.adelantetechnologies.com/).

## D.2.2 A|RT Designer

Esta herramienta nos permite compilar descripciones en C de moderada y alta complejidad, genera automáticamente una arquitectura de datapath reconfigurable, más un procesador de instrucciones complejas (VLIW Very Large Instruction Word) y el microcódigo necesario.
Con A|RT Designer se puede interactivamente agregar o remover recursos como ALUs, ROMs, RAMs, etc., reasignar operaciones y analizar el efecto de estos cambios en la performance del sistema final.
En la siguiente figura se puede apreciar un esquema simplificado de la arquitectura generada. ASU = Aplication Specific Unit es donde se encuentra el bloque definido por el usuario y que provoca que el sistema realice la operación deseada.
Soporta un subconjunto de ANSI C así como también los tipos de datos de punto fijo (fixed point) provistos por A/RT Library o SystemC.

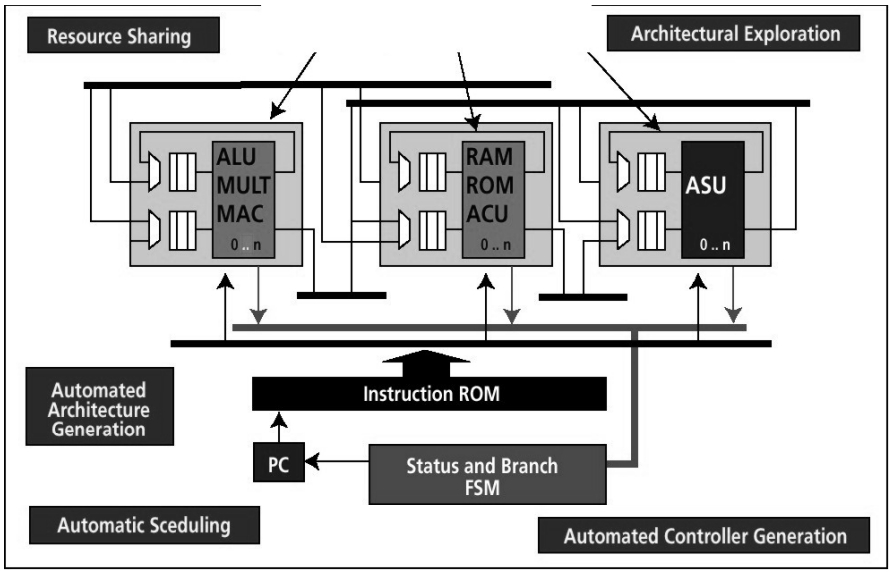



**Figura D-3** Esquema de la arquitectura generada

## D.2.3 A|RT Builder

Permite al diseñador pasar de un programa escrito en un subconjunto amplio del ANSI C a un modelo sintetizable en RTL descrito en VHDL o Verilog HDL.
El código generado es totalmente compatible con las herramientas líderes en síntesis lógica tanto para implementaciones en ASIC o FPGA. Es compatible con los tipos de datos de punto fijo y la aritmética provistos por A/RT Library y un conjunto de SystemC. Se tiene un completo control sobre las dimensiones del datapath generado, así como también de los operadores aritméticos utilizados. En la Figura D-4 podemos apreciar un esquema funcional de la máquina de estados tipo Mealy que se genera.
Además se pueden generar automáticamente bancos de prueba para verificar el HDL contra el código C.

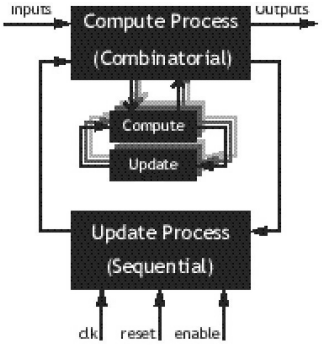



**Figura D-4** Esquema funcional del FSM tipo Mealy

## D.2.4 Situación actual

| Estado | Programa Universitario | Precios | Plataformas | Subset de ANSI C y tipos de datos principales soportados |
|---|---|---|---|---|
| Vigente | No tiene | Designer .....U$S 45.000 [13]<br>Builder .....U$S 20.000 [14]<br>Library ..... libre | HP-UX, Sun Solaris, Windows NT y Linux. | • Amplio conjunto no especificado.<br>• Números en punto fijo. |

Proyectos realizados con A|RT Builder:

- 3-Taps DSP FIR filter
- Cordic Algorithm for Inverse Tangent
- Design with Single Cycle Static RAM
- Integer Division
- Linear PCM to Alaw Converter
- Resource Shared DSP FIR filter
- Traffic Light Controller

**Obs**.: El código (C, VHDL y Verilog) de todos estos ejemplos esta disponible en la página del fabricante (http://www.frontierd.com/).

[13] Dato obtenido de artículo publicado por EETIMES el 5 de Febrero de 2001 (http://www.eetimes.com/story/OEG20010205S0075)

[14] Dato obtenido de artículo publicado por EETIMES el 9 de Marzo de 1999 (http://www.eetimes.com/story/OEG19990309S0030)

## D.3 Co-Design Automation

Fundada en 1997, a logrado conseguir reconocimiento como una empresa líder en tecnología de EDA (Electronic Design Automation) la misión de Co-Design Automation, Inc. es lograr un único lenguaje de diseño motivado a raíz de los problemas que surgieron acerca de las metodologías System-On-Chip.
En el proceso de diseño moderno encontramos una combinación de lenguajes, por ejemplo lenguajes de especificación de sistemas como SDL, lenguajes de descripción de hardware como Verilog y VHDL, lenguajes de verificación como Vera y lenguajes de programación como C. Esta mezcla provoca que el objetivo del diseño sea recodificado en diferentes lenguajes, provoca un dificultoso mantenimiento y verificación, complejas metodologías de diseño y la introducción de bugs (o errores)
Por lo tanto un solo lenguaje que cubra todos los aspectos del proceso de diseño facilita las tareas y posibilita la reutilización del código en las distintas etapas del diseño acelerando este proceso. Esta es la base para el lenguaje de diseño de sistemas SUPERLOG, que es utilizado en dos productos que produce esta compañía, Systemsim, simulador de sistemas, y Systemex, extractor de implementaciones de diseños.

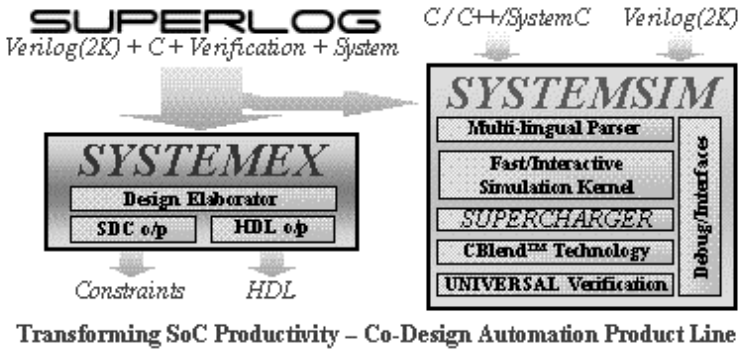



**Figura D-5** Proceso de diseño con productos de Co-Design

### D.3.1 Superlog

Es un lenguaje basado en el Verilog (95 y 2000) con la inclusión de muchos constructores de alto nivel de C y algunos con el espíritu de VHDL. Fue diseñado para interactuar con la mayoría de los HDL y con código en C y también cuenta con nuevas características que permite especificar la interconexión entre diferentes niveles de abstracción del diseño.
Es un lenguaje conciso, en oposición a muchos otros lenguajes que requieren gran cantidad de código redundante. Además incluidos en el lenguaje encontramos constructores específicos para ayudarnos con la creación de test avanzados y finalmente herramientas de chequeo de modelos.
Superlog no es estrictamente un conjunto ampliado de Verilog. Asume un modelo dirigido a eventos, el mecanismo básico de comunicación entre procesos son las variables compartidas (shared variable) pero la resolución de las funciones utiliza el modelo tipo cable (wire behavior).

Superlog acepta las siguientes características de C:

| Operadores | | Operadores Relaciónales y lógicos | Tipos de Datos | Estructura de Control |
|---|---|---|---|---|
| + | = | < | int | begin – end |
| - | &= | ! | shorint | while |
| * | != | & | longint | for |
| / | -= | && | signed | forever |
| << | += | >= | unsigned | if – else |
| >> | *= | > | Shortreal | repeat |
| ** | /= | == | Double | goto |
| | <<= | <= | Char | break – continue |
| | ^= | ^ | Enum | return |
| | >>= | ~ | Typedef | |
| | \|= | \|\| | Struct | |
| | -- | \| | Union | |
| | ++ | | Const | |
| | | | Array Multidim. | |
| | | | Globals | |
| | | | ( *casting* ) | |

## D.3.2 Systemsim

Herramienta de simulación de última generación que apunta a la verificación de sistemas, que permite realizar simulaciones mixtas de sistemas escritos en Superlog, Verilog (2000), C/C++ y SystemC.
La tecnología CBlend permite mezclar libre, simple y eficientemente el Verilog y C/C++. El motor SPUERCHARGER hace posible acelerar el código compilado manteniendo la flexibilidad para depurar (debug)
A través de la potencia de UNIVERSAL (ver Figura D-6) combinado con SUPERLOG, Systemsim provee una forma rápida y simple de construcción automática de testbench al estilo Verilog.

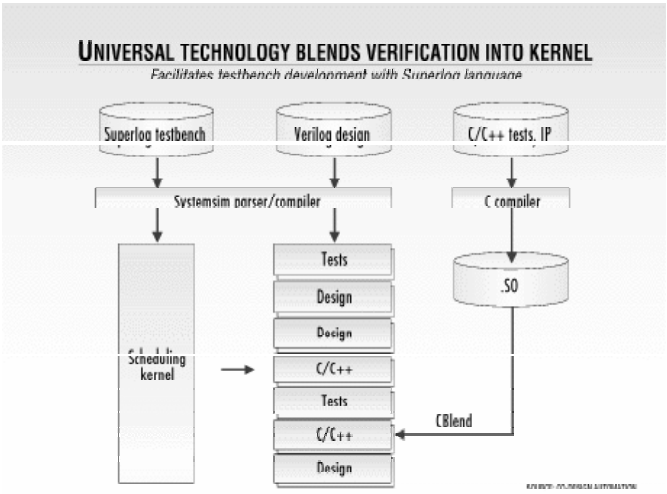



**Figura D-6** Descripción funcional de UNIVERSAL Verification

### D.3.3 Systemex

Esta herramienta extrae implementaciones HDL de los modelos en Superlog para facilitar el pasaje al silicio usando herramientas estándar de síntesis.

### D.3.4 Situación Actual

| Estado | Programa Universitario | Precios | Plataformas | Subset de ANSI C y tipos de datos principales soportados |
|---|---|---|---|---|
| Vigente | No tiene | Systemsim.....U$S 18.200<br>Systemex……U$S 25.000 [15] | Sun Solaris y Linux. | • Gran parte del ANSI C.<br>• No soporta punto flotante ni punteros. |

[15] Obtenido de un artículo publicado por EETIMES el 8 de Mayo de 2000 (http://www.eetimes.com/story/OEG20000508S0076)

## D.4 Forte Design Systems

Esta empresa se dedica exclusivamente a cubrir las necesidades de diseño y verificación de sistemas eléctricos tipo Gigascale, éstos poseen características y requerimientos que generalmente incluyen un alto flujo de datos, velocidades grandes de simulación, su implementación requiere de alrededor de 1000 millones de compuertas, múltiples lenguajes de diseño y verificación y procesadores embebidos. Una forma de explorar las múltiples implementaciones de un algoritmo es utilizando el Cynthesizer que las genera automáticamente utilizando diferentes recursos y restricciones de tiempos. Veamos a continuación una descripción de las distintas herramientas ofrecidas por el fabricante.

### D.4.1 Cynlib

Es un conjunto de clases de C++ que implementan muchas de las características de Verilog y VHDL, es un vocabulario para el modelado de hardware desde C++. Esta librería pretende crear un entorno donde el hardware y su entorno de test puedan ser modelados y simulados.

Cynlib soporta el desarrollo de hardware en un ambiente C++. Para esto Cynlib extiende la capacidad de C/C++ implementando las siguientes características:

- Modelo de ejecución concurrente
- Basado en ciclos
- Módulos
- Puertos
- Threads (Hilos)
- Asignaciones diferidas
- Valores de datos con dos estados
- Múltiples clocks
- Soporte para debug

### D.4.2 Cyn++

Facilita la tarea de escribir código en Cynlib/C++ resumiendo algunas de las características más verbosas de C++ en un conjunto de macros simples y convenientes. Los números de línea del código y otras informaciones de debug son preservadas facilitando la tarea de depuración ya sea utilizando CynGDB, GDB o cualquier otro debugger.

Cyn++ soporta el siguiente juego de macros:

- Module/Endmodule
- Always/Initial Blocks
- Posedge/Negedge
- Changed
- Verilog-style module instantiation
- Dynamic module instantiation

Cada uno de estos macros provee funcionalidad similar a las estructuras de Verilog sugeridas por los nombres de los macros.
Cyn++ trabaja de forma transparente con las demás herramientas ofrecidas en el paquete Cynlib Tool Suite. Es la herramienta ideal para los diseñadores que utilicen Verilog y que desean trasladarse sin demasiadas complicaciones a trabajar en un entorno de C++.

### D.4.3 Cynchronizer

Es una herramienta de traducción que toma un diseño en Verilog o VHDL y genera un código C++ usando Cynlib. Al usar las clases de Cynlib el código C++ se comporta justo como hardware: precisión de bit en puertos y señales, procesos concurrentes, comportamientos síncronos y asíncronos e instancias jerárquicas. Además conserva los nombres originales de señales, etiquetas y comentarios.
Es una herramienta muy útil si desea comenzar a trabajar con C++ y reutilizar módulos previamente programados en algún lenguaje de HDL.

### D.4.4 Cynthesizer

Toma un diseño escrito en Cynlib y produce una representación equivalente en Verilog o VHDL, manteniendo la precisión del diseño, pronta para ser utilizada en un amplio rango de sintetizadores tanto de ASIC como de FPGA. Las estructuras de datos de alto nivel, como las clases de C++, son automáticamente transformadas en funcionalmente equivalentes Verilog o VHDL. Otras características del lenguaje orientado a objetos, como la herencia, son automáticamente resueltas en el Verilog o VHDL resultante. Además una serie de inteligentes transformaciones son hechas al código para hacerlo más digerible por las herramientas de síntesis.
Los grandes beneficios de esta herramienta son la obtención de una implementación para hardware lo más rápido posible, y además la posibilidad de utilizar la programación orientada a objetos que produce un diseño mas claro y natural facilitando su comprensión.

### D.4.5 Cynware

Es una colección de clases usadas para modelar sistemas permitiendo al diseñador abstraerse de los detalles del diseño. Cynware actualmente cuenta con tres clases:

- **Abstract interfaces** para el modelado de interfaces entre módulos o procesos. En el mas alto nivel estas pueden ser usadas para modelar relaciones de flujo de datos abstractos (abstract data flow relationships) o espacios de direcciones compartidos (shared address space). En el mas bajo nivel se pueden usar para modelar el comportamiento de interfaces con precisión de ciclo (cycle accurate interface behavior).
- **FIFO classes** para modelar diferentes tipos de FIFOs. Estas clases se pueden usar en muchas formas dentro o entre procesos. Se pueden usar en modelos muy abstractos o en modelos de precisión de ciclo.
- **Pipe classes** para modelar latencia.

### D.4.6 Cyntax

Es una herramienta que ayuda en el desarrollo y debug de los diseños basados en Cynlib. Cyntax rápidamente busca y analiza programas de Cynlib buscando errores comunes de diseño, proporcionando la misma funcionalidad que *lint* para programas basados en C.

### D.4.7 CynGDB

Herramienta para depurar programas escritos en Cynlib y Cyn++ derivada del clásico paquete GNU GDB.

### D.4.8 Situación Actual

<table>
<tr><th>Estado</th><th>Programa Universitario</th><th colspan="2">Precios</th><th>Plataformas</th><th>Subset de ANSI C y tipos de datos principales soportados</th></tr>
<tr><td rowspan="2">Vigente</td><td rowspan="2">No tiene</td><td>Cynlib<br>Cyn++<br>CynGDB</td><td>libre</td><td rowspan="2">No especifica.</td><td rowspan="2">No especifica</td></tr>
<tr><td>Cynchronizer<br>Cynthesizer<br>Cyntax<br>Cynware</td><td>+ U$S 40.800 [16]</td></tr>
</table>

[16] Obtenido de artículo publicado por EETIMES el 19 de Noviembre de 2001 (http://www.eetimes.com/story/OEG20011119S0025)

## D.5 IMEC (Interuniversity Micro Electronics Center)

El IMEC desarrollo el OCAPI-xl, es una librería de clases de C++ cuyo objetivo es el diseño de sistemas para diferentes arquitecturas de hardware y software (codiseño). Es un modelo a nivel de sistema en el que la funcionalidad y la arquitectura se pueden describir separadamente, permitiendo hacer modelados a alto nivel de abstracción. Una estrategia de refinamiento y unas especificaciones ejecutables a todos los niveles permiten un camino estructurado hacia la implementación.

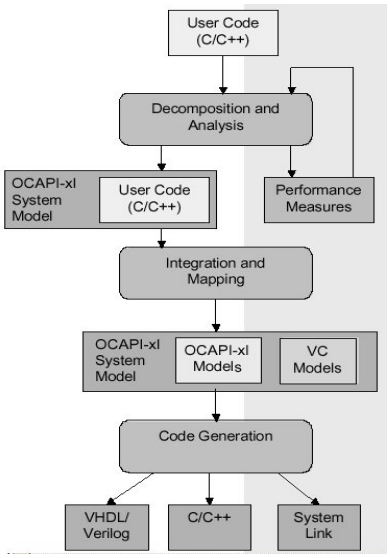



**Figura D-7** Proceso de diseño con OCAPI-xl

El modelo ejecutable contiene la funcionalidad especificada en una forma independiente del objetivo. También puede contener las propiedades de la arquitectura, pero ambas son especificadas en una forma ortogonal.
Durante la exploración del espacio de diseño, OCAPI-xl da al diseñador una realimentación basada en simulación de la performance. Esto permite explorar múltiples alternativas de particionado en hardware / software y tomar una decisión bien fundada para la implementación final.

**Características principales**

- Estilo de modelado unificado en C++ para representar funcionalidad a diferentes niveles de abstracción independiente de la arquitectura o las decisiones de implementación.
- Modelado de sistemas consistente en procesos paralelos comunicados. Para el comportamiento en el dominio del tiempo se puede escoger entre múltiples modelos para cada proceso a varios niveles de precisión.
- Tiene un conjunto de primitivas de comunicación, tipo software, de alto nivel que permiten describir un gran rango de aplicaciones y que pueden ser implementadas tanto en hardware como en software.
- El mapeado y remapeado de la funcionalidad en la arquitectura no requiere rescribir el código. Por lo tanto la exploración del particionado puede ser fácilmente llevada a cabo. Para ayudar al diseñador se genera un reporte de performance, basado en simulación, en formato HTML. Este reporte es detallado y a la vez fácil de usar.

- El camino del modelo unificado hacia la implementación yace en un generador automático de código. Las partes de hardware resultan en un sintetizable HDL de transferencia de registros (VHDL o Verilog). Las partes de software son volcadas en ANSI-C requiriendo un mínimo OS-API para threads y comunicación de threads.

## D.5.1 Situación Actual

| Estado | Programa Universitario | Precios | Plataformas | Subset de ANSI C y tipos de datos principales soportados |
|---|---|---|---|---|
| • En proceso de juntarse con SystemC.<br>• Licenciado por CoWare Inc. | No tiene | No específica. | No específica. | No específica |

**Proyectos realizados con OCAPI-xl:**

- Gecko. Demostrador de multitarea Hardware / Software sobre plataformas reconfigurables (http://www.imec.be/design/reconfig/gecko.shtml)
- Hipermac. (http://www.imec.be/design/M4/hipermac.shtml)
- Cam-E-Leon. Cámara web con hardware reconfigurable. (http://www.imec.be/design/reconfig/cameleon.shtml)
- Go4Net. Cámara conectable directamente a red Ethernet. (http://www.imec.be/design/reconfig/go4net.shtml)
- 80 KGate upstream cable modem
- 75 KGate DECT transceiver
- 40 KGate MPEG-4 image decoder
- 100 Mbit/s OFDM WLAN Mode

## D.6 SYNOPSYS

Esta empresa es líder en herramientas para el diseño de circuitos integrados complejos. Dentro de la amplia gama de productos desarrollados podemos apreciar en la Figura D-8 un subconjunto de ellos que aborda la problemática planteada por este proyecto.

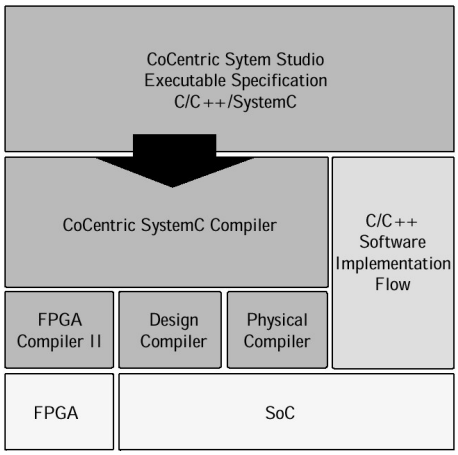



**Figura D-8** Proceso de diseño con CoCentric System Studio Package

A continuación veamos una breve descripción de la funcionalidad de cada uno de ellos.

### D.6.1 CoCentric SystemC Compiler

Este producto sintetiza hardware a partir de código en SystemC. Proporciona un camino rápido y de alta calidad desde una descripción en C/C++ hacia compuertas o a una descripción sintetizable de Verilog o VHDL.

**Características**

- Síntesis automática de hardware partiendo de código en SystemC.
- Fácil de usar.
- Análisis arquitectural gráfico.
- Single pass timing closure.
- Caminos de implementación hacia FPGAs y/o System on a chip.
- Integración estrecha con otras herramientas de Synopsys.

**Beneficios**

- Evitar proceso de conversión a HDL que es fuente de errores y consume tiempo.
- Ir al hardware mucho mas rápido.
- Determinar los cuellos de botella en el diseño para aumentar performance y/o área.
- Satisfacer restricciones de tiempos y área más rápidamente.
- Evitar el pasaje de test bench escritos en C/C++ a HDL para verificar nuestro diseño.

## D.6.2 Cocentric System Studio

Ambiente de especificación y simulación para verificación y análisis conjunto de algoritmos, arquitectura, hardware y software a múltiples niveles de abstracción. System Studio cuenta con 2000 modelos de algoritmos frecuentemente usados en su librería de modelos. Una rápida verificación de cumplimiento de normas es posible en los sectores de telecomunicaciones inalámbricas, multimedia, redes de computadoras por medio de una librería de diseños de referencia. System Studio también se usa para crear código SystemC sintetizable para el SystemC Compiler. Los diseños son capturados, verificados y optimizados utilizando un intuitivo grafo de flujo de datos y máquinas de estados finitas (FSM).

## D.6.3 CoCentric Fixed-Point Designer

Esta herramienta de software que tiene dos funciones principales:

- leer código ANSI-C que contiene tipos de datos floating-point, entender el código, automáticamente generar una nueva versión de código fuente sustituyendo los tipos floating-point con los tipos de datos públicos de SystemC fixed-point (punto fijo)
- proveer un conjunto de funciones para asistir al diseñador a personalizar y ajustar (custom-fit) el diseño como le parezca. La herramienta provee información necesaria que el diseñador necesita para implementar hardware ASICs o desarrollar fixed-point DSP software mas rapidamente y con mayor seguridad.

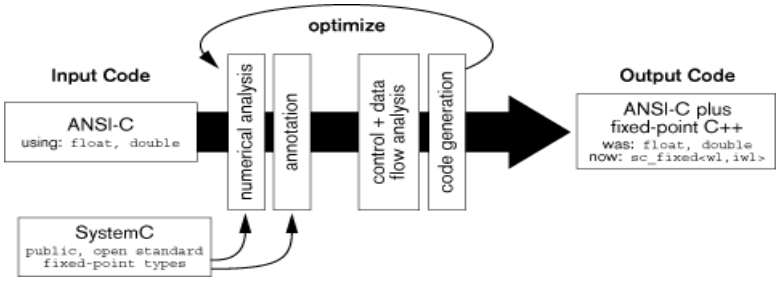



**Figura D-9** Diagrama flujo para el Fixed-Point Designer

## D.6.4 Situación Actual

| Estado | Programa Universitario [17] | Precios | Plataformas | Subset de ANSI C y tipos de datos principales soportados |
|---|---|---|---|---|
| Vigente | CoCentric System Studio Package: 20 o 50 licencias de 1 año | U$S 500 | HP-UX, Linux, and Solaris. | Punto Fijo y Punto Flotante |
| | Paquete universitario (incluye SystemC compiler):<br>20 licencias de 1 año ----------<br>50 licencias de 1 año ---------- | <br>U$S 2.500<br>U$S 3.500 | | |

[17] Por mas detalles ver http://www.synopsys.com/partners/university/tools.html

## D.7 University of Toronto

El departamento de Ingenieria Eléctrica y en Computación desarrollaron el compilador llamado Transmogrifier C (o TMCC), su nombre es tomado de la placa Transmogrifier 1 (o TM-1) armada también esta universidad y que consta de cuatro Xilinx XC4010, dos Aprix AX1024 y cuatro SRAM de 32Kx9.
Otros grupos de trabajo desarrollan otras herramientas similares al TMCC como ser el compilador llamado HardwareC parte del sistema de síntesis Olympus [14] , pero en éste se realizaron grandes cambios de sintaxis y semántica para soportar una síntesis de alto nivel.

El Transmogrifier C acepta un subconjunto del lenguaje C y genera un netlist (XNF) apto para ser utilizado en los chips de la familia XC4000 de Xilinix. Los tipos de datos soportados son los enteros sin signo (de tamaño definido por el programador), no soporta punteros, arrays o tipos estructurados de datos. Soporta variables locales, expresiones y constantes, sentencias de bifurcación como ser IF, construcción de bucles tipo WHILE y llamadas a funciones. No posee implementado divisiones o productos.
La última versión desarrollada es la 3.1 del 10 de enero de 1996.

### D.7.1 Situación Actual

| Estado | Programa Universitario | Precios | Plataformas | Subset de ANSI C y tipos de datos principales soportados |
|---|---|---|---|---|
| Discontinuado 1996 | No aplicable | Libre | Se poseen las fuentes para compilar. | Instrucciones de bifurcación, de bucle y llamadas a funciones.<br>Números Enteros.<br>No soporta producto y división. |

# D.8 Stanford University

El grupo Stanford Compiler Group de esta universidad ha desarrollo el compilador llamado SUIF (Stanford University Intermediate Format) que es un infraestructura libre diseñada para soportar investigación colaborativa en optimizado y paralelizado de compiladores.
SUIF consiste en un núcleo (kernel) pequeño, claramente documentado y un conjunto de *herramientas* de distintas etapas del proceso de compilación reunidas y accesibles desde un kernel principal. Este kernel define la representación intermedia, provee funciones de acceso y manipulación de esa representación y estructura la interfase entre las distintas pasadas de compilación.
Las *herramientas*, como mencionamos antes, consisten en diferentes programas que implementan las distintas etapas del proceso de compilación, las que están actualmente incluidas son front ends para C y Fortran, buscador de loops paralelizables, un optimizador local, conjunto de herramientas de desarrollo del compilador. Típicamente cada etapa de un compilador lleva a cabo solamente un análisis o transformación y escribe sus resultados en un archivo de salida, esto es claramente ineficiente pero muy flexible, el SUIF utiliza el mismo formato de archivo de salida para las diferentes etapas gracias a ello se puede cambiar libremente el orden de ejecución de ellas así como también agregar nuevas en cualquier punto del proceso de compilación.
El sistema incluye muchas características para soportar paralelización como análisis de dependencia de datos, reconocedor de reducciones o simplificaciones, un conjunto de análisis simbólicos que mejoran la detección del paralelismo y transformaciones *unimodular* para incrementar el paralelismo y la localidad.

Dentro del paquete contamos además con un conversor SUIF to C que nos permite compilar el código paralelizado en cualquier compilador paralelo y por tanto en cualquier plataforma.

La versión 2 del SUIF además incluye un front end para Java y extensiones para Orientación a objetos.

Actualmente no hay ningún módulo para pasar de C a VHDL o mejor dicho del formato intermedio generado hacia VHDL, pero se podría utilizar en conjunto con alguna de las dos herramientas que veremos en los puntos siguientes: Mini Proyecto SUIF to VHDL translator y SiliconC.

## D.8.1 Situación Actual

| Estado | Programa Universitario | Precios | Plataformas | Subset de ANSI C y tipos de datos principales soportados |
|---|---|---|---|---|
| Vigente | No aplicable | libre | Se poseen la fuentes para compilar. | ANSI C |

## D.9 Mini Proyecto. Suif to VHDL translator

Proyecto de fuente abierta del Departamento de ciencias de la computación e Ingenieria del Instituto Indio de Tecnología (Delhi) es parte del proyecto mayor "Methodology for Heterogeneous Implementation of Real-Time Embedded Systems for Vision/Image Processing" cuyo objetivo es desarrollar una metodología de diseño para sistemas embebidos.
El objetivo del miniproyecto era desarrollar una herramienta de conversión de SUIF a VHDL. Dada una especificación en C se usa el compilador de SUIF, luego se pueden aplicar diversos algoritmos para marcar cuales nodos se implementaran en hardware y cuales en software. Después de esta partición la parte de hardware se convierte en VHDL mediante la herramienta que se desarrollo en el proyecto.

Características del C que soporta:

| Operadores | Operadores Relacionales y lógicos | Tipos de Datos | Estructura de Control |
|---|---|---|---|
| Todas las operaciones aritméticas | Todos los operadores | integer<br>float<br>characters<br>arrays<br>pointers | begin – end<br>while – do<br>do – while<br>for<br>if – then – else<br>repeat<br>repeat – until<br>switch – case |

En este proyecto no todas las instrucciones de SUIF fueron tenidas en cuenta, solo aquellas necesarias para implementar los tipos y construcciones que se querían manejar. Se basa en el formato intermedio generado y en particular utiliza el árbol sintáctico como entrada para el programa que se encarga de la translación a VHDL, pero el código generado no es sintetizable.

### D.9.1 Situación Actual

| **Estado** | **Programa Universitario** | **Precios** | **Plataformas** | **Subset de ANSI C y tipos de datos principales soportados** |
|---|---|---|---|---|
| Terminado 1999 | No aplicable | libre | Se poseen la fuentes para compilar. | • ANSI C<br>• Enteros, punto flotante, char, array y punteros |

# D.10 SILICON C

Compilador de C a hardware usando el sistema de compilación de SUIF (Stanford University Intermediate Format). La traducción de C a VHDL se da en un pipeline de seis etapas, como se puede apreciar en la Figura D-10, usando dos librerías de código para extender la base del sistema SUIF.

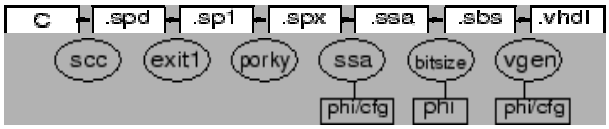



**Figura D-10** Etapas del compilador de C a VHDL

Excepto por el primero y el último cada módulo lee y escribe la representación intermedia SUIF, anotada con varias piezas de información coleccionadas de cada etapa.
Se realiza propagación de constantes, eliminación de sub-expresiones y eliminación de código muerto, para reducir la cantidad de lógica generada.
El módulo bitsize intenta estrechar el ancho de bit de las variables para generar datapaths más eficientes.
El módulo final, vgen, genera código VHDL estructural que es solamente una notación conveniente para describir lógica a nivel de módulos y netlist.

**Limitaciones**

Esta implementación no soporta punto flotante. Tampoco soporta división de enteros, la multiplicación si la soporta pero advierte que no es buena idea usarla. Arreglos, punteros y variables estáticas tampoco son manejados.

## D.10.1 Situación Actual

| Estado | Programa Universitario | Precios | Plataformas | Subset de ANSI C y tipos de datos principales soportados |
|---|---|---|---|---|
| Terminado 1998 | No aplicable | libre | No se poseen las fuentes. | No soporta punto flotante, arrays ni punteros.<br>No soporta división |

## D.11 Oxford University

El laboratorio de Computación de esta universidad en el año 1997 ha desarrollado un compilador llamado Handel C (o HCC), producto de un largo programa de investigación que analiza el codiseño de sistemas.
Handel C, en su versión 0.1 del 2 enero de 1997, no es un lenguaje de descripción de hardware, se basa en un subconjunto sencillo del lenguaje ANSI C, y está diseñado para permitir la compilación de programas (algoritmos) a implementaciones de hardware síncronas (usualmente basadas en FPGA). También se agregan unas extensiones del lenguaje que permiten paralelizar secciones de código, otras que facilitan las comunicaciones entre procesos manejando un protocolo sencillo y tipos que nos permiten definir variables ubicadas en RAM o ROM tanto internas como externas. Se aceptan todas las directivas del pre-procesador (cpp) de C..
Acepta definiciones de procesos pero sin parámetros de entrada o salida, la comunicación de valores debe hacerse mediante canales o variables globales y solamente la ejecución de un solo proceso es permitida por estado. Para las variables tipo RAM o ROM se restringe también el acceso a una sola dirección por estado.

Características de C que son soportadas:

| Operadores | Operadores Relacionales y lógicos | | Tipos de Datos | Estructura de Control |
|---|---|---|---|---|
| + | > | != | boolean | while – do |
| - | < | & | Void | do – while |
| * | >= | \| | Int | for |
| >> | <= | ^ | Char | if – then – else |
| << | == | ~ | Const | switch – case |
| ?: | | | | { ; } |

Como funcionalidades específicas tenemos que el ancho de las variables es definido en el momento de su declaración o sino es inferido del contexto durante la compilación del programa. Los enteros definidos son considerados con signo y en representación de complemento a dos, las operaciones aritméticas y los operadores relacionales se pueden definir para trabajar con y sin signo.

### D.11.1 Situación Actual

| **Estado** | **Programa Universitario** | **Precios** | **Plataformas** | **Subset de ANSI C y tipos de datos principales soportados** |
|---|---|---|---|---|
| Vigente [18] | No aplicable | libre | MS-DOS | No soporta punto flotante, arrays multidim. ni punteros.<br>No soporta división. |

[18] Actualmente licenciado y desarrollado por Celoxica. (http://www.celoxica.com)

# D.12 Celoxica

Celoxica aplica tecnologías de software al diseño de hardware con herramientas que hacen converger la automatización del diseño de electrónica (EDA) y desarrollo de software embebido (ESD) para mejorar la productividad del diseñador, y superar cuellos de botella en la performance del sistema.

Las herramientas de diseño de la compañía, y los productos y servicios de soporte, introducen tres aspectos del desarrollo de software al proceso de diseño de hardware; un lenguaje basado en C para describir rápidamente la funcionalidad, en lugar del detalle estructural del hardware necesario, un ambiente integrado de desarrollo para le diseño de hardware (IDE) con características tanto de debugging simbólico como síntesis que se correlaciona con la compilación de software en que es muy rápido; y bibliotecas de funciones predefinidas incluyendo el acceso a los periféricos y procesadores en hardware por medio de APIs comunes.

Su núcleo es el compilador Handel – C, cuya versión original tratamos en el punto anterior.

## D.12.1 DK1 Design Suite

Mantiene las características principales de su predecesor desarrollado por la universidad de Oxford

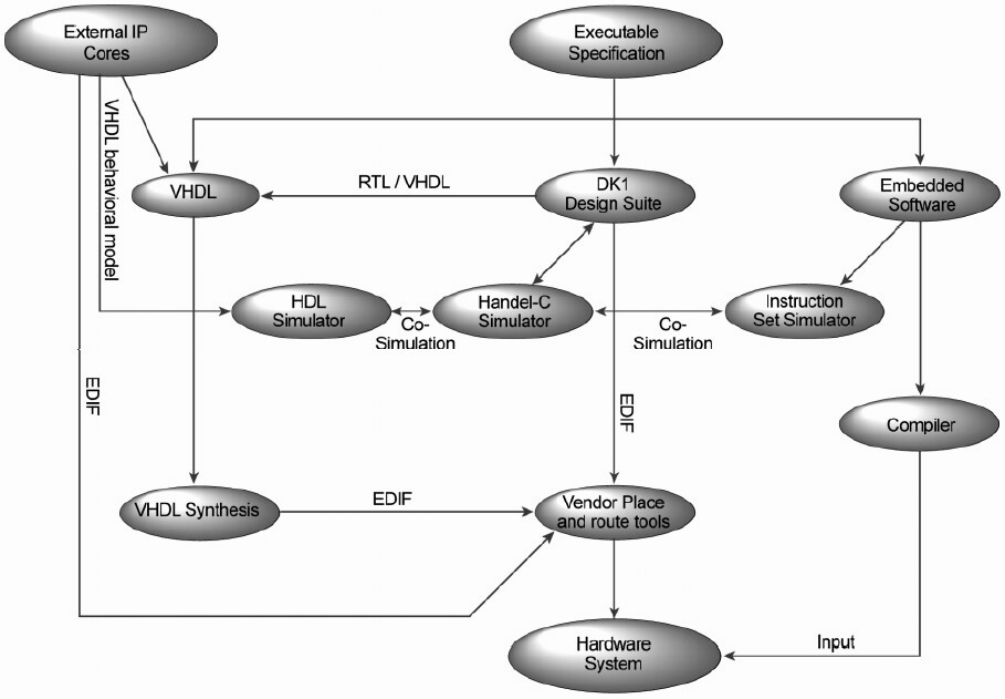



**Figura D-11** Entorno de diseño con DK1 Design Suite

Veamos que características del ANSI C son soportadas:

| Operadores | Operadores Relacionales y lógicos | Tipos de Datos | Estructura de Control |
|---|---|---|---|
| + | > | booleanos | while – do |
| - | < | Void | do – while |
| * | >= | Int | for |
| / | <= | char | if – then – else |
| % | == | Unsigned | switch – case |
| >> | != | signed | break |
| << | & | short | continue |
| ?: | && | long | goto |
| ++ | \| | Const | return |
| -- | \|\| | typedef | { ; } |
| += | ^ | struc | |
| -= | ! | enum | |
| *= | ~ | extern | |
| /= | | static | |
| %= | | volatile | |
| >>= | | | |
| <<= | | | |
| &= | | | |
| \|= | | | |
| ^= | | | |

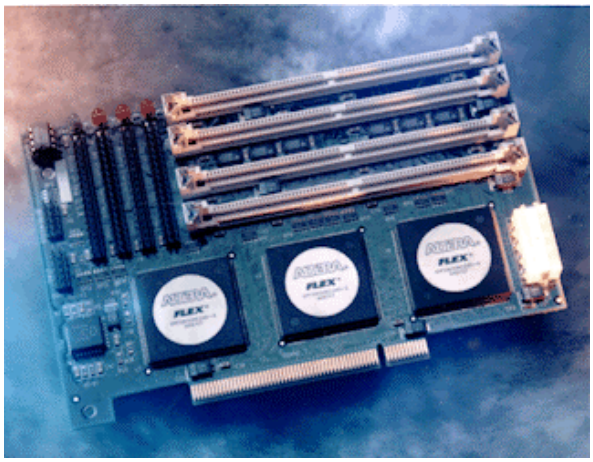
## D.12.2 Situación Actual

| **Estado** | **Programa Universitario** | **Precios** | **Plataformas** | **Subset de ANSI C y tipos de datos principales soportados** |
|---|---|---|---|---|
| Vigente | 1 licencia flotante de 1 año | U$S 300 | Windows NT, 2K, XP | No soporta Punto flotante, punteros. |

# E Datos de placas y chips

## E.1 Tarjeta ARC-PCI (Altera Reconfigurable Computer with PCI interface)

La tarjeta contiene:



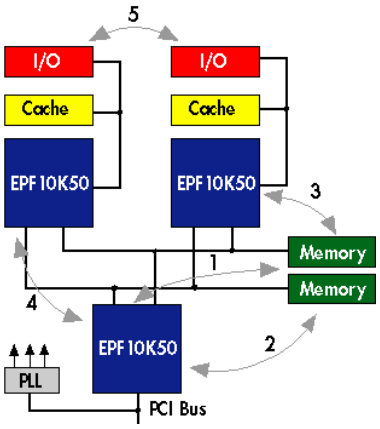
- Tres chip EPF10K50
- Un controlador del sistema PLD
- Bus PCI, PLD de configuración
- Memoria y I/O Interface
- Dos PLDs configurables
- Cada uno tiene 2.880 elementos lógicos y 20Kb de RAM
- Alta velocidad de configuración
- 4 conectores de 72 bit de RAM de 1 M x 32 bits
- PLL (Phese -locked loop) en la generación de reloj

**Altera al diseñar la placa se enfoco los siguientes objetivos:**

- Fácil uso
- La arquitectura simple permite más fácil combinar un algoritmo al conjunto de PLDs
- Reconfiguración parcial, permite reconfigurar cada dispositivo.
- Alta velocidad de transferencia de memoria de la máquina a memoria de la placa a través del bus PCI
- Alta velocidad de configuración de PLD
- Alta capacidad y bajo tiempo de acceso a la memoria local

### Camino del Flujo de datos

1. Transferencia de datos entre la memoria local y memoria del PC a traves del bus PCI.
2. Configuracion con los datos transferidos desde la memoria hacia el PLD
3. Read/White desde la memoria al PLDs
4. Transferecia de datos desde el PLD hacia el bus PCI
5. Transferecia de datos desde un PLD hacia otro



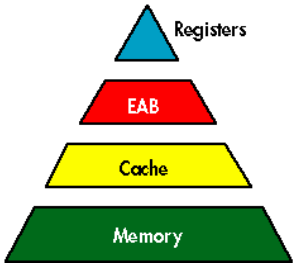

### Reloj

La tarjeta utiliza el reloj de 33 MHz del bus PCI, pero consta de un divisor por 2 y un multiplicador por 2, que resulta en obtener dos señales extra de reloj de 16.5 MHz y otra de 66 MHz.

### Organización y jerarquía de la memoria

La tarjeta tiene 4 niveles de jerarquía de la memoria

- Registros PLD
- PLD EABs
- Cache
- Memoria, los tres PLDs tienen acceso directo a las dos SRAM.



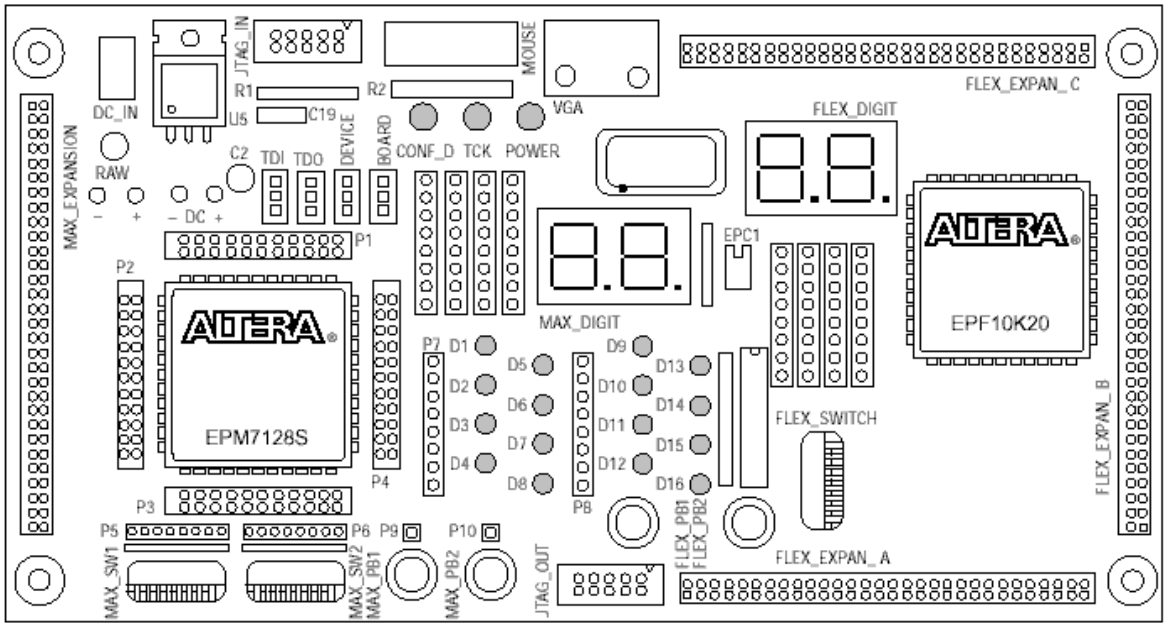

El árbitro de las memoria es realizado por el PLD controlador del sistema, pero si en los PLD configurables existe suficiente espacio, estos pueden realizar la tarea, utilizando para eso 5 pins de I/O por PLD

| Type | Size |
|---|---|
| **Registros** | 3,184 bits cada chip EPF10K50 |
| **EAB** | 20,480 bits cada chip EPF10K50 |
| **Cache** | 1 M x 32 bits cada SRAM SIMM |
| **Memoria** | 1 M x 32 bits cada SRAM SIMM |

## E.2 Placa UP1

Es una tarjeta que contiene dos chip de altera, uno de la familia MAX7000 (EPM7128s) y otro de la familia FLEX 10k (EPF10K20). Para la programación de ambas se puede hacer a través de "ByteBlaster download cable" y la EPF10K20 también a tarves de "EPC1 configuration EPROM".

Placa UP 1




La tarjeta contiene dos chips programables, ellos son:

- **Chip EPM7128S** es un chip de mediana densidad y alta performance. Basada en elementos de EEPROM, cuenta con 84 pines y 128 macroceldas. Cada macrocelda es configurada con arreglos de and-or y con registros configurables con clock, clock eneble, clerr y present independientes. Tiene una capacidad de hasta 2500 puertas.

  **Recursos de la placa para este chip:**
  - 84 pines motados sobre la placa
  - 2 push-button switches
  - 2 octal dipswitches
  - 16 leds
  - 2 display de 7 segmentos
  - Oscilador on-board (25.175 MHz)
  - Expansión con 42 puertos I/O y 4 pines globales

- **Chip EPF10K20** es un chip de alta densidad, basado sobre elementos SRAM reconfigurables, cuenta con 240 pines, 1152 celdas lógicas (LEs) y 6 "embedded array bolcks (EABs)". Cada LE contiene 4 entradas "look-up table" (LUT), un flip-flop programable y señales dedicadas a carry y cascada de funciones. Cada EAB esta provisto de 2.048 bits de memoria que pueden ser usados como RAM, ROM o FIFO, pueden ser utilizadas para implementar funciones lógicas como multiplicador, microcontroladores, maquinas de estado, procesador digital de señales (DSP). Con las 20.000 puertas, es ideal par diseños avanzados.

  **Recursos de la placa para este chip:**
  - 2 push-button switches
  - 1 octal dipswitches
  - 2 display de 7 segmentos
  - Oscilador on-board (25.175 MHz)
  - 3 Expansiones con 42 puertos I/O y 7 pines globales
  - Puerto VGA
  - Puerto Mouse

Además contamos con:

- **ByteBlaster Parallel Port Download Cable:** Utilizado para bajar los diseños hacia la placa, a través del puerto paralelo del PC.
- **Conectores para fuente y reguladores de voltaje**
- **Oscilador:** La UP1 consta con un cristal oscilador de 25.175 MHz, esta señal entra a ambos chip y controla el "global clock"
- **Jumpers:** Utilizados para setear como quiero trabajar, programar solo una de las placas, ambas o para conectar múltiples placas UP1.

# E.3 Familia Flex10k

## Especificaciones:

- Soporte MultiVoltaje en la interface I/O
- Tolera 5V en los pines de entrada
- Bajo consumo
- Soportan componentes periféricos interconectados con PCI
- Operan con 3.3 V o 5.0V según el chip

**Tabla E-1** Características de la familia FLEX10K

| **Dispositivos** | **EPF10K10 EPF10K10A** | **EPF10K20** | **EPF10K30 EPF10K30A** | **EPF10K40** | **EPF10K50 EPF10K50V** | **EPF10K70** |
|---|---|---|---|---|---|---|
| Typical gates (logic and RAM) | 10.000 | 20.000 | 30.000 | 40.000 | 50.000 | 70.000 |
| Maximum system gates | 31.000 | 63.000 | 69.000 | 93.000 | 116.000 | 118.000 |
| Logic elements (LEs) | 576 | 1.152 | 1.728 | 2.304 | 2.880 | 3.744 |
| Logic array blocks (LABs) | 72 | 144 | 216 | 288 | 360 | 468 |
| Embedded array blocks (EABs) | 3 | 6 | 6 | 8 | 10 | 9 |
| Total RAM bits | 6.144 | 12.288 | 12.288 | 16.384 | 20.480 | 18.432 |
| Maximum user I/O pins | 150 | 189 | 246 | 189 | 310 | 358 |

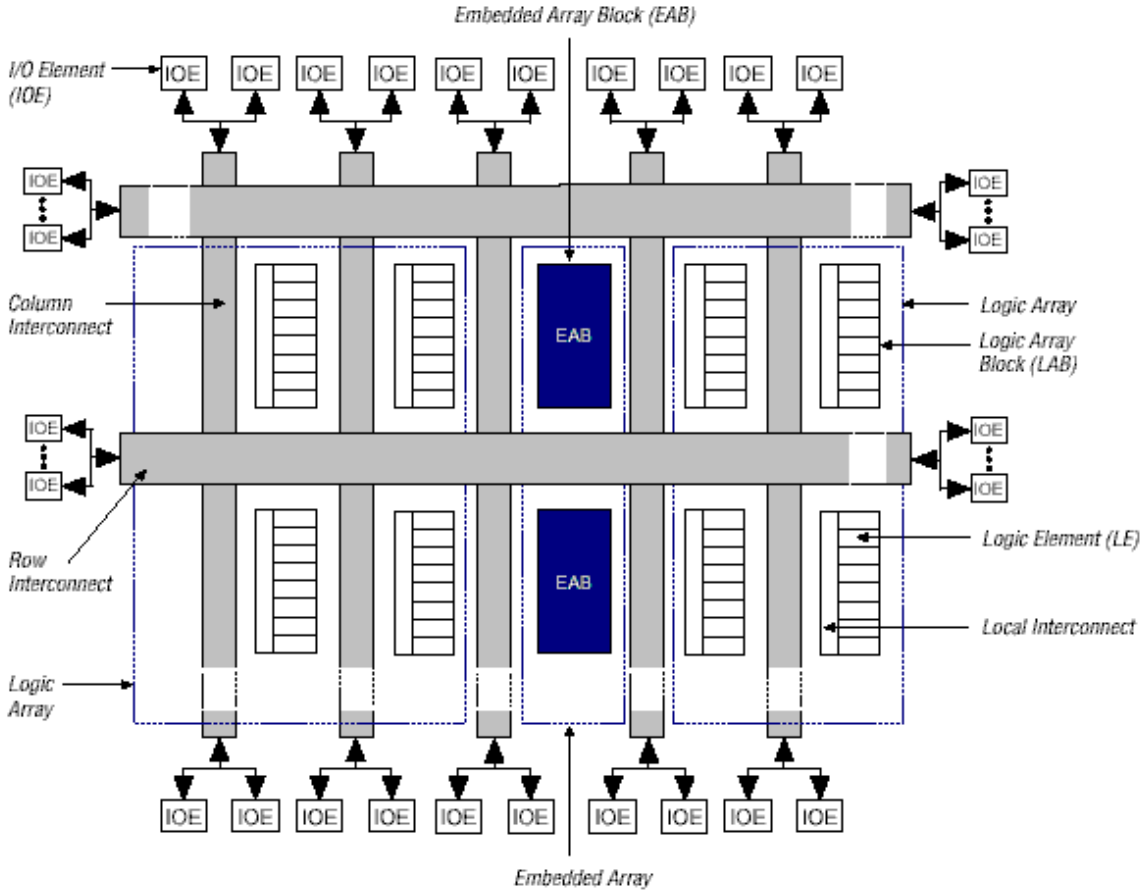



**Figura E-1** Estructura general de las FPGA de la familia FLEX10K.

## E.4 Familia Xc4000

### Características

- On-chip RAM
- Lógica de Carry dedicada de alta velocidad
- Capacidad para buses triestado internos
- Resistencias pull-up y pull-down programables en la entrada
- Entre 64 y 3136 CLBs que se traducen en 1000 a 185000 compuertas, considerando 30 % de los CLB usados como RAM.

Los dispositivos de la serie 4000 son implementados con una arquitectura flexible y programable de CLBs (Figura E-3 ) interconectados por una poderosa jerarquía de recursos de ruteo versatiles y rodeada por un perímetro de IOBs programables ( Figura E-4 ).

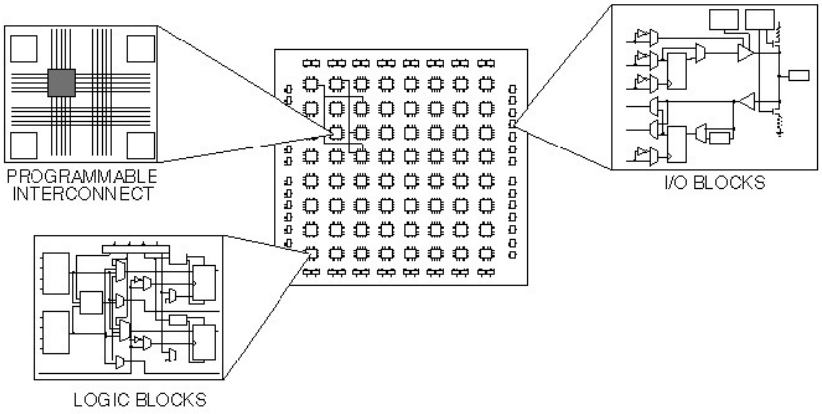



**Figura E-2** Estructura general de las FPGA de la serie 4000.

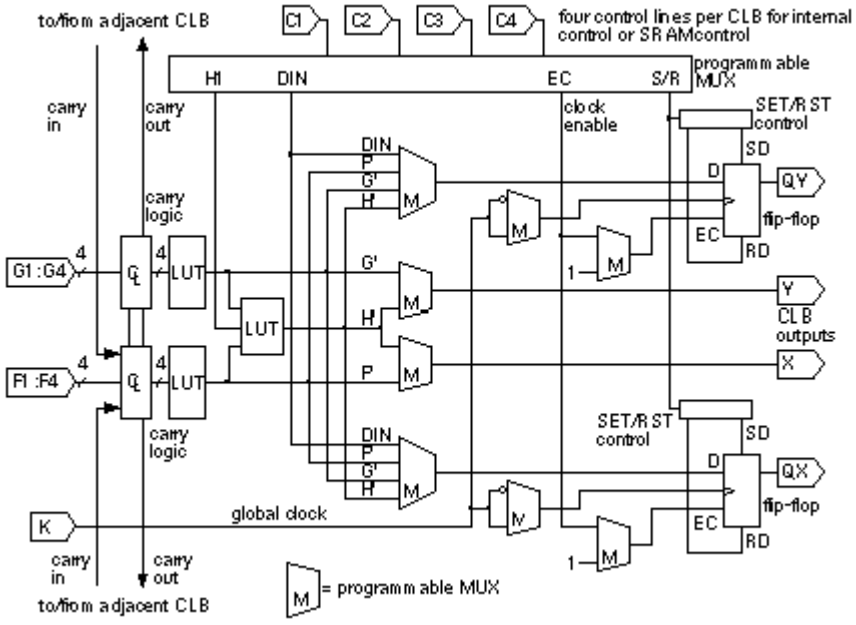



**Figura E-3** Esquema de un CLB de la serie 4000.

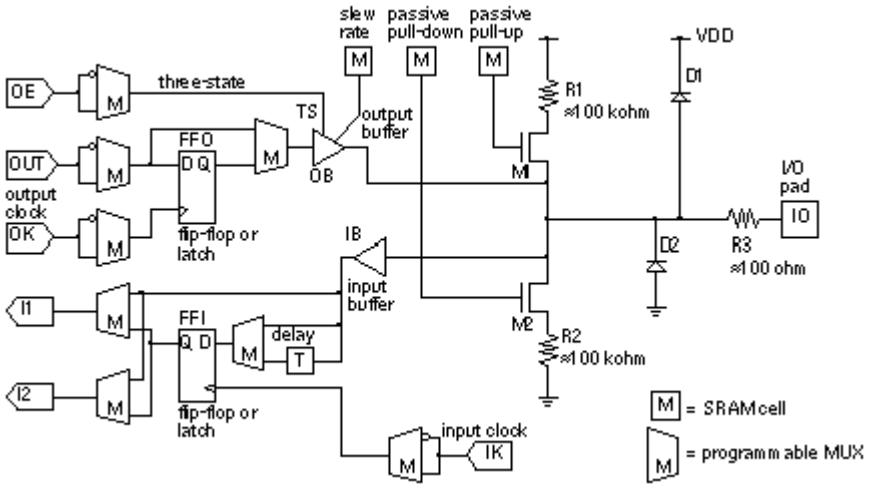



**Figura E-4** Esquema de un IOB de la serie 4000.

# F Fuentes

## F.1 Librerías en VHDL

Al compilar con el HCC para obtener un archivo de salida en VHDL, observamos que este hacia mencion a componentes que no estaban definidos y por lo tanto compiladores como el Fundation y el Max+Plus II no funcionaban. De esta forma observando el nombre que el HCC adjudicaba a los distintos componentes se realizaron las siguientes librerias.

Libor2: or de 2 entradas.

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
entity libor2 is
        port(   A,B     :in std_logic;
                Y       :out std_logic);
end libor2;
architecture comp of libor2 is
begin
                Y<=A or B;
end comp;
```

Libor3: or de 3 entradas.

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
entity libor3 is
        port(   A,B,C   :in std_logic;
                Y       :out std_logic);
end libor3;
architecture comp of libor3 is
begin
                Y<=A or B or C;
end comp;
```

Libor4: or de 4 entradas.

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
entity libor4 is
        port(   A,B,C,D :in std_logic;
                Y       :out std_logic);
end libor4;
architecture comp of libor4 is
begin
                Y<=A or B or C or D;
end comp;
```

Libxor: xor de 2 entradas.

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
entity LIBXOR is
        port (  Y : out std_logic;
                A,B : in std_logic);
end     LIBXOR;
architecture comp of LIBXOR is
begin
                Y<=A xor B;
end comp;
```

Liban2: and de 2 entradas.

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
entity LIBAN2 is
        port (  Y : out std_logic;
                A,B : in std_logic);
END LIBAN2;
architecture comp of LIBAN2 is
begin
                Y<=A and B;
end comp;
```

Liban3: and de 3 entradas.

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
entity LIBAN3 is
        port (  Y : out std_logic;
                A,B,C : in std_logic);
END LIBAN3;
architecture comp of LIBAN3 is
begin
                Y<=A and B and C;
end comp;
```

Libinv: not de 1 entrada.

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
entity LIBINV is
        port (  Y       : out std_logic;
                A       : in std_logic );
end LIBINV;
architecture comp of LIBINV is
begin
                Y<=not A;
end comp;
```

Libdffr: flip-flop D con Reset de 1 entrada.

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
entity LIBDFFR is
        port (  Q : out std_logic;
                CK,RESETZ,D : in std_logic);
end LIBDFFR;
architecture comp of LIBDFFR is
begin
        ff : process (CK)
        begin
                if rising_edge(CK) then
                        if RESETZ = '0' then
                                Q <= '0';
                        else
                                Q <= D;
                        end if;
                end if;
        end process ff;
end comp;
```

Libdff: flip-flop D de 1 entrada.

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
entity LIBDFF is
        port (  Q : out std_logic;
                CK,D : in std_logic );
end LIBDFF;
architecture comp of LIBDFF is
begin
        ff : process (CK)
        begin
                if rising_edge(CK) then
                                Q <= D;
                end if;
        end process ff;
end comp;
```

Liblog2: asignacion de valores.

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
entity liblog2 is
        port(   VDDLOG,GNDLOG : out std_logic);
end liblog2;
architecture comp of liblog2 is
begin
                VDDLOG<='1';
                GNDLOG<='0';
end comp;
```

## F.2 Fuentes del capítulo 4

### Contador de 0 a 99 en 7 segmentos con retardo

Versión para HCC

```
const spec tg= {
  fpga_type = "Xilinx4000",
  clock_pad = "P11"
};

void main(  target = tg,
     port (in) switches: 8,
     port (out) rightdigit:7,
     port (out) leftdigit:7){

  int x:4;
  int result, leftd, rightd:7;
  int n, count: 8;
  int y;

void seven_seg(){
  result = 0;
  if(x == 0x0) result = 0x01;
  else if(x == 0x1) result = 0x4f;
  else if(x == 0x2) result = 0x12;
  else if(x == 0x3) result = 0x06;
  else if(x == 0x4) result = 0x4c;
  else if(x == 0x5) result = 0x24;
  else if(x == 0x6) result = 0x20;
  else if(x == 0x7) result = 0x0f;
  else if(x == 0x8) result = 0x00;
  else if(x == 0x9) result = 0x0c;
  else result = 0x7e;
}

void retardo(){
  int cuenta1:8;
  int cuenta2, cuenta3:7;
  while(n != 0){
    n = n - 1;
    cuenta1 = 0;
    while (cuenta1 != 0xff){
      cuenta1 = cuenta1 + 1;
      cuenta2 = 0;
      while (cuenta2 != 0x7f){
        cuenta3 = 0;
        cuenta2 = cuenta2 + 1;
        while (cuenta3 != 0x7f)
          cuenta3 = cuenta3 + 1;
      };
    };
  };
}

void twodigit(){
  int tens:4;
  y = count;

  tens = 0;
  while(y >= 10) {
    tens = tens + 1;
    y = y - 10;
  };
  x = tens;
  seven_seg();
  leftd=result;
  x = y.(0..3);
  seven_seg();
  rightd=result;
}

  count = 0;
  par{
    while(1){
      par{
        leftdigit ! leftd;
        rightdigit ! rightd;
      }
    };
    while(1){
      twodigit();
      count = count + 1;
      if(count >= 100)
        count = 0;
      switches ? n;
      retardo();
    };
  }
}
```

Versión para TMCC

```
#pragma intbits 7
seven_seg(x)
#pragma intbits 4
  int x;
  {
#pragma intbits 7
  int result;

  result = 0;
  if(x == 0x0) result = 0x01;
  else if(x == 0x1) result = 0x4f;
  else if(x == 0x2) result = 0x12;
  else if(x == 0x3) result = 0x06;
  else if(x == 0x4) result = 0x4c;
  else if(x == 0x5) result = 0x24;
  else if(x == 0x6) result = 0x20;
  else if(x == 0x7) result = 0x0f;
  else if(x == 0x8) result = 0x00;
  else if(x == 0x9) result = 0x0c;
  else result = 0x7e;
  return(result);
  }


#pragma intbits 8
delay(n)
  int n;
  {
  int cuenta1;
#pragma intbits 7
  int cuenta2, cuenta3;
  while(n != 0){
    n = n - 1;
    cuenta1=0;
```

```
    while (cuenta1 != 0xff){
      cuenta1++;
      cuenta2=0;
      while (cuenta2 != 0x7f){
        cuenta2++;
        cuenta3=0;
        while (cuenta3 != 0x7f)
          cuenta3++;
      };
    };
  };
}

twodigit(y)
#pragma intbits 8
  int y;
  {
#pragma intbits 7
  int tens;
  int leftdigit, rightdigit;

  outputport(leftdigit, 37, 44, 40, 29, 35, 36, 38);
  outputport(rightdigit, 41, 51, 50, 45, 46, 47, 48);

  tens = 0;
  while(y >= 10) {
    tens = tens + 1;
    y = y - 10;
  };
  leftdigit = seven_seg(tens);
  rightdigit = seven_seg(y);
}

main()
  {
#pragma intbits 8
  int count;
  int switches;

  inputport(switches, 28, 27, 26, 25, 24, 23, 20, 19);

  count = 0;
  while(1) {
    twodigit(count);
    count = count + 1;
    if(count >= 100)
      count = 0;
    delay(switches);
    }
  }
```

## F.3 Fuentes del capítulo 5

### Contador de 0 a 99 en 7 segmentos con retardo

Versión 1 para HCC

```
const spec tgt={     // especificación del target
            fpga_type="Xilinx4000",
            clock_pad="P160"};

void main(target = tgt,  // declaración del target
            /* declaro puerto de entrada de 8 bits */
            port (in) switches:8,
            /* declaro puertos  de salida de 8 bits */
            port (out) leftdigit:8,
            port (out) rightdigit:8)
{
// se declaran variables especificando el ancho
int count, n, y, result, leftd, rightd :8;
int x, tens:4;

/* espera en x un n°  de 0 a f y devuelve el mismo n° en 7
seg. (8 bits) */
void seven_seg(){
            if(x == 0x0) result =~ 0xfc;
            if(x == 0x1) result =~ 0x60;
            if(x == 0x2) result =~ 0xda;
            if(x == 0x3) result =~ 0xf2;
            if(x == 0x4) result =~ 0x66;
            if(x == 0x5) result =~ 0xb6;
            if(x == 0x6) result =~ 0xbe;
            if(x == 0x7) result =~ 0xe0;
            if(x == 0x8) result =~ 0xfe;
            if(x == 0x9) result =~ 0xf6;
            if(x == 0xa) result =~ 0xee;
            if(x == 0xb) result =~ 0x3e;
            if(x == 0xc) result =~ 0x9c;
            if(x == 0xd) result =~ 0x7a;
            if(x == 0xe) result =~ 0x9e;
            if(x == 0xf) result =~ 0x8e;

}

void retardo(){ // espera n periodos de reloj
            while(n != 0)   n = n - 1;
}
```

```
//toma de "count" el numero a mostrar
void twodigit(){
      tens, y = 0, count; // asignación paralela
      while(y >= 10)
            { /* este loop guarda en  tens el digito
             alto  y en "y" el digito bajo */
             tens, y = tens+1, y-10; //asig. paralela
            };
      x = tens;  /* asigno a x el digito alto antes de llamar a
                  seven_seg */
            seven_seg();
            x, leftd = y.(0..3), result;  /* asig.  paralela, y.(0..3)
            asigna  los bits bajos de y a  x antes de llamar a
            seven_seg,  a leftd se le  asigna result (lo que
            devuelve seven_seg()) */
            seven_seg();
            rightd= result; /* trabajo con  rightd y leftd donde
            antes trabajaba con los puertos lefdigit y rightdigit
            */
}

      // Comienza el programa principal
      count = 0;
      par{
            while(1){   /* este loop lo agregamos para
                        actualizar las salidas */
                  par{ /* escritura en puerto leftdigit */
                     leftdigit ! leftd;
                     /* escritura  en puerto rightdigit */
                     rightdigit ! rightd;
                      }
                  };
             while(1) {
                        twodigit();
                        par{
                           count = count + 1;
                           // lectura en n  del puerto switches
                           switches ? n;
                            }
                        if(count >= 100)  count = 0;
                        retardo();
                        };
            }
}
```

Versión 1 para TMCC

```
#pragma intbits 8  /* fija ancho de valor de retorno */
seven_seg(x)
#pragma intbits 4  // fija ancho de x
      int x; // declaro el parametro de la función
      {
#pragma intbits 8
/* fija el ancho de las variables declaradas a partir de aquí
(result, n, y, leftdigit, rightdigit) */
    int result;

    x = x & 0xf; result = 0;
    if(x == 0x0) result = 0xfc;
    if(x == 0x1) result = 0x60;
    if(x == 0x2) result = 0xda;
    if(x == 0x3) result = 0xf2;
    if(x == 0x4) result = 0x66;
    if(x == 0x5) result = 0xb6;
    if(x == 0x6) result = 0xbe;
    if(x == 0x7) result = 0xe0;
    if(x == 0x8) result = 0xfe;
    if(x == 0x9) result = 0xf6;
    if(x == 0xa) result = 0xee;
    if(x == 0xb) result = 0x3e;
    if(x == 0xc) result = 0x9c;
    if(x == 0xd) result = 0x7a;
    if(x == 0xe) result = 0x9e;
    if(x == 0xf) result = 0x8e;
    return(~result);
}

delay(n)
    int n;
```

```
{    while(n != 0)   n = n - 1;  }

twodigit(y)
    int y;
{
    int tens;
    int leftdigit, rightdigit;
            /* declaro puertos de salida */
            outputport(leftdigit);
            outputport(rightdigit);

            tens = 0;
            while(y >= 10) {
                    tens = tens + 1;
                    y = y - 10;
                    }
            leftdigit = seven_seg(tens);
            rightdigit = seven_seg(y);
}

// comienza programa principal
main(){
#pragma intbits 8
        int count, switches;
        // decl. switches como puerto de entrada
        inputport(switches);

            count = 0;
            while(1) {
                    twodigit(count);
                    count = count + 1;
                    if(count >= 100)
                            count = 0;
                    delay(switches);
                    }
            }
```

## División

### Versión para HCC

```
/* Integer division by long-division method */
const spec tgt={ // especificación del target
      fpga_type="Xilinx4000", // para que  se  genere XNF
      clock_pad="P160"};  /* es necesario para que no use
                             osc. interno */

const dw = 16;

void main(  target=tgt,
      port (in)  STDIN  : dw, /* declaro puerto de entrada de
                              ancho dw (16) */
      port (out) STDOUT : dw) // declaro  puerto de salida
{
      int a, b, c : dw;  // declaración de variables
      int bits : 5;

while (1) {
       STDIN ? a;  // cargo en a el valor de  stdin
      STDIN ? b;  // en el siguiente ciclo se carga en b
      c, bits = 0, 1;  // asignación paralela
      while (b .<=. a)  /* desplazo b hasta  que sea mayor
      que a. Guardo en bits el nº de rotaciones
      Problema: a debe ser menor que 0x8000 si no, no
      sale del loop */
            b, bits = b << 1, bits + 1;
      do par {
          if (a .>=. b)
                a, c = a - b, (c << 1) ^ 1;  // ^ es = xor
          else
                c = c << 1;
          b, bits = b >> 1, bits - 1;
      } while (bits != 0);
      STDOUT ! c; // se escribe c en el  puerto
   };
}
```

### Versión para TMCC

```
// fijo ancho de las próximas variables
#pragma intbits 16
main(){
            int c, stdin, stdout;
#pragma intbits 17
            int a;
#pragma intbits 18
            int b;
#pragma intbits 5
            int bits;
            // declaro puertos de I/O
            inputport(stdin); outputport(stdout);

            while (1) {
                        // cargo en a la entrada
                        a = stdin & 0x0ffff;
                        while( 0 ){};
                        // en el siguiente ciclo cargo en b
                        b = stdin&0x0ffff;
                        c = 0;
                        bits =  1;
/* desplazo b hasta que sea mayor que  a,  guardo en bits
el nº de  desplazamientos */
                        while( b <= a ){
                                b = b<<1;
                                bits = bits+1;
                        };
                        while( bits!=0 ){
                                if (a >= b){
                                        a = a-b;
                                        c = (c<<1)^1;
                                }
                                else
                                        c = c<<1;
                                b = b>>1;
                                bits = bits-1;
                        };
                        // escribo en el puerto
                        stdout = c;
            };
}
```

## Raíz cuadrada ( algoritmo 1)

Versión para HCC

```
/* especificación del target */
const spec tgt={
        fpga_type="Xilinx4000",
        clock_pad="Pº160"};

/* declaración de constantes */
const half_wd = 4 : 3;
const wd = half_wd << 1;
const sq = (1 << (wd - 2)) : wd;

void main( target=tgt,  // declaro target
        port (in)  STDIN: wd, // declaro puertos
        port (out) STDOUT : wd)
{
/* declaraciones de variables, el compilador infiere los anchos, por eso no se declaran */
    int a, p, q, r;
    int i;
    while (1) {
      par { STDIN ? a;  // cargo puerto de entrada en a
            p, q, r = 0, 0, sq;  /* asignación paralela, se
                                    realizan las 3 en el mismo ciclo */
            i  = half_wd; };

      do {
        if (p+q+r .<=. a)  /* comparación sin signo  */
            i, r, q, p=i-1, r>>2, (q>>1)+r, p+q+r;
        else
            i, r, q    = i - 1, r >> 2, (q >> 1);
      } while (i != 0);
      STDOUT ! q;  // doy como salida q
    }
}
```

Versión para TMCC

```
#pragma intbits 8  /* fijo anchos de stdin y  stdout*/
main(){
        int stdin,stdout;
#pragma intbits 9  /* fijo anchos de a,p,q y r */
        int a, p, q, r;
#pragma intbits 4  /* fijo ancho de i */
        int i;
        /* declaro puertos */
        inputport (stdin); outputport(stdout);

        while (1) {
                a = stdin & 0x0ff;  /* en los 8 bits bajos
                        de a cargo stdin y en el alto 0 */
                p = 0;
                q = 0;
                r = 0x80;
                i = 4;
                while(i!=0){
                        if (p+q+r <= a){
                                i = i-1;
                                p = p+q+r;
                                q = (q>>1)+r;
                                r = r>>2;
                        }
                        else{
                                i = i-1;
                                r = r>>2;
                                q= q>>1;
                        };
                };
                /* escribo en el puerto de salida */
                stdout=q;
        }
}
```

## Raiz cuadrada (algoritmo 2)

Versión para HCC

```
const spec tgt={ /* especificacion del target */
    fpga_type="Xilinx4000",
    clock_pad="P160"};

const cero=0:16; /* declaro un cero de 16 bits de ancho */

void main( target=tgt, // declaracion del target
    port (out) sq:8, // declaracion de puertos
    port (in)  va:8)
{
    /* declaracion de variables, con su respectivo  ancho */
    int N,tt,Rt,nn:8;
    int Bit1:4;
    int N_reg,R_reg,tmp:24;

    while(1){
        va ? N; /* cargo puerto de entrada en N*/
        Bit1,N_reg,R_reg,Rt=8,0 @ N,0,0;
        /* @ indica concatenación */
while (Bit1!=0){
        R_reg=((cero @ Rt)<<16)|0x4000;
       /* si no le especificaramos el ancho a la constante
        cero no se puede inferir su ancho */
        tmp, Bit1 =(N_reg - R_reg), Bit1-1;
        if (tmp>=0){
            par{ /* asignación paralela */
                N_reg=tmp<<2;
                Rt=(Rt<<1)|1;
            }
        }
        else {
            par{
                N_reg=N_reg<<2;
                Rt=(Rt<<1);
            }
        };
      };
      sq ! Rt; /* escritura en puerto */
   };
}
```

Versión para TMCC

```
main(){
#pragma intbits 4 /* fijo ancho de bit1*/
            int Bit1;
#pragma intbits 24 /* fijo ancho de N_reg,  R_reg, tmp */
            int N_reg, R_reg, tmp;
#pragma intbits 8 /* fijo ancho de Rt, stdin, stdout */
            int Rt, stdin, stdout;
#pragma intbits 9 /* fijo ancho de N */
            int N;
            /*declaro puertos de I/O */
            inputport(stdin); outputport(stdout);

            while(1){
              // cargo stdin en 8 bits bajos de N y 0 en el alto
                  N=stdin & 0x0ff;
                  N_reg=N;
                  R_reg=0;
                  Rt=0;
                  Bit1=8;
                  while (Bit1--){
                        R_reg=(Rt<<16)|0x4000;
                        while(0){}; /* while puesto para
                              generar un nuevo estado */
                        tmp=(N_reg - R_reg);
                        while(0){}; /* Idem anterior */
                        if (tmp>=0){
                              N_reg=tmp<<2;
                              Rt=(Rt<<1)|1; // | es = OR
                        }
                        else {
                              N_reg=N_reg<<2;
                              Rt=(Rt<<1);
                        };
                  };
                  stdout=Rt; /* escribo resultado en el puerto */
            };
}
```

## Raíz cuadrada (algoritmo 3)

Versión para HCC

```
const spec tgt={  /* especificacion del target */
    fpga_type="Xilinx4000", /* genera XNF */
    clock_pad="P160"}; /* necesario */

const sw=8;

void main(      target=tgt, /* declaracion target */
          port (out) sq:sw, /* declaro puertos */
          port (in)  va:sw)
{
    /* declaración de variables */
    int tt, N, nn: sw;
    int l2, tmp, u, v, u2, m:sw;

    while(1){
        va ? N; // cargo puerto de entrada en N
        if (2 .>. N){  //comparación sin signo de 2 con N
            u=N;
        }
        else {
            tmp, l2 = N>>2, 0; /*asignación  paralela, se
                                realiza en un Tclk */
            while (tmp!=0){
                tmp, l2 =(tmp>>2), l2+1;
            };

          /*implementamos u=1<<l2, u2=u<<l2 teniendo en
          cuenta que el máx. valor de l2 en este punto es 3*/
            case(l2){
                0 : { u, u2 = 1, 1;  }
                1 : { u, u2 = 2, 4;  }
                2 : { u, u2 = 4, 16; }
                3 : { u, u2 = 8, 64; }
                default : { u, u2 = 0, 0; }
            };
            v=u;
            while (l2!=0) {
                  v, l2 = v>>1, l2-1;
                  case(l2){   /*implementa m=(u+u+v)<<l2, el
                        máx. valor de l2 en este pto. es 2 */
                      0 : {m = (u + u + v);  }
                      1 : {m = (u + u + v) << 1;  }
                      2 : {m = (u + u + v) << 2;  }
                      default : {m = 0;  }
                  };
                  m = m+u2; /* hcc no acepta m+=u2 */
                  if (m .<=. N) { /*comparacion sin
                                    signo de m y N */
                      u, u2 =u+v, m;
                  };
            };
        };
        sq ! u;  /* escritura en puerto */
    };
}
```

## Versión para TMCC

```
#pragma intbits 8 /* fijo ancho de l2, stdin, stdout*/
main(){
    int l2,  stdin, stdout;
#pragma intbits 9 /* fijo ancho de N, nn,.. ,v */
    int N, nn, tt,u, u2,sal,v;
/* declaro puertos */
    inputport(stdin); outputport(stdout);
    while(1){
        N=stdin&0x0ff; /* cargo stdin en 8 bits bajos de N y 0
en el alto */
        if (N > 1){
            tt = N>>2; l2 = 0;
            while (tt){ tt>>=2; l2++;};
            u = 1 << l2;
            v = u;
            u2 = u << l2;
            while (l2) {
                l2--;
                v >>= 1;
                while(0){}; /* while puesto para generar un
nuevo estado y simplificar lógica */
                nn = (u + u + v) << l2;
                while(0){}; /* idem anterior */
                nn += u2;
                while(0){}; /* idem anterior */
                if (nn <= N) {
                    u += v; /* u = u + v */
                    u2 = nn;
                };
            };
            sal=u;
        }
        else { sal=N;};

        stdout=sal; /* escribo en el puerto */
    };
}
```

# Árbitro de bus

## Versión 1 para TMCC

```
main(){
#pragma intbits 1 //fijo ancho de variables
    int arts,brts,acts,bcts;
    int est_ant_A,est_ant_B;
    int a,b,c,d;
#pragma intbits 2
    int estado,reserva;
    /* declaro puertos */
    inputport(arts); inputport(brts);
    outputport(acts); outputport(bcts);
    /* declaro entradas como registradas */
    portflags(arts,0x3); portflags(brts,0x3);

    while(1){
        a=arts;b=brts; // leo puertos entrada
        c=a & (~est_ant_A); /* formo c y d  con el valor
                               anterior de est_ant_A*/
        d=b & (~est_ant_B);
        est_ant_A= a; // actualizo est_ant_A
        est_ant_B= b;
        if (estado==1){ /* canal ocupado por  A */
            if (c & (~d)){ // A devuelve el bus
                estado=reserva; /* próximo  estado */
                if (reserva==2) bcts=1;
                reserva=0;
                acts=0;
            }
            else if (d & (~c)){reserva=2;} /* B  reserva */
            else if (d & c){ // devolucion y pedido simult.
                estado=2;
                acts=0;
                bcts=1;
            }
            else {acts=1;bcts=0;};
        }
        else if (estado==2){    /* bus ocupado por B */
            if (d & (~c)){ // B devuelve el bus
                estado=reserva;
                if (reserva==1) acts=1;
                reserva=0;
                bcts=0;
            }
            else if (c & (~d)) {reserva=1;} /* A  reserva */
            else if (d & c) { /* devolucion y  pedido
                                 simultaneos */
                estado=1;
                acts=1;
                bcts=0;}
            else {acts=0;bcts=1;};
        }
        else if (estado==0){    // canal libre
            if (c){ /* A tiene prioridad */
                estado=1;acts=1;
                if(d) reserva=2;
            }
            else if (d & (~c)){/* si A esta libre puede
                                  reservar B */
                estado=2;bcts=1;
            }
            else {
                acts=0;bcts=0;
            };
        };
    };
}
```

## Versión 1 para HCC

```
const spec harp1 = { /* especificacion de target */
    fpga_type = "Xilinx4000",
    clock_pad = "P160"};

const LIBRE  = 0;
const OCUPAA = 1;
const OCUPAB = 2;
#define a (temp_a & (~est_ant_A))
#define b (temp_b & (~est_ant_B))

main(   target=harp1,  /* decl. de target */
    port(in) arts: 1, /* decl. de puertos */
    port (in) brts:1,  port (out) acts: 1,
    port (out) bcts: 1)
{
    /* declaracion de variables*/
    int estado,reserva:2;
    int temp_a,temp_b:1;
    int est_ant_A ,est_ant_B:1;

    while(1){
        par{
            arts ? temp_a; /* leo puertos I */
            brts ? temp_b;
            est_ant_A=temp_a;
            est_ant_B=temp_b;
            case (estado){
                OCUPAA: par{ /* bus ocupado por A, el
                 par{} hace que las dos escrituras a puertos
                y el if se ejecuten en el mismo ciclo */
                        acts ! 1;
                        bcts ! 0;
                        if (a & (~b)){ /* A devuelve  el bus */
                            estado,reserva = reserva,LIBRE;
                        }
                        else if (b & (~a)){ // B  reserva el bus
                            reserva=OCUPAB;
                        }
                        else if (a & b) { /* devolucion y pedido
                                             simultaneos */
                            estado=OCUPAB;
                        }
                        else    delay;
                    }
                    OCUPAB: par{ /* bus ocupado por B */
                        bcts ! 1; /* esc. a puertos */
                        acts ! 0;
                        if (b & (~a)){ /* B devuelve el bus */
                            estado,reserva = reserva,LIBRE;
                        }
                        else if (a & (~b)){ /* A reserva el bus */
                            reserva=OCUPAA;
                        }
                        else if (b & a){ /* devolución y pedido
                                             simultaneo */
                            estado=OCUPAA;
                        }
                        else delay;
                    }
                    default: if (a){ /* A tiene prioridad */
                            par{
                                estado=OCUPAA;
                                reserva=(b ? OCUPAB : 0);
                           /* asignación condicional, si b=1 =>
                            reserva = OCUPAB sino reserva = 0 */
                            }
                        }
                        else if (b){ /* si A no reserva lo puede
                                        hacer B*/
                            estado=OCUPAB;
                        }
                        else delay;
                };
            }
        };
}
```

## Versión 1 en VHDL

```
-- maquina de estados generica
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
entity arbit_v1 is port (
      hclk, reset : in std_logic ;
      arts,brts : in std_logic ;
      acts,bcts : out std_logic );
end arbit_v1 ;

architecture ar1 of arbit_v1 is
      signal a,b,tmpA,tmpB,est_ant_A,est_ant_B : std_logic;
      type tipo_estado is (LIBRE,OCUPAA,OCUPAB) ;
      signal estado, reserva : tipo_estado ;
begin

      tmpA<=(a and not est_ant_A);
      tmpB<=(b and not est_ant_B);

      process (hclk, reset)
      begin
            if reset='1' then
                  a<='0';
                  b<='0';
                  est_ant_A<='0';
                  est_ant_B<='0';
                  estado<=LIBRE;
                   acts<='0';
                   bcts<='0';
                   reserva<=LIBRE;
      elsif rising_edge(hclk) then
            a<=arts;
            b<=brts;
            est_ant_A<=a;
            est_ant_B<=b;
            case estado is
                  when OCUPAA=>
                        if (tmpA and not tmpB)='1' then
                              estado<=reserva;
                              eserva<=LIBRE;
                              acts<='0';
                        if (reserva=OCUPAB) then
                              bcts<='1'; end if;
                        elsif (tmpB and not tmpA)='1' then
                              reserva<=OCUPAB;
                        elsif (tmpA and tmpB)='1' then
                              estado<=OCUPAB;
                              acts<='0';bcts<='1';
                        else
                              acts<='1';bcts<='0';
                        end if;
                  when OCUPAB=>
                        if (tmpA and not tmpB)='1' then
                              reserva<=OCUPAA;
                         elsif (tmpB and not tmpA)='1' then
                              estado<=reserva;
                              reserva<=LIBRE;
                              bcts<='0';
                              if (reserva=OCUPAA) then
                                    acts<='1';
```

```
                end if;
            elsif (tmpA and tmpB)='1' then
                estado<=OCUPAA;
                acts<='1';  bcts<='0';
            else
                acts<='0';  bcts<='1';
            end if;
        when LIBRE=>
            if (tmpA='1') then
                estado<=OCUPAA;
                acts<='1';
                if (tmpB='1')then
                    reserva<=OCUPAB;
                end if;
            elsif (tmpB='1') then
                estado<=OCUPAB;
                bcts<='1';
            end if;
        end case;
    end if;
  end process;
end ar1;
```

## Versión 2 en VHDL

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;

entity árbitro is port (
    clk ,reset: in std_logic ;
    arts,brts : in std_logic ;
    acts,bcts : out std_logic );
end árbitro ;

architecture ar1 of árbitro is
    type tipo_estado is (s0,s1,s2,s3) ;
    type tipo_estado3 is (s0,s1,s2) ;
    signal estado_act1, estado_prox1: tipo_estado ;
    signal estado_act2, estado_prox2 : tipo_estado ;
    signal estado_act3, estado_prox3  : tipo_estado3 ;
    signal pa,pb: std_logic;
    signal acts_d,bcts_d : std_logic ;
begin

registros : process (clk,reset)
    begin
       if reset='1' then
          estado_act1 <= s0;
          estado_act2 <= s0;
          estado_act3 <= s0;
          acts <= '0';
          bcts <= '0';
       elsif clk'event and clk = '1' then
          estado_act1 <= estado_prox1;
          estado_act2 <= estado_prox2;
          estado_act3 <= estado_prox3;
          acts <= acts_d;
          bcts <= bcts_d;
       end if ;
    end process ;

maqA : process (estado_act1,arts)
    begin
       case estado_act1 is
          when s0 =>
             if arts='1' then
                pa <= '1';
                estado_prox1 <= s1 ;
             else
                pa <= '0';
                estado_prox1 <= s0 ;
             end if;
          when s1 =>
             pa <= '1';
             if arts='1' then
                estado_prox1 <= s1 ;
             else
                estado_prox1 <= s2 ;
             end if;
          when s2 =>
             if arts='1' then
                estado_prox1 <= s3 ;
                pa <= '0';
             else
                estado_prox1 <= s2 ;
                pa <= '1';
             end if;
          when s3 =>
             pa <= '0';
             if arts='1' then
                estado_prox1 <= s3 ;
             else
                estado_prox1 <= s0 ;
             end if;
       end case ;
    end process ;

maqB : process (estado_act2,brts)
    begin
       case estado_act2 is
          when s0 =>
             if brts='1' then
                pb <= '1';
                estado_prox2 <= s1 ;
             else
                pb <= '0';
                estado_prox2 <= s0 ;
             end if;
          when s1 =>
             pb <= '1';
             if brts='1' then
                estado_prox2 <= s1 ;
             else
                estado_prox2 <= s2 ;
             end if;
          when s2 =>
             if brts='1' then
                estado_prox2 <= s3 ;
                pb <= '0';
             else
                estado_prox2 <= s2 ;
                pb <= '1';
             end if;
          when s3 =>
             pb <= '0';
             if brts='1' then
                estado_prox2 <= s3 ;
             else
                estado_prox2 <= s0 ;
             end if;
       end case ;
 end process ;

maqArb : process (estado_act3,pa,pb)
    begin
       case estado_act3 is
          when s0 =>
             if pa='1'  then
```

```
                estado_prox3 <= s1 ;
                acts_d <= '1';
                bcts_d <='0';
            elsif pb='1'  then
                estado_prox3 <= s2 ;
                acts_d <= '0';
                bcts_d <='1';
            else
                estado_prox3 <= s0 ;
                acts_d <= '0';
                bcts_d <='0';
            end if;
          when s1 =>
            acts_d <= '1';
            bcts_d <='0';
            if pa='1' then
                estado_prox3 <= s1 ;
            else
                estado_prox3 <= s0 ;
            end if;
          when s2 =>
            bcts_d <= '1';
            acts_d <='0';
            if pb='1' then
                estado_prox3 <= s2 ;
            else
                estado_prox3 <= s0 ;
            end if;
        end case ;
    end process ;
end ar1 ;
```

## Versión 2 para TMCC

```
main(){
/* declaracion de variables, aquí no es necesario especificar que son de ancho uno */
int arts, brts, acts, bcts, pa, pb;
#pragma intbits 2 /*ahora fijo ancho 2 */
int est1, est2, est3;

/* declaracion de puertos */
inputport(arts); inputport(brts);
outputport(acts); outputport(bcts);
/* declaro salidas como no registradas */
portflags(acts, 0x1); portflags(bcts, 0x1);

    while(1){
        if (est3 == 0){ // máquina 3
            /* el and con 1 es para forzar el segundo
               bit a cero */
            est3 = (pb & ~pa) << 1 | (pa & 1); /* si pb=1
                        y pa=0 => est3=2; si pa=1
                        =>  est3=1; sino est3=0 */
            acts = pa; /* esc. en puertos O*/
            bcts = pb & ~pa;
        }
        else if (est3 == 1){
            if (pa == 0) est3 = pb << 1; /* est3 va a 2 si
                                    pb=1 y a 0 si pb=0 */
            acts = pa; /* esc. en puertos O*/
            bcts = pb & ~pa; /* ~ representa
                                complemento a 1*/
        }
        else if (est3 == 2){
            if (pb == 0) est3 = pa & 1;
            acts = pa & ~pb; /* esc. en O*/
            bcts = pb;
        }

        /* máquina 1 - de Moore las salidas  dependen
           solo del estado, aunque no lo  parezca */
        if (est1 == 0){/* si en un flanco de  reloj arts=1
                  entonces pa pasa a valer 1 al  mismo
                  tiempo que est1 pasa a valer 1*/
            pa = arts;
            est1 = arts & 1; // fuerzo bit 2 a 0
        }
        else if (est1 == 1){/* me quedo en est1 dando
                        salida 1 hasta que baja arts */
            pa = 1;
            if (~arts)  est1 = 2;
            else  est1 = 1;
        }
        else if (est1 == 2){/* me quedo en est2 dando
                        salida 1 hasta que sube arts */
            pa = 1;
            if (arts) est1 = 3;
        }
        else { /* me quedo en est3 dando  salida 0
                  hasta que baje arts */
            pa = 0;
            if (arts)       est1 = 3;
            else            est1 = 0;
        };
        // máquina 2 analoga a la anterior
        if (est2 == 0){
            pb = brts;
            est2 = brts&1;
        }
        else if (est2 == 1){
            pb = 1;
            if (~brts)      est2 = 2;
        }
        else if (est2 == 2){
            pb = 1;
            if (brts)       est2 = 3;
        }
        else {
            pb = 0;
            if (~brts)      est2 = 0;
        };
    }
}
```

## Versión 2 para HCC

```
const spec tg = { /* espec. del target */
    fpga_type = "Xilinx4000",
    clock_pad = "P160"};
/* defino constantes para los estados */
const s0 = 0;
const s1 = 1;
const s2 = 2;
const s3 = 3;

void main(      target = tg, // decl. del target
            port (in) arts:1, // decl. de puertos
            port (in) brts:1,
            port (out) acts:1,
            port (out) bcts:1){
    /* declaración de variables globales*/
    int pa, pb:1;
```

```
    void maq1(){ // maq. 1, maneja arts y da salida pa
        int est1:2; /* variable local */
        case (est1){ /* en cada rama calculo nuevos
                        valores de est1 y pa en paralelo  */
          0 :  est1, pa = (arts ? s1 : s0), arts;
             /* _?_:_ asignacion condicional, si arts=1
              => est1=s1 sino est1=s0 */
          1 :  est1, pa = (arts ? s1 : s2), 1;
          2 :  est1, pa = (arts ? s3 : s2), ~arts;
          3 :  est1, pa = (arts ? s3 : s0), 0;
        };
    }

    void maq2(){// maq. 2, maneja brts y da salida pb
        int est2:2;
        case (est2){
          0:  est2, pb = (brts ? s1 : s0), brts;
          1:  est2, pb = (brts ? s1 : s2), 1;
          2:  est2, pb = (brts ? s3 : s2), ~brts;
          3:  est2, pb = (brts ? s3 : s0), 0;
        };
    }

void maq3(){ /* maquina 3, da salidas  acts y bcts
                    en funcion de pa y pb */
        int est3:2;
        case (est3){
            0:  par{
                    est3 = (pa ? s1 : (pb ? s2 : s0));
                    /* asignación condicional andidada, si
                      pa=1  => est3=s1, si pa=0 y pb=1
                      => est3=s2  sino est3=s0 */
                    acts ! pa;
                    bcts ! (~pa & pb);
                }
            1:  par{
                    est3 = (pa ? s1 : (~pa & pb ? s2 : s0));
                    acts ! pa;
                    bcts ! (~pa & pb);
                }
            2:  par{
                    est3 = (pb ? s2 : (~pb & pa ? s1 : s0));
                    acts ! (pa & ~pb);
                    bcts ! pb;
                }
            3: delay;
        };
    }

    // Programa principal
    while(1){
        par{ // ejecuto en paralelo las 3 máq de estados
            maq1();
            maq2();
            maq3();
        }
    };
}
```

## Ejemplo de RAM externa y multiplicación

Versión para HCC

```
const spec yyy = {  //especificación del target
    fpga_type= "Xilinx4000",
    clock_pad= "P13"};
// especificacion de la ram externa
const spec tipo_ram={
    addr = {"mem_Addr<0>", "mem_Addr<1>",
        "mem_Addr<2>","mem_Addr<3>","mem_Addr<4>",
        "mem_Addr<5>","mem_Addr<6>","mem_Addr<7>"},
    data = {"mem_Data<0>","mem_Data<1>",
        "mem_Data<2>","mem_Data<3>","mem_Data<4>",
        "mem_Data<5>","mem_Data<6>","mem_Data<7>"},
    ce   = "mem_ce",
    wb  = "mem_wb",
    en   = "mem_en"
};

main(   target = yyy, //declaracion de target
    port (in) comenzar:1, // decl. de puertos
    port (out) fin:1,
    eram mem[256] = tipo_ram:8) //decl. de la ram externa
{
    //declaración de variables
    int aux:1;
    int a,b,i,j:8;
int c:16;

    while(1){
        comenzar ? aux; //leo entrada
        // espero a que comenzar sea igual a 1
        while (aux==0){comenzar ? aux;};
        do {     //leo 256 datos y escribo 128 de a 2 lugares
            par{
                a=mem[i]; /*leo posicion i de la ram */
                j=i+1;
            }
            b = mem[ j ];   //leo sig. posicion
            c = a .*. b;    //multiplico datos
            mem[ i ] = c.(0..7);    /*escribo byte bajo en
                                      posicion baja */
            par{
                mem[j]=c.(8..15); /*escribo byte alto en
                                    posicion alta */
                i=i+2;
            }
        } while (i != 0);
        fin ! 1; //aviso que termine
    };
}
```

Versión 1 para TMCC

```
main(){
    // declaro variables de ancho 1
    int mem_ce, mem_wb, mem_en;
    int comenzar, fin;
#pragma intbits 8
    int a, b, i, j;
    int mem_addr, mem_data;
#pragma intbits 16
    int c;

    //declaro puertos entrada, salida y buses
    inputport(comenzar);
    bus_port(mem_data);
    outputport(fin);outputport(mem_ce);
    outputport(mem_en); outputport(mem_wb);
    outputport(mem_addr);

    mem_en=1;mem_ce=1;mem_wb=1;
    while(1){
```

```
        fin=0; i=0; j=0;
        while (comenzar==0){}; // espero pulso de comienzo
        mem_en=0; mem_ce=0; /*activo señales para
                                primer dato */
        mem_addr=i;
        while ((i!=0) | (j!=1)){
            mem_en=1; //desactivo señales de ram
            mem_wb=1; mem_ce=1;
            a=mem_data; //leo dato
            j =i+1;
            while(0){};          //espero a siguiente
             mem_en=0; /* activo señales de lectura */
            mem_addr=j; mem_ce=0;
            while(0){};
            b=mem_data; //leo dato
            mem_en=1; /* desactivo señales de lectura */
            mem_ce=1;
            /* multiplico a y b se genera un nuevo estado */
            c = multip8(a,b);
            mem_data=c;  //escribo c
            mem_ce=0; /* activo señales de escritura */
            mem_addr=i; mem_wb=0;
            while(0){};
            mem_ce=1; mem_wb=1; /* finalizo escritura */
            while(0){};
            mem_ce=0; /* activo señales de escritura */
            mem_addr=j; mem_wb=0;
            mem_data=(c>>8); //escribo byte alto
            while(0){};
            i =i+2;
            mem_ce=1; mem_wb=1; /* finalizo escritura */
            while(0){};
            /* si no llegue al final genero señales para una
               nueva lectura */
            if (i!=0){mem_ce=0; mem_en=0; mem_addr=i;};
            bus_idle(mem_data); /* apronto bus para leer */
        };
        fin=1; // aviso que termine
    };
}

#pragma intbits 16  /* fijo ancho de bits que devuelve la
                       funcion */
int
multip8(a,b)
#pragma intbits 8 // fijo ancho de los parametros
    int a,b;
{
#pragma intbits 16
    //declaracion de variables locales
    int tmp0,tmp1,tmp2,tmp3;
    int tmp4,tmp5,tmp6,tmp7;
    int aux,sum0,sum1,sum2,sum3,mul0,mul1;
    int sal1,sal2,sal3, c;

    aux=a & 0x00ff; /*en aux cargo 8 ceros y a en los bits
                      bajos */
    //desplazamientos
    if ((b & 1  )==1  ){tmp0= aux    ;}
    else {tmp0=0;};
    if ((b & 2  )==2  ){tmp1=(aux<<1);}
    else {tmp1=0;};
    if ((b & 4  )==4  ){tmp2=(aux<<2);}
    else {tmp2=0;};
    if ((b & 8  )==8  ){tmp3=(aux<<3);}
    else {tmp3=0;};
    if ((b & 16 )==16 ){tmp4=(aux<<4);}
    else {tmp4=0;};
    if ((b & 32 )==32 ){tmp5=(aux<<5);}
    else {tmp5=0;};
    if ((b & 64 )==64 ){tmp6=(aux<<6);}
    else {tmp6=0;};
    if ((b & 128)==128){tmp7=(aux<<7);}
    else {tmp7=0;};

    //sumas
    sum0=tmp0+tmp1;
    sum1=tmp2+tmp3;
    sum2=tmp4+tmp5;
    sum3=tmp6+tmp7;
    //sumas 2 nivel
    mul0=sum0+sum1;
    mul1=sum2+sum3;
    //ultimo nivel de suma
    c=mul0+mul1;
    return(c);
}
```

## Versión 2 para TMCC

```
main(){
   int mem_ce, mem_wb, mem_en, comenzar, fin;
#pragma intbits 8
    int a, b, i, j, mul_m1_mn, mul_m1_mr;
    int mem_addr, mem_data;
#pragma intbits 16
    int c, z, mul_m1_res;

    inputport(comenzar); inputport(mul_m1_res);
    bus_port(mem_data);
    outputport(fin);outputport(mem_ce);
    outputport(mem_en);    outputport(mem_wb);
    outputport(mem_addr);    outputport(mul_m1_mn);
    outputport(mul_m1_mr);   portflags(mul_m1_mn, 0x1);
    portflags(mul_m1_mr, 0x1);

    mem_en=1;mem_ce=1;mem_wb=1;
    while(1){
        fin=0; i=0; j=0;
        while (comenzar==0){}; // espero pulso de comienzo
        mem_en=0; mem_ce=0; /*activo señales para
                                primer dato */
        mem_addr=i;
        while ((i!=0) | (j!=1)){
            mem_en=1; //desactivo señales de ram
            mem_wb=1; mem_ce=1;
            a=mem_data; //leo dato
            j =i+1;
            while(0){};          //espero a siguiente
             mem_en=0; /* activo señales de lectura */
            mem_addr=j; mem_ce=0;
            while(0){};
            b=mem_data; //leo dato
            mem_en=1; /* desactivo señales de lectura */
            mem_ce=1;
            /* multiplico a y b se genera un nuevo estado */
             mul_m1_mn=a;
             mul_m1_mr=b;
             z=mul_m1_res;
             while(0){};
             c=z;
            mem_data=c;  //escribo c
            mem_ce=0; /* activo señales de escritura */
            mem_addr=i; mem_wb=0;
            while(0){};
            mem_ce=1; mem_wb=1; /* finalizo escritura */
            while(0){};
            mem_ce=0; /* activo señales de escritura */
            mem_addr=j; mem_wb=0;
            mem_data=(c>>8); //escribo byte alto
            while(0){};
            i =i+2;
```

```
        mem_ce=1; mem_wb=1; /* finalizo escritura */
        while(0){};
        /* si no llegue al final genero señales para una
           nueva lectura */
        if (i!=0){mem_ce=0; mem_en=0; mem_addr=i;};
        bus_idle(mem_data); /* apronto bus  para leer */
        };
        fin=1; // aviso que termine
    };
}
```

## Versión en VHDL

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity memMult is
  port(
     clk, comenzar,  reset:in std_logic;
     memAddr: out std_logic_vector(7 downto 0);
     memData: inout std_logic_vector(7 downto 0);
     memWb, memEn, memCe, fin: out std_logic
  );
end memMult;
architecture a of memMult is
  type tipo_estado is (s0,s1,s2,s3,s4,s5,s6) ;
  signal estado : tipo_estado ;
  signal z_tmp, memIn_D : std_logic_vector(7 downto 0);
  signal memIn_Q, memOut, mn_D : std_logic_vector(7
downto 0);
  signal mn_Q, contadorL_Q, contadorL_D :
std_logic_vector(7 downto 0);
  signal z_D, z_Q, mul16 : std_logic_vector(15 downto 0);
  signal contadorH_Q, contadorH_D: std_logic_vector(8
downto 0);
begin
  process(clk, reset)
  begin
     if reset='1' then
        estado <= s0;
     elsif rising_edge(clk) then
        case estado is
           when s0 =>
              if comenzar='1' then
                    estado<=s1;
              else
                    estado<=s0;
              end if;
           when s1 =>
              estado<=s2;
           when s2 =>
              estado<=s3;
           when s3 =>
              estado<=s4;
           when s4 =>
              estado<=s5;
           when s5 =>
              if contadorH_D(8) = '1' then
                    estado<=s6;
              else
                    estado<=s1;
              end if;
           when s6 =>
                    estado <= s0;
        end case;
     end if;
  end process;
  process(clk, reset)
  begin
        if reset='1' then
              memIn_Q <= "00000000";
              mn_Q <= "00000000";
              z_Q <= "0000000000000000";
              contadorL_Q <= "00000000";
              contadorH_Q <= "000000000";
        elsif rising_edge(clk) then
              memIn_Q <= memIn_D;
              z_Q <= z_D;
              mn_Q <= mn_D;
              contadorL_Q <= contadorL_D;
              contadorH_Q <= contadorH_D;
        end if;
  end process;

  z_tmp(7 downto 0) <= z_Q(15 downto 8);
  memData <= z_Q(7 downto 0) when estado = s4
                    else  z_tmp when estado = s5
                    else   "ZZZZZZZZ";
  memIn_D<= memData;
  fin <= '1' when estado = s6 else '0';
  memEn <= '0' when (estado = s1) or (estado = s2) else '1';
  memCe <= clk when (estado = s1) or (estado = s2) or
                          (estado = s4) or (estado = s5)
                  else '1';
  memWb <= clk when (estado = s4) or (estado = s5)
                   else '1';
  memAddr <= contadorL_Q when (estado = s1) or
                  (estado = s4) else contadorH_Q(7 downto 0)
                   when (estado = s2) or (estado = s5) else
                  "00000000";
  contadorL_D <=  contadorL_Q +2 when (estado = s5) else
                "00000000" when estado = s0 else
                contadorL_Q;
  contadorH_D <=  contadorH_Q +2 when estado = s5 else
                "00000001" when estado = s0 else
                contadorH_Q;
  z_D<=(("00000000" & mn_Q) * ("00000000" & memIn_Q))
           when estado = s3 else z_Q;
  mn_D<= memIn_Q when estado = s2 else mn_Q;
end a;
```

## Ejemplo de RAM tipo FIFO

### Versión 1 en VHDL

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity fifoV is
  port(
        fifo_llena, fifo_vacia :out std_logic;
        clk, reset_fifo, reset, leer, escribir : in std_logic;
        datoOut : out std_logic_vector(7 downto 0);
        datoIn : in std_logic_vector(7 downto 0);
        esp_ocup : out std_logic_vector (5 downto 0)
  );
end fifoV;
architecture a of fifoV is
  COMPONENT LPM_FIFO
     GENERIC (LPM_WIDTH: POSITIVE;
          LPM_WIDTHU: POSITIVE := 1;
          LPM_TYPE: STRING := "LPM_FIFO";
          LPM_NUMWORDS: POSITIVE;
          LPM_SHOWAHEAD: STRING := "OFF";
          LPM_HINT: STRING := "UNUSED");
     PORT (  data: IN STD_LOGIC_VECTOR(LPM_WIDTH-
1 DOWNTO 0);
          clock, rdreq, wrreq: IN STD_LOGIC;
          aclr, sclr: IN STD_LOGIC := '0';
          full, empty: OUT STD_LOGIC;
          usedw: OUT
STD_LOGIC_VECTOR(LPM_WIDTHU-1 DOWNTO 0);
          q: OUT STD_LOGIC_VECTOR(LPM_WIDTH-1
DOWNTO 0));
  END COMPONENT;
  signal resFifo : std_logic;
begin
  lpmF : lpm_fifo generic map( lpm_width => 8,
lpm_numwords => 64, lpm_widthu => 6)
            port map (data => datoIn, wrreq =>escribir ,
                      rdreq=>leer ,
                      clock => clk, sclr => resFifo,
                      empty => fifo_vacia, full => fifo_llena,
                      q => datoOut, usedw => esp_ocup);

  resFifo <= '1' when ((reset ='1') or (reset_fifo ='1')) else '0';
end a;
```

### Versión 1 para HCC

```
// fifo usando la ram provista por handelC
const spec tgt={  //espec. del target
   fpga_type="Xilinx4000",
   clock_pad="P160"};

void main( target=tgt,
        port (in) escribir: 1, //decl. de puertos
        port (in) leer : 1,
        port (in) reset_fifo: 1,
        port (out) fifo_llena:1,
        port (out) fifo_vacia:1,
        port (in) datoIn : 8,
        port (out) esp_ocup : 6,
        port (out) datoOut : 8){

   int ent, sal : 8; /* variables para almacenar los datos */
   int contE, contL:6; /* contadores de  escritura y lectura */
   int usedw:7; //palabras usadas
   int wrreq, rdreq:1; // pulsos internos de lectura/escritura
   int hab_lec, hab_esc:1;//señales de habilitación/deshab.
   ram x [64]:8;  // ram interna

   par{
      while(1){
         par{
            esp_ocup ! usedw.(0..5); /* escribo  en puerto */
            //escribo o leo en la ram segun wrreq
            if (wrreq) x[contE]=ent;
            else if (rdreq) datoOut ! x[contL];
            else delay;
            fifo_llena ! hab_esc; //escribo puerto
            fifo_vacia ! ~hab_lec; //escribo  puerto
// asignaciónes cond. según reset_fifo  , wrreq/rdreq
            contE=(reset_fifo ? 0 : (wrreq?contE+1:contE));
            contL=(reset_fifo ? 0 : (rdreq ?contL+1:contL));
         }
      };
      while(1){
         if (reset_fifo==1)
            hab_lec,hab_esc,wrreq,rdreq,usedw=0,0,0,0,0;
         else par{
            datoIn ? ent; //registro datoIn
            case(hab_lec @ leer @ ~hab_esc @ escribir){
                  /* hab_esc la trabajo al reves que hab_lec
                   xq'  sino al principio indica fifo_llena */
                  12,13,14:wrreq,rdreq,usedw=0,1,usedw-1;
                  3, 7,11: wrreq,rdreq,usedw=1,0,usedw+1;
                  15:{
                     wrreq, rdreq = 0, 1;
                     wrreq, rdreq = 1, 0;
                   }
                  default : wrreq, rdreq = 0, 0;
            };
          /*deshabilito lectura cuando queda una palabra  y se
            hace una lectura o cuando no quedan  palabras */
            hab_lec=(((usedw==1)&leer)|(usedw==0)?0:1);
         /*deshabilito lectura cuando queda un espacio y se
           hace una escritura o cuando no quedan espacios */
            hab_esc=(((usedw==63)&escribir)|usedw.6?1:0);
         }
      };
   }
}
```

### Versión 2 para HCC

```
// fifo usando la ram provista por handelC
const spec tgt={  //espec. del target
   fpga_type="Xilinx4000",
   clock_pad="P160"};

void main( target=tgt,
        port (in) escribir: 1, //decl. de puertos
        port (in) leer : 1,
        port (in) reset_fifo: 1,
        port (out) fifo_llena:1,
        port (out) fifo_vacia:1,
        port (in) datoIn : 8,
        port (out) esp_ocup : 6,
        port (out) datoOut : 8,
        port (out) ram_fifo_we : 1,
        port (out) ram_fifo_din : 8,
```

```
        port (in)  ram_fifo_dot : 8,
        port (out) ram_fifo_dir : 6)
{

    int ent, sal : 8; /* variables para almacenar los datos */
    int contE, contL:6; /* contadores de  escritura y lectura */
    int usedw:7; //palabras usadas
    int wrreq, rdreq:1; // pulsos internos de lectura/escritura
    int hab_lec, hab_esc:1;//señales de habilitación/deshab.
    int rdreq_sh:1;  // ram interna

    par{
       while(1){
          par{
             esp_ocup ! usedw.(0..5); /* escribo  en puerto */
//escribo o leo en la ram segun wrreq
             ram_fifo_din ! ent;  // asigno ent a din de la ram
             ram_fifo_dir  !  (wrreq  ?  contE  :  contL);      /*  si
wrreq=1 dir=contE si no dir=contL*/
             ram_fifo_we  !  wrreq;   /*  pulso  de  we  de  la  ram,
coincide con wrreq*/
             datoOut ! (rdreq_sh ? ram_fifo_dot : 0);
             rdreq_sh = rdreq;
             fifo_llena ! hab_esc; //escribo puerto
             fifo_vacia ! ~hab_lec; //escribo  puerto
// asignaciónes cond. según reset_fifo  , wrreq/rdreq
             contE=(reset_fifo ? 0 : (wrreq?contE+1:contE));
             contL=(reset_fifo ? 0 : (rdreq ?contL+1:contL));
          }
       };
       while(1){
          if (reset_fifo==1)
             hab_lec,hab_esc,wrreq,rdreq,usedw=0,0,0,0,0;
          else par{
             datoIn ? ent; //registro datoIn
             case(hab_lec @ leer @ ~hab_esc @ escribir){
                   /* hab_esc la trabajo al reves que hab_lec
                    xq'  sino al principio indica fifo_llena */
                  12,13,14:wrreq,rdreq,usedw=0,1,usedw-1;
                  3, 7,11: wrreq,rdreq,usedw=1,0,usedw+1;
                  15:{
                     wrreq, rdreq = 0, 1;
                     wrreq, rdreq = 1, 0;
                   }
                  default : wrreq, rdreq = 0, 0;
             };
          /*deshabilito lectura cuando queda una palabra  y se
            hace una lectura o cuando no quedan  palabras */
             hab_lec=(((usedw==1)&leer)|(usedw==0)?0:1);
          /*deshabilito lectura cuando queda un espacio y se
            hace una escritura o cuando no quedan espacios */
             hab_esc=(((usedw==63)&escribir)|usedw.6?1:0);
          }
       };
    }
}
```

## Versión para TMCC

```
main(){
    int escribir, leer, reset_fifo, fifo_llena;
    int fifo_vacia, int wrreq, rdreq, rdreq_sh,
    int ram_fifo_we, segunda, seg_sh;
    int hab_esc, hab_escQ, hab_lecQ, hab_lec;
#pragma intbits 8
    int datoIn, datoIn_sh, datoOut;
    int ram_fifo_din, ram_fifo_dot;
#pragma intbits 7
    int usedw;
#pragma intbits 6
    int contE, contL, ram_fifo_dir, esp_ocup;

    inputport(datoIn); inputport(escribir);
    inputport(leer);inputport(reset_fifo);
    outputport(datoOut); outputport(fifo_llena);
    outputport(fifo_vacia); outputport(esp_ocup);
    outputport(ram_fifo_dir);inputport(ram_fifo_dot);
    outputport(ram_fifo_we);outputport(ram_fifo_din);
// declaro datoIn como registrado
    portflags(datoIn, 0x3);
// declaro puertos varios como no registrados
    portflags(fifo_llena, 0x1);  portflags(fifo_vacia, 0x1);
    portflags(datoOut, 0x1);  portflags(esp_ocup, 0x1);
    portflags(ram_fifo_dir, 0x1);
    portflags(ram_fifo_din,0x1);
    portflags(ram_fifo_we,0x1);

    while(1){
       if (reset_fifo == 1) {
          contE = 0; contL = 0;
          esp_ocup = 0; fifo_llena = 0;
          fifo_vacia = 1; usedw = 0;
          hab_esc = 0; hab_lec = 0;
       }
       else{
          /* si en el ciclo anterior wrreq=rdreq=1 entonces
             uso datoIn_sh, si no uso datoIn */
          if (seg_sh==1)  ram_fifo_din=datoIn_sh;
          else   ram_fifo_din=datoIn;
          if (wrreq){
             ram_fifo_dir= contE;
             contE++;}
          else {ram_fifo_dir = contL;};
          if (rdreq) contL++;
          /* si en el pulso anterior rdreq=1 doy como salida
             ram_fifo_dot */
          if (rdreq_sh) {datoOut=ram_fifo_dot;}
          else{    datoOut=0;};
          ram_fifo_we=wrreq;
          esp_ocup = usedw;
          fifo_vacia = ~hab_lec; //esc. en puerto
          fifo_llena = hab_esc;  //esc. en puerto
          rdreq_sh = rdreq;  // rdreq_sh = rdreq retrasada
          seg_sh = segunda;
          datoIn_sh = datoIn;
          if (((usedw==63) & (escribir==1)) | usedw==64){
             /* si queda un espacio y escribo o si no  queda
                espacio se deshab. la escritura */
             hab_esc = 1;
             hab_lec = 1;
          }
          else if (((usedw==1) & (leer==1)) | (usedw==0)){
             /* si queda una palabra y leo o si no quedan
                palabras se deshab. la lectura */
             hab_esc = 0;
             hab_lec = 0;
          }
          else {
             hab_esc = 0;
             hab_lec = 1;
          };
          /* genero pulsos internos de lec/esc y actualizo
             palabras usadas segun  habilitaciones y puertos */
          if ((~hab_escQ & escribir &
                      ((~hab_lecQ)|(~leer))&(~segunda))==1){
             wrreq=1; rdreq=0; usedw=usedw+1; segunda=0;
          }
          else if ((hab_lecQ & leer &
                    ((~escribir)|(hab_escQ))&(~segunda))==1){
             wrreq=0; rdreq=1; usedw=usedw-1; segunda=0;
          }
          else if (((hab_lecQ & leer & ~hab_escQ& escribir) |
                                                  segunda)==1){
```

```
            wrreq=segunda; rdreq=~segunda;
            segunda=~segunda;
        }
        else{   segunda=0;
                wrreq=0; rdreq=0;
        };
        hab_escQ = hab_esc;
        hab_lecQ = hab_lec;
    };
};
}
```

## Versión 2 en VHDL

```
-- fifo version VHDL usando lpm_ram
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity fifoVram is
    port(
            fifo_llena, fifo_vacia :out std_logic;
            clk, reset_fifo, leer, escribir : in std_logic;
            datoOut : out std_logic_vector(7 downto 0);
            datoIn : in std_logic_vector(7 downto 0);
            esp_ocup: out std_logic_vector(5 downto 0)
    );
end fifoVram;
architecture a of fifoVram is
    component lpm_ram_dq
        generic (lpm_width: positive;
            lpm_widthad: positive;
            lpm_outdata: string := "unregistered"
        );
        port (
            data:in std_logic_vector(lpm_width-1 downto 0);
            address: in std_logic_vector(lpm_widthad-1
                                                    downto 0);
            we: in std_logic;
            inclock: in std_logic:='0';
            q: out std_logic_vector(lpm_width-1 downto 0));
    end component;

    signal  ramWe, aux, rdreq, hab_lec: std_logic;
    signal hab_esc, wrreq, rdwrreq, rdreq_sh: std_logic;
    signal ramQ, datoOut_i, ent : std_logic_vector(7
                                                    downto 0);
    signal ramAddr: std_logic_vector(5 downto 0);
    signal contL_Q, contL_D: std_logic_vector (5 downto 0);
    signal contE_Q,contE_D: std_logic_vector (5 downto 0);
    signal usedw : std_logic_vector(6 downto 0);
    signal grupoC : std_logic_vector (3 downto 0);

begin
    lpmF : lpm_ram_dq
        generic map( lpm_width => 8, lpm_widthad => 6)
        port map (data => ent, we =>ramWe,
                    address => ramAddr, inclock => clk,
                    q => datoOut_i);

    ramWe <= '1' when (wrreq='1' or aux='0') else '0';
    ramAddr <= contE_Q when (wrreq = '1' or aux='0')
                    else contL_Q;
    contL_D <=  "000000" when reset_fifo='1' else
                    contL_Q + '1' when
                            ((rdreq or rdwrreq) and aux)='1'
                    else contL_Q;
    contE_D <= "000000" when reset_fifo='1' else
                    contE_Q + '1' when ((wrreq = '1')
                        and (aux='1')) or (aux='0') else
                    contE_Q;
    datoOut <= datoOut_i when rdreq_sh='1' else
                    "00000000";
    esp_ocup <= usedw(5 downto 0);
    fifo_llena <= hab_esc;
    fifo_vacia <= not hab_lec;

    grupoC <= (hab_lec & leer & hab_esc & escribir);

    process(clk, reset_fifo)
    begin
        if reset_fifo = '1' then
            aux <= '1';
            usedw <= "0000000";
            rdreq<= '0';
            rdreq_sh <= '0';
            hab_lec <= '0';
            hab_esc <= '0';
            wrreq <= '0';
            rdwrreq <= '0';
        elsif rising_edge(clk) then
            if (rdwrreq='1') then
                aux <= not aux;
                else aux <= '1';
            end if;
            if ((rdwrreq and  aux)='0') then
                ent <= datoIn;
            end if;
            rdreq_sh <= rdreq or (rdwrreq and aux);
            contL_Q <= contL_D;
            contE_Q <= contE_D;

            if (((usedw = 1) and (leer = '1')) or(usedw = 0)) then
                hab_lec <= '0';
            else  hab_lec <= '1';
        end if;

        if (((usedw= 63)and(escribir = '1'))or usedw = 64) then
            hab_esc <= '1';
        else  hab_esc <= '0';
        end if;

        case (grupoC) is
            when "1110" | "1100" | "1111" =>
                wrreq <= '0' ;
                rdreq <= '1' ;
                usedw <= usedw - '1' ;
                rdwrreq <= '0';
            when "0001" | "0101" | "1001" =>
                wrreq <= '1' ;
                rdreq <= '0' ;
                usedw <= usedw + '1' ;
                rdwrreq <= '0';
            when "1101" =>
                rdwrreq <= '1';
            when others =>
                wrreq <= '0' ; rdreq <= '0' ; rdwrreq <= '0';
        end case;
    end if;
  end process;
end a;
```

## Ejemplos de uso de pipeline

### Versión 1 para HCC

```
const spec yyy = {
    fpga_type  = "Xilinx4000",
    fpga_chip  = "4003APC84-6",
    clock_pad = "P13"
    };
const spec ent={ data={} };
const spec sal={ data={} };
const dw=8;
const ddw=dw*2;
const zero=0:dw;

main ( target = yyy,
    port (in) mp=ent :dw,
    port (in) mr=ent :dw,
    port (out) res=sal : ddw)
{
    int  ta,tb:dw;
    int tr : ddw;

    while (true) par {
                    // leo operandos de los puertos
                    mp ? ta;
                    mr ? tb;
                    tr = (ta .*. tb);
                    // devuelvo resultado por el puerto
                    res ! tr;
    };
}
```

### Versión 2 para HCC

```
const spec yyy = {
    fpga_type   = "Xilinx4000",
    fpga_chip   = "4003APC84-6",
    clock_pad  = "P13"
    };
const spec ent={ data={} };
const spec sal={ data={} };
const dw=8;
const ddw=dw*2;
const zero=0:dw;

main ( target = yyy,
    port (in) mp=ent :dw,
    port (in) mr=ent :dw,
    port (out) res=sal : ddw)
{
    int ta:dw;
    int temp0,temp1,temp2,temp3,temp4 : ddw;
    int temp5,temp6,temp7,tb,tr : ddw;

   while (true) par {
            mp ? ta;
            mr ? tb;
            ta=a;
            tb=; // aumento ancho de b
                    // nuevo producto de a y b
            {   par {  // realizo rotaciones
                    temp0 = (ta.0 ? (zero @ tb) : 0);
                    /* si el bit 1 de ta es 1 entonces
                     temp1 es rotado 1 vez */
                    temp1 = (ta.1 ? (zero @ tb) << 1 : 0);
                    temp2 = (ta.2 ? (zero @ tb) << 2 : 0);
                    temp3 = (ta.3 ? (zero @ tb) << 3 : 0);
                    temp4 = (ta.4 ? (zero @ tb) << 4 : 0);
                    temp5 = (ta.5 ? (zero @ tb) << 5 : 0);
                    temp6 = (ta.6 ? (zero @ tb) << 6 : 0);
                    temp7 = (ta.7 ? (zero @ tb) << 7 : 0);
                };
                 // sumo todos los resultados parciales
                tr= temp0+temp1+temp2+temp3+
                            temp4+temp5+temp6+temp7;
            };  // fin de la implementacion del producto
            res ! tr;
     }; // fin del while
}
```

### Versión 3 para HCC

```
const spec yyy = {
    fpga_type   = "Xilinx4000",
    fpga_chip   = "4003APC84-6",
    clock_pad  = "P13"
    };
const spec ent={ data={} };
const spec sal={ data={} };
const dw=8;
const ddw=dw*2;
const zero=0:dw;

main( target = yyy,
    port (in) mp=ent :dw,
    port (in) mr=ent :dw,
    port (out) res=sal : ddw)
{
    int ta:dw;
    int temp0,temp1,temp2,temp3,temp4 : ddw;
    int temp5,temp6,temp7,tb,tr : ddw;

    res ! (zero @ a);
    while (true) par {
            mp ? ta;  // leo del Puerto el multiplicando
            mr ? tb;  // leo del Puerto el multiplicador
             // rotaciones
            temp0 = (ta.0 ? (zero @ b): 0);
            temp1 = (ta.1 ? (zero @ b) << 1 : 0);
            temp2 = (ta.2 ? (zero @ b) << 2 : 0);
            temp3 = (ta.3 ? (zero @ b) << 3 : 0);
            temp4 = (ta.4 ? (zero @ b) << 4 : 0);
            temp5 = (ta.5 ? (zero @ b) << 5 : 0);
            temp6 = (ta.6 ? (zero @ b) << 6 : 0);
            temp7 = (ta.7 ? (zero @ b) << 7 : 0);
            // sumas
            tr= temp0+temp1+temp2+temp3+
                                temp4+temp5+temp6+temp7;
            res ! tr;  // salida del resultado por puerto
    };
}
```

## Versión 4 para HCC

```
// 4 etapas
const spec yyy = {
    fpga_type   = "Xilinx4000",
    fpga_chip   = "4003APC84-6",
    clock_pad  = "P13"
    };
const spec ent={ data={} };
const spec sal={ data={} };
const zero=0:8;

main (  target = yyy,
    port (in) mp=ent :8,
    port (in) mr=ent :8,
    port (out) res=sal :16)
{
    int ta,tb:8;
    int tmp0,tmp3,bux:16;
    int tmp1,tmp2,tmp4,tmp5,tmp6,tmp7:16;
    int sum0,sum1,sum2:16;

    res ! tmp0;
    while (true) {
        par {
            mp ? ta;
            mr ? tb;
            // rotaciones
            tmp0 = (ta.0 ? (zero @ tb) : 0);
            tmp1 = (ta.1 ? (zero @ tb) << 1 : 0);
            tmp2 = (ta.2 ? (zero @ tb) << 2 : 0);
            tmp3 = (ta.3 ? (zero @ tb) << 3 : 0);
            tmp4 = (ta.4 ? (zero @ tb) << 4 : 0);
            tmp5 = (ta.5 ? (zero @ tb) << 5 : 0);
            tmp6 = (ta.6 ? (zero @ tb) << 6 : 0);
            tmp7 = (ta.7 ? (zero @ tb) << 7 : 0);
             // primer nivel de sumas
            sum0 = tmp3 + tmp0 + tmp5 + tmp7;
            sum1 = tmp2 + tmp1 + tmp4 + tmp6;
            // Segundo nivel de sumas
            sum2 = sum0 + sum1;
            res ! sum2;  // devuelvo resultado por el puerto
        }
    };
}
```

## Versión 5.1 para HCC

```
// 5 etapas
const spec yyy = {
    fpga_type   = "Xilinx4000",
    fpga_chip   = "4003APC84-6",
    clock_pad  = "P13"
    };
const spec ent={ data={} };
const spec sal={ data={} };
const zero=0:8;

main (  target = yyy,
    port (in) mp=ent :8,
    port (in) mr=ent :8,
    port (out) res=sal :16)
{
    int ta,tb:8;
    int tmp0,tmp3,bux:16;
    int tmp1,tmp2,tmp4,tmp5,tmp6,tmp7:16;
    int sum00,sum01,sum02,sum03:16;
    int sum10,sum11,sum2:16;

    res ! tmp0;
    while (true) {
        par{
            mp ? ta;
            mr ? tb;
            // rotaciones
            tmp0 = (ta.0 ? (zero @ tb) : 0);
            tmp1 = (ta.1 ? (zero @ tb) << 1 : 0);
            tmp2 = (ta.2 ? (zero @ tb) << 2 : 0);
            tmp3 = (ta.3 ? (zero @ tb) << 3 : 0);
            tmp4 = (ta.4 ? (zero @ tb) << 4 : 0);
            tmp5 = (ta.5 ? (zero @ tb) << 5 : 0);
            tmp6 = (ta.6 ? (zero @ tb) << 6 : 0);
            tmp7 = (ta.7 ? (zero @ tb) << 7 : 0);
             // primer nivel de sumas
            sum00 = tmp3 + tmp0 ;
            sum01 = tmp2 + tmp1 ;
            sum02 = tmp5 + tmp7;
            sum03 = tmp4 + tmp6;
             // Segundo nivel de sumas
            sum10 = sum00 + sum01;
            sum11 = sum02 + sum03;
             // tercer nivel de sumas
            sum2   = sum10 + sum11;
            res ! sum2;  // salida de resultado por puerto
        }
    };
}
```

## Versión 5.2 para HCC

```
// 5 etapas
const spec yyy = {
    fpga_type   = "Xilinx4000",
    fpga_chip   = "4003APC84-6",
    clock_pad  = "P13"
    };
const spec ent={ data={} };
const spec sal={ data={} };
const zero8=0:8;  // constantes con ancho en bits
const zero1=0:1;
const zero2=0:2;
const zero3=0:3;
const zero4=0:4;
const zero5=0:5;
const zero6=0:6;
const zero7=0:7;

main ( target = yyy,
    port (in) mp=ent :8,
    port (in) mr=ent :8,
    port (out) res=sal :16)
{
    int ta,tb, tmp0:8;
    int tmp1,sum00:10;
    int tmp2:11;
    int tmp3,sum01,sum10:12;
    int tmp4:13;
    int tmp5,sum02:14;
    int tmp6:15;
    int tmp7,sum03,sum11:16;
    int sum2:16;
```

```
    res ! tmp7;
    while(true){
        par {
            mp ? ta;
            mr ? tb;
            // rotaciones
            tmp0 = (ta.0 ? tb : 0);
            tmp1 = (ta.1 ? (zero2 @ tb) << 1 : 0);
            tmp2 = (ta.2 ? (zero3 @ tb) << 2 : 0);
            tmp3 = (ta.3 ? (zero4 @ tb) << 3 : 0);
            tmp4 = (ta.4 ? (zero5 @ tb) << 4 : 0);
            tmp5 = (ta.5 ? (zero6 @ tb) << 5 : 0);
            tmp6 = (ta.6 ? (zero7 @ tb) << 6 : 0);
            tmp7 = (ta.7 ? (zero8 @ tb) << 7 : 0);
            // primer nivel de sumas
            sum00 = (0 @ tmp0) + tmp1;
            sum01 = (0 @ tmp2) + tmp3 ;
            sum02 = (0 @ tmp4) + tmp5;
            sum03 = (0 @ tmp6) + tmp7;
            // Segundo nivel de sumas
            sum10 = (0 @ sum00) + sum01;
            sum11 = (0 @ sum02) + sum03;
            // tercer nivel de sumas
            sum2   = (0 @ sum10) + sum11;
            res ! sum2;  // salida de resultado por puerto
        }
    };
}
```

## Versión 6.1 en VHDL

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
LIBRARY lpm;
USE lpm.lpm_components.ALL;

entity lpm1 is
    port (
        INPUT_CLOCK:in std_logic;
        globalReset: std_logic;
        mp:in std_logic_vector(7 downto 0);
        mr:in std_logic_vector(7 downto 0);
        res:out std_logic_vector(15 downto 0)
    );
end lpm1;

architecture netlist of lpm1 is
  COMPONENT lpm_mult
  GENERIC (LPM_WIDTHA: POSITIVE;
    LPM_WIDTHB: POSITIVE;
    LPM_WIDTHS: NATURAL := 1;
    LPM_WIDTHP: POSITIVE;
    LPM_REPRESENTATION: STRING := "UNSIGNED";
    LPM_PIPELINE: INTEGER := 0;
    LPM_TYPE: STRING := "LPM_MULT";
    LPM_HINT: STRING := "UNUSED");
  PORT (dataa: IN STD_LOGIC_VECTOR(LPM_WIDTHA-
1 DOWNTO 0);
    datab:    IN    STD_LOGIC_VECTOR(LPM_WIDTHB-1
DOWNTO 0);
    aclr, clock: IN STD_LOGIC := '0';
    clken: IN STD_LOGIC := '1';
    sum:     IN     STD_LOGIC_VECTOR(LPM_WIDTHS-1
DOWNTO 0) := (OTHERS => '0');
    result:   OUT   STD_LOGIC_VECTOR(LPM_WIDTHP-1
DOWNTO 0));
  END COMPONENT;
  signal ta,tb : std_logic_vector(7 downto 0);
  signal tr : std_logic_vector(15 downto 0);
  signal clk : std_logic;
begin
    clk <= INPUT_CLOCK;
    reg12a  : lpm_ff
        GENERIC MAP (LPM_WIDTH => 8)
        PORT MAP (data => mp(7 DOWNTO 0),
                clock => clk,
                q => ta(7 DOWNTO 0));
    reg12b  : lpm_ff
        GENERIC MAP (LPM_WIDTH => 8)
        PORT MAP (data => mr(7 DOWNTO 0),
                clock => clk,
                q => tb(7 DOWNTO 0));
    prod :  lpm_mult
        generic map (LPM_WIDTHA => 8,
            LPM_WIDTHB => 8,
            LPM_WIDTHS => 1,
            LPM_WIDTHP => 16,
            LPM_REPRESENTATION => "UNSIGNED",
            LPM_PIPELINE => 0,
            LPM_TYPE => "LPM_MULT")
        port map (dataa => ta, datab => tb, result => tr);
    reg12r  : lpm_ff
        GENERIC MAP (LPM_WIDTH => 16)
        PORT MAP (data => tr, clock => clk,
                q => res);
end netlist;
```

## Versión 6.2 en VHDL

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
LIBRARY lpm;
USE lpm.lpm_components.ALL;

entity lpm1 is
    port (
        INPUT_CLOCK:in std_logic;
        globalReset: std_logic;
        mp:in std_logic_vector(7 downto 0);
        mr:in std_logic_vector(7 downto 0);
        res:out std_logic_vector(15 downto 0)
    );
end lpm1;
```

```
architecture netlist of lpm1 is
  COMPONENT lpm_mult
  GENERIC (LPM_WIDTHA: POSITIVE;
    LPM_WIDTHB: POSITIVE;
    LPM_WIDTHS: NATURAL := 1;
    LPM_WIDTHP: POSITIVE;
    LPM_REPRESENTATION: STRING := "UNSIGNED";
    LPM_PIPELINE: INTEGER := 0;
    LPM_TYPE: STRING := "LPM_MULT";
    LPM_HINT: STRING := "UNUSED");
  PORT (dataa: IN STD_LOGIC_VECTOR(LPM_WIDTHA-
1 DOWNTO 0);
    datab: IN STD_LOGIC_VECTOR(LPM_WIDTHB-1
DOWNTO 0);
    aclr, clock: IN STD_LOGIC := '0';
    clken: IN STD_LOGIC := '1';
```

```
    sum: IN STD_LOGIC_VECTOR(LPM_WIDTHS-1
DOWNTO 0) := (OTHERS => '0');
    result: OUT STD_LOGIC_VECTOR(LPM_WIDTHP-1
DOWNTO 0));
  END COMPONENT;
  signal ta,tb : std_logic_vector(7 downto 0);
  signal tr : std_logic_vector(15 downto 0);
  signal clk : std_logic;
begin
      clk <= INPUT_CLOCK;
      reg12a  : lpm_ff
            GENERIC MAP (LPM_WIDTH => 8)
            PORT MAP (data => mp(7 DOWNTO 0),
                  clock => clk,
                  q => ta(7 DOWNTO 0));
      reg12b  : lpm_ff
            GENERIC MAP (LPM_WIDTH => 8)
            PORT MAP (data => mr(7 DOWNTO 0),
                  clock => clk,
                  q => tb(7 DOWNTO 0));
      prod :  lpm_mult
            generic map (LPM_WIDTHA => 8,
                  LPM_WIDTHB => 8,
                  LPM_WIDTHS => 1,
                  LPM_WIDTHP => 16,
                  LPM_REPRESENTATION => "UNSIGNED",
                  LPM_PIPELINE => 1,
                  LPM_TYPE => "LPM_MULT")
            port map (dataa => ta, datab => tb, result => tr);
      reg12r  : lpm_ff
            GENERIC MAP (LPM_WIDTH => 16)
            PORT MAP (data => tr, clock => clk,
                  q => res);
end netlist;
```

## Versión 6.3 en VHDL

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
LIBRARY lpm;
USE lpm.lpm_components.ALL;

entity lpm1 is
    port (
          INPUT_CLOCK:in std_logic;
          globalReset: std_logic;
          mp:in std_logic_vector(7 downto 0);
          mr:in std_logic_vector(7 downto 0);
          res:out std_logic_vector(15 downto 0)
    );
end lpm1;

architecture netlist of lpm1 is
  COMPONENT lpm_mult
  GENERIC (LPM_WIDTHA: POSITIVE;
    LPM_WIDTHB: POSITIVE;
    LPM_WIDTHS: NATURAL := 1;
    LPM_WIDTHP: POSITIVE;
    LPM_REPRESENTATION: STRING := "UNSIGNED";
    LPM_PIPELINE: INTEGER := 0;
    LPM_TYPE: STRING := "LPM_MULT";
    LPM_HINT: STRING := "UNUSED");
  PORT (dataa: IN STD_LOGIC_VECTOR(LPM_WIDTHA-
1 DOWNTO 0);
    datab: IN STD_LOGIC_VECTOR(LPM_WIDTHB-1
DOWNTO 0);
    aclr, clock: IN STD_LOGIC := '0';
    clken: IN STD_LOGIC := '1';
    sum: IN STD_LOGIC_VECTOR(LPM_WIDTHS-1
DOWNTO 0) := (OTHERS => '0');
    result: OUT STD_LOGIC_VECTOR(LPM_WIDTHP-1
DOWNTO 0));
  END COMPONENT;
  signal ta,tb : std_logic_vector(7 downto 0);
  signal tr : std_logic_vector(15 downto 0);
  signal clk : std_logic;
begin
      clk <= INPUT_CLOCK;
      reg12a  : lpm_ff
            GENERIC MAP (LPM_WIDTH => 8)
            PORT MAP (data => mp(7 DOWNTO 0),
                  clock => clk,
                  q => ta(7 DOWNTO 0));
      reg12b  : lpm_ff
            GENERIC MAP (LPM_WIDTH => 8)
            PORT MAP (data => mr(7 DOWNTO 0),
                  clock => clk,
                  q => tb(7 DOWNTO 0));
      prod :  lpm_mult
            generic map (LPM_WIDTHA => 8,
                  LPM_WIDTHB => 8,
                  LPM_WIDTHS => 1,
                  LPM_WIDTHP => 16,
                  LPM_REPRESENTATION => "UNSIGNED",
                  LPM_PIPELINE => 3,
                  LPM_TYPE => "LPM_MULT")
            port map (dataa => ta, datab => tb, result => tr);
      reg12r  : lpm_ff
            GENERIC MAP (LPM_WIDTH => 16)
            PORT MAP (data => tr, clock => clk,
                  q => res);
end netlist;
```

## Versión para TMCC

```
main(){
#pragma intbits 4
    int mp,mr;
#pragma intbits 8
    int sal;
    int tmp0,tmp1,tmp2,tmp3;
    int sum0,sum1,sum2;
    inputport(mp); inputport(mr);
    outputport(sal);
    while (1) {
        sal     = sum0 + sum1;
        sum0 = tmp3 + tmp0;
        sum1 = tmp2 + tmp1;
        if ((mp & 0x1)==0x1) {tmp0=mr;} else {tmp0=0;};
        if ((mp & 0x2)==0x2) {tmp1=mr<<1;} else {tmp1=0;};
        if ((mp & 0x4)==0x4) {tmp2=mr<<2;} else {tmp2=0;};
        if ((mp & 0x8)==0x8) {tmp3=mr<<3;} else {tmp3=0;};
    }
}
```

## Operaciones con matrices: producto entre elementos de matrices

Ejemplo 1, versión 1 para HCC

```
const spec harp1 ={
            fpga_type= "Xilinx4000",
            clock_pad= "P13"    };

const dim = 3;
const dw = 8;
const ddw = 16;

main(       target=harp1,
            //defino puertos de entrada y salida
           port (in) datos  : dw,
           port (out) R0 : ddw)
{
            //defino dimensiones de la memoria interna
            ram int A[(dim*dim)] : dw;
            ram int B[(dim*dim)] : dw;
            int aux:ddw;
            int i : 4;
            // guardo en memoria el valor de la matriz B cte
            B[0] = -1; B[1] = 1; B[2] = 1; B[3] = -1; B[4] = 1;
            B[5] = -1; B[6] = 1; B[7] = -1; B[8] = 1;

            //Ingreso los valores de la matriz A (3x3)
            i=0;
            while (i.<.9) {par{
                  i = i+1;
                  datos ? A[i];} }
             //realizo las operaciones con un acumulador
             i,aux=0,0;
             while (i.<.9)
             {par { aux = aux + (A[i]*B[i]); i = i+1;}}
            R0 ! aux; // escribo el  resultado por el puerto R0
}
```

Ejemplo 1, versión 2 para HCC

```
const spec harp1 ={
            fpga_type= "Xilinx4000",
            clock_pad= "P13"    };

const dim = 3;
const dw = 8;
const ddw = 16;

main ( target=harp1,
          port (in) datos  : dw,
          port (out) R0  : ddw) {

     ram int A[(dim*dim)] : dw;
     int aux:ddw;
     int i : 4;
     // valor de los elementos de la matriz B cte.
     int B0 = -1;int B1 = 1;int B2 = 1;int B3 = -1;int B4 = 1;
int B5 = -1;int B6 = 1;int B7 = -1;int B8 = 1;
        //Ingreso los valores de la matriz A (3x3)
        while (i.<.9)
              par {datos ? A[i]; i=i+1; }

        //realizo las operaciones con un acumulador
        aux = (A[0]*B0) ;
        aux = aux + (A[1]*B1) ;
        aux = aux + (A[2]*B2) ;
        aux = aux + (A[3]*B3) ;
        aux = aux + (A[4]*B4) ;
        aux = aux + (A[5]*B5) ;
        aux = aux + (A[6]*B6) ;
        aux = aux + (A[7]*B7) ;
        aux = aux + (A[8]*B8) ;
        R0 ! aux;
}
```

Ejemplo 1, versión 3 para HCC

```
const spec harp1 ={
            fpga_type= "Xilinx4000",
            clock_pad= "P13"
};
const dim = 3;
const dw = 8;
const ddw = 16;

main( target=harp1,
       port (in) datos : dw,
       port (out) R0 : ddw) {

       RAM int A1[dim] : dw;
       RAM int A2[dim] : dw;
       RAM int A3[dim] : dw;
       int i,j,k:2;
       int aux:ddw;

       int B0= -1; int B1= 1; int B2= 1; int B3= -1;
 int B4= 1;int B5= -1; int B6=1; int B7=-1;int B8=1;
//Ingreso los valores de la matriz A(3x3)
while (i.<.3)
            { par { datos ? A1[i];
                    i=i+1;}}
while (j.<.3)
            { par { datos ? A2[j];
                    j=j+1;}}
while (k.<.3)
            { par { datos ? A3[k];
                    k=k+1;}}

// realizo las operaciones con acumulador
aux = (A1[0]*B0) + (A2[0]*B3) + (A3[0]*B6);
aux = aux + (A1[1]*B1) + (A2[1]*B4) + (A3[1]*B7);
aux = aux + (A1[2]*B2) + (A2[2]*B5) + (A3[2]*B8);
R0 ! aux;
}
```

## Ejemplo 1, versión 4 para HCC

```
const spec harp1 ={
        fpga_type= "Xilinx4000",
        clock_pad= "P13"
};
const dw = 8;
const ddw = 16;

main( target=harp1,
      port (in) datos = puerto1 : dw,
      port (out) R0 = puerto2 : ddw)
{
      int B0= -1; int B1= 1; int B2= 1; int B3= -1; int B4= 1;
      int B5= -1; int B6= 1; int B7= -1; int B8= 1;
      int A0,A1,A2,A3,A4,A5,A6,A7,A8:dw;
      int result:ddw;
      //Ingreso los valores de la matriz A(3x3)

      datos ? A0;  datos ? A1;  datos ? A2;
      datos ? A3;  datos ? A4;  datos ? A5;
      datos ? A6;  datos ? A7;  datos ? A8;
      /* otbencion del resultado utilizando una variable
      auxiliar */
      result = (A0*B0) + (A1*B1) + (A2*B2) + (A3*B3) +
      (A4*B4) + (A5*B5) + (A6*B6) + (A7*B7) + (A8*B8);
      R0 ! result;
}
```

## Ejemplo 1, versión 5 para HCC

```
const spec ArcPci = {
        fpga_type= "Xilinx4000",
        clock_pad= "P13"  };
/* definicion de pines para bus de datos, direcciones, ce,
write, read */
const spec APmem={
    addr = {"mem_Addr<0>", "mem_Addr<1>",
            "mem_Addr<2>","mem_Addr<3>",
            "mem_Addr<4>","mem_Addr<5>"},
    data = "mem_Data<0>","mem_Data<1>",
           "mem_Data<2>", "mem_Data<3>",
           "mem_Data<4>","mem_Data<5>",
           "mem_Data<6>","mem_Data<7>"},
    ce   = "mem_ce",
    wb   = "mem_wb",
    en   = "mem_en"    };

const dw = 8;
const ddw = 16;
const tam_ram=2<<5;

void main (   target=ArcPci,
              eram mem[tam_ram]=APmem: dw){

      int R0          :ddw;
      int A0,A1,A2,A3,A4,A5,A6,A7,A8:dw;
      // cargo los valores de la matriz B cte
      int B0=-1; int B1=1; int B2=1; int B3=-1; int B4=1;
      int B5=-1; int B6=1; int B7=-1; int B8=1;

      / * ingreso desde memoria externa los valores
      de la matriz A */
      A0=mem[ 0 ];  //lectura de la RAM
      A1=mem[ 1 ];
      A2=mem[ 2 ];
      A3=mem[ 3 ];
      A4=mem[ 4 ];
      A5=mem[ 5 ];
      A6=mem[ 6 ];
      A7=mem[ 7 ];
      A8=mem[ 8 ];

      // calculo del promedio
      R0 = (A0*B0)+(A1*B1)+(A2*B2)+(A3*B3) +
           (A4*B4)+(A5*B5)+(A6*B6)+(A7*B7)+(A8*B8);
      // cargo en mem. externa el resultado del promedio
      mem[ 9 ]  = (R0\\dw);  // guardo bits mas signif.
      mem[ 10 ] = (R0<-dw); // guardo bits menos signif.
}
```

## Ejemplo 2, versión 1 para HCC

```
const spec harp1 ={ fpga_type= "Xilinx4000",
                    clock_pad= "P13"    };
const dim =  3;
const dw  =  8; const ddw = 16;

main(     target=harp1,
          port (in) datos : dw,
          port (out) R0 : ddw)
{
          ram int A[(dim*dim)] : dw;
          ram int B[(dim*dim)] : dw;
          int result,salida:ddw;
          int multa,multb:dw;
          int i : 4;
    /* definicion de la expresion que calcula la multiplicion
    de dos elementos y lo vuelca al acumulador */
    void multiplico ( ){
          result=multa*multb;
          salida= salida + result; }
    // comienza el main
    B[0] = -1;B[1] = 1;B[2] = 1; B[3] = -1; B[4] = 1;
    B[5] = -1;B[6] = 1;B[7] = -1; B[8] = 1;

    //Ingreso los valores de la matriz A (3x3)
    while (i.<.9)
              { par { datos ? A[i]; i = i+1;} }
    // loop que calucula el promedio
    salida,i = 0,0;    // salida = 0 ; i=0;

    while (i.<.9) {
              par { multa,multb=A[i],B[i]; i = i+1; }
              multiplico( );
    }
    R0 ! salida;
}
```

## Ejemplo 2, versión 2 para HCC

```
const spec harp1 ={ fpga_type= "Xilinx4000",
                    clock_pad= "P13"    };
const dim =  3;
const dw  =  8; const ddw = 16;

main(       target=harp1,
            port (in) datos : dw,
            port (out) R0 : ddw)
{
            ram int A[(dim*dim)] : dw;
            ram int B[(dim*dim)] : dw;
            int salida:ddw;
            int multa,multb:dw;
            int i : 4;
    /* definicion de la expresion que calcula la multiplicion
    de dos elementos y lo vuelca al acumulador */
void multiplico ( )
          { salida= salida + multa*multb; }
```

```
 // comienza el main
B[0] = -1;B[1] = 1;B[2] = 1; B[3] = -1; B[4] = 1;
B[5] = -1;B[6] = 1;B[7] = -1; B[8] = 1;

//Ingreso los valores de la matriz A (3x3)
while (i.<.9)
            { par { datos ? A[i]; i = i+1;} }
// loop que calucula el promedio
salida,i = 0,0;    // salida = 0 ; i=0;

while (i.<.9) {
            par { multa,multb=A[i],B[i]; i = i+1; }
            multiplico( );
}
R0 ! salida;
}
```

## Ejemplo 2, versión 3 para HCC

```
const spec harp1 ={ fpga_type= "Xilinx4000",
                    clock_pad= "P13"    };
const dim =  3;
const dw  =  8; const ddw = 16;

main(       target=harp1,
            port (in) datos : dw,
            port (out) R0 : ddw)
{
            ram int A[(dim*dim)] : dw;
            ram int B[(dim*dim)] : dw;
            int salida:ddw;
            int multa,multb:dw;
            int i : 4;
    /* definicion de la expresion que calcula la multiplicion
    de dos elementos y lo vuelca al acumulador */
    int multiplico () = salida + multa*multb;
    // comienza el main
```

```
B[0] = -1;B[1] = 1;B[2] = 1; B[3] = -1; B[4] = 1;
B[5] = -1;B[6] = 1;B[7] = -1; B[8] = 1;

//Ingreso los valores de la matriz A (3x3)
 while (i.<.9) {
       par{multa,multb=A[i],B[i];i=i+1;}
       salida = multiplico();
}// loop que calucula el promedio
salida,i = 0,0;    // salida = 0 ; i=0;

while (i.<.9) {
            par { multa,multb=A[i],B[i]; i = i+1; }
            multiplico( );
}
R0 ! salida;
}
```

## Operaciones con matrices: multiplicación simple

## Ejemplo1 para HCC

```
const spec harp1 = { fpga_type = "Xilinx4000",
                     clock_pad = "P160"};
const dim = 3;
const dw = 5;
const ddw = 10;

main( target=harp1,
      port (in) datos : dw,
      port (out) R0 : ddw) {
      /* defino memoria interna (tipo RAM) de 9 bytes y 8
      bit de ancho */
      ram int A[(dim*dim)] : dw;
      ram int B[(dim*dim)] : dw;
      int result:ddw;
      int inc:4;
      //Ingreso los valores de la matriz A (3x3)
       while (inc.<.9)
            {par { datos ? A[inc];inc=inc+1;}}
      //Ingreso los valores de la matriz B (3x3)
      inc=0;
      while (inc.<.9)
     {par { datos ? B[inc];inc=inc+1;}}
```

```
/* calculo cada elemento de la matriz utilizando un
acumulardor */
result =(A[0]*B[0]);result = result + (A[1]*B[3]);
result = result + (A[2]*B[6]);
R0 ! result;      // escribo del pueto R0

result =(A[0]*B[1]);    result = result + (A[1]*B[4]);
result = result + (A[2]*B[7]);   R0 ! result;
result =(A[0]*B[2]);    result = result + (A[1]*B[5]);
result = result + (A[2]*B[8]);   R0 ! result;
result =(A[3]*B[0]);    result = result + (A[4]*B[3]);
result = result + (A[5]*B[6]);   R0 ! result;
result =(A[3]*B[1]);    result = result + (A[4]*B[4]);
result = result + (A[5]*B[7]);   R0 ! result;
result =(A[3]*B[2]);    result = result + (A[4]*B[5]);
result = result + (A[5]*B[8]);   R0 ! result;
result =(A[6]*B[0]);    result = result + (A[7]*B[3]);
result = result + (A[8]*B[6]);   R0 ! result;
result =(A[6]*B[1]);    result = result + (A[7]*B[4]);
result = result + (A[8]*B[7]);   R0 ! result;
result =(A[6]*B[2]);    result = result + (A[7]*B[5]);
result = result + (A[8]*B[8]);   R0 ! result;
}
```

## Ejemplo 2 para HCC

```
const spec harp1 = {fpga_type = "Xilinx4000",
                    clock_pad = "P160"};

const dw = 5;
const ddw = 10;

main(     target=harp1,
          port (in) datos : dw,
          port (out) salida : ddw){

          int A0,A1,A2,A3,A4,A5,A6,A7,A8:dw;
          int B0,B1,B2,B3,B4,B5,B6,B7,B8:dw;
          int R0,R1,R2,R3,R4,R5,R6,R7,R8:ddw;

          // Ingreso los valores de la matriz A(3x3)
          datos ? A0;   datos ? A1;   datos ? A2;
          datos ? A3;   datos ? A4;   datos ? A5;
          datos ? A6;   datos ? A7;   datos ? A8;
          // Ingreso los valores de la matriz B (3x3)
          datos ? B0;   datos ? B1;   datos ? B2;
          datos ? B3;   datos ? B4;   datos ? B5;
          datos ? B6;   datos ? B7;   datos ? B8;
          /* paralelizo operaciones */
          par{
               // calculo de cada elemento de la matriz
               R0=(A0*B0)+(A1*B3)+(A2*B6);
               R1=(A0*B1)+(A1*B4)+(A2*B7);
               R2=(A0*B2)+(A1*B5)+(A2*B8);
               R3=(A3*B0)+(A4*B3)+(A5*B6);
               R4=(A3*B1)+(A4*B4)+(A5*B7);
               R5=(A3*B2)+(A4*B5)+(A5*B8);
               R6=(A6*B0)+(A7*B3)+(A8*B6);
               R7=(A6*B1)+(A7*B4)+(A8*B7);
               R8=(A6*B2)+(A7*B5)+(A8*B8);
          }
          //escritura del puerto R0
          salida ! R0;   salida ! R1;   salida ! R2;
          salida ! R3;   salida ! R4;   salida ! R5;
          salida ! R6;   salida ! R7;   salida ! R8;
}
```

## Ejemplo 3 para HCC

```
const spec harp1 = {fpga_type = "Xilinx4000",
                    clock_pad = "P160"};

const dw = 5;
const ddw = 10;

main( target=harp1,
      port (in) datos : dw,
      port (out) salida : ddw)
{
  int A0,A1,A2,A3,A4,A5,A6,A7,A8:dw;
  int B0,B1,B2,B3,B4,B5,B6,B7,B8:dw;
  int A_11,B_11,A_22,B_22,A_33,B_33:dw;
  int acum:ddw;
  // expresion que calcula cada elemento de la matriz
  void elemento_matriz(){
          acum = (A_11*B_11+A_22*B_22+A_33*B_33);
          salida ! acum;  //escirtura del puerto salida
  }
  //Ingreso los valores de la matriz A(3x3)
  datos ? A0;
  ... /* el resto de los datos se ingresan igual que antes */
  datos ? B8;

  // Calculo de los elementos de la matriz
  A_11,B_11,A_22,B_22,A_33,B_33=A0,B0,A1,B3,A2,B6;
  par{
     elemento_matriz();   // invoco el procedimiento
     B_11,B_22,B_33=B1,B4,B7; /* en paralelo cargo
                                 nuevos valores */
  }
  par{
     elemento_matriz();
     B_11,B_22,B_33=B2,B5,B8;
  }
  par{
     elemento_matriz();

  A_11,B_11,A_22,B_22,A_33,B_33=A3,B0,A4,B3,A5,B6;
  }
  par{
     elemento_matriz();
     B_11,B_22,B_33=B1,B4,B7;
  }
  par{
     elemento_matriz();
     B_11,B_22,B_33=B2,B5,B8;
  }
  par{
     elemento_matriz();

  A_11,B_11,A_22,B_22,A_33,B_33=A6,B0,A7,B3,A8,B6;
  }
  par{
     elemento_matriz();
     B_11,B_22,B_33=B1,B4,B7;
  }
  par{
     elemento_matriz();
     B_11,B_22,B_33=B2,B5,B8;
  }
  elemento_matriz();
}
```

# F.4 Fuentes de la aplicación final (capítulo 6)

## Filtro Original

```
/*  Modificado por Daniel Gomez para proyecto de grado con Handel C  02/2003

  Juan P. Oliver
  El programa hace la convolucion discreta de una imagen PGM binaria con una  matriz entera que se lee de un archivo ascii.

  parte del codigo: Y. Casals, L. Betarte, G. Errandonea, R. Grompone, mayo 2002
  parte del codigo: A. Gomez, A. Pardo */

#include <stdio.h>
#include <sys/types.h>
#include <string.h>
#include <ctype.h>
#include "pgmio.h"
#include <time.h>

#define __DEBUGGING

/* parametros */
char *arch_filtro = NULL;
char *arch_im_in = NULL;
char *arch_im_out = NULL;

/* matriz con filtro a usar */
int matriz_fil;  /* cantidad filas */
int matriz_col;  /* cantidad columnas */
int *matriz;
/*  la siguiente macro se usa para hacer mas leible el codigo
  al darle apariencia de matriz a un array unidimensional de enteros */
#define mat(i,j) (*(matriz+matriz_col*(i)+(j)))  /* i fila, j
                    columna, notacion estandar de matrices */

/* imagenes */
int ancho;
int alto;
unsigned char** imagen_entrada;
unsigned char** imagen_salida;

/* manejo de errores */
int hubo_error = 0;

/* declaracion de funciones */
int  leer_parametros( int, char** );
void leer_filtro();
void leer_imagen( unsigned char** I, char* arch_im_in,
                                        int* rows, int* cols );
void filtrar( int, int );
void error( char * mesg );
void uso();
unsigned char** allocate_image( int rows, int cols );

main(int argc, char** argv)
{
  int    i,j, n, size;
  int    status;
  clock_t tini,tfin;
  leer_parametros(argc,argv);
  leer_filtro();
  /*leer imagen*/
  imagen_entrada = ReadPGM(arch_im_in, &alto, &ancho);
  if( imagen_entrada == NULL )
              error("en la lectura de la imagen de entrada.\n");
  imagen_salida = allocate_image( alto, ancho);
  //filtrar( (n-1)*pixels_por_proceso, n*pixels_por_proceso );
  //filtrar( (n-1)*pixels_por_proceso, ancho*alto );
  printf("Comenzando filtrado de imagen...\n");
  tini=clock();
  filtrar( 0, alto*ancho-1 );
  tfin=clock();
  printf("Fin del filtrado. \n");
  printf(" Inicial %d  Final %d   Diferencia %d\n",
                                    tini,tfin, tfin-tini);
  // escribimos el archivo de salida
  WritePGM(arch_im_out, imagen_salida, alto, ancho);
  exit(0);
}

/*****************************************************/
int leer_parametros( int argc, char** argv )
{
  int i;

  for(i=1; i<argc; ++i)
    {
      if( strcmp(argv[i],"-i")==0 )
        {
          if( ++i >= argc ) uso(); /* aumenta i para apuntar al
                                      dato del parametro */
          arch_im_in = argv[i];
        }
      else if( strcmp(argv[i],"-o")==0 )
        {
          if( ++i >= argc ) uso(); /* aumenta i para apuntar al
                                      dato del parametro */
          arch_im_out = argv[i];
        }
      else if( strcmp(argv[i],"-f")==0 )
        {
          if( ++i >= argc ) uso(); /* aumenta i para apuntar al
                                      dato del parametro */
          arch_filtro = argv[i];
        }
      else
        uso();
    }

  if( arch_filtro==NULL || arch_im_in == NULL ||
                                    arch_im_out == NULL <= 0 )
    uso();
}

/*****************************************************/
void leer_filtro()
{
  FILE *filtro;
  int n, c, i, j;
  int aux_col;
  char buf[100];
  char *tok;

  if( (filtro=fopen(arch_filtro,"r")) == NULL )
              error("no puedo abrir el archivo con el filtro.\n");

  aux_col = 0;
  matriz_fil = 0;
```

```
  /* lee la primer fila para contar la cantidad de columnas */
  if( fgets(buf, 100, filtro) == NULL )
                                  error("el filtro no tiene filas.\n");
  for (tok = strtok(buf, " ,"); tok; tok=strtok(0, " ,")){
        aux_col = aux_col +1;
        printf("tok = `%s'\n", tok);
  }
  if( aux_col == 0 ) error("el filtro no tiene columnas.\n");
  matriz_col = aux_col;

  /* lee el resto de las filas para contarlas y chequear el
     formato */
  while( fgets(buf, 100, filtro) != NULL )
    {
      if( aux_col != matriz_col ) error("las filas del filtro no
                    tienen la misma cantidad de elementos.\n");
      ++matriz_fil;
      aux_col = 0;
      for (tok = strtok(buf, " ,"); tok; tok=strtok(0, " ,")){
        aux_col = aux_col +1;
      }
    }
  if( aux_col != 0 )
    {
      if( aux_col != matriz_col ) error("las filas del filtro no
                    tienen la misma cantidad de elementos.\n");
      ++matriz_fil;
    }

  /* pedido de memoria para guardar la matriz de
coeficientes */
  matriz = (int*)calloc(matriz_col*matriz_fil, sizeof(int));
  if( matriz == NULL ) error("no pude obtener memoria para
guardar el filtro.\n");

  // se vuelve al archivo al principio para la segunda lectura
  rewind(filtro);

  /* lectura del la matriz, ya se sabe el tama~o y que el
     formato es correcto */
  for(i=0; i<matriz_fil; ++i){
    for(j=0; j<matriz_col; ++j){
      n = fscanf(filtro, "%d", &mat(i,j));
      if(n!=1) error("error en la lectura de la matriz");
    }
  }
  if( fclose(filtro) == EOF )
                error("no puedo cerrar el archivo del filtro.\n");

#ifdef __DEBUGGING
  printf("Matriz de coeficientes leida:\n");
  for(i=0; i<matriz_fil; ++i)
    {
      for(j=0; j<matriz_col; ++j)
        printf("%d ",mat(i,j));
      printf("\n");
    }
#endif

}
/*****************************************/
unsigned char** allocate_image( rows, cols )
{
  unsigned char** J;
  int i;

  /* Allocate space for image data */
  J= (unsigned char**)malloc((rows)*sizeof(unsigned char*));
  if (J==NULL){fprintf(stderr,
                "Allocation Problems (ReadPGM)\n");exit(1);}

  for(i=0;i<(rows);i++)
    {
      J[i] = (unsigned char*)malloc((cols)*sizeof(
                                          unsigned char));
      if (J[i]==NULL){fprintf(stderr,
                "Allocation Problems (ReadPGM)\n");exit(1);}
    }
  return(J);
}

/*
  filtra los pixels desde "desde" hasta "hasta-1"
  donde estan numerados desde 0 a alto*ancho-1
  el cero es (0,0) y el alto*ancho-1 es (ancho,alto)
 */
void filtrar( int desde, int hasta )
{
  int x,y,x_aux,y_aux;
  int i,j;
  int sum,r,g,b;
  int pixel;

  for(pixel=desde; pixel<hasta; ++pixel)
    {
      x = pixel % ancho;
      y = pixel / ancho;

      sum = 0;
      r = 0;
      /* i fila de la matriz, corresponde a coordenada y de
         imagen, j columnas de la matriz, corresponde a
coordenada x de la imagen */
      for(i=0; i<matriz_fil; ++i)
        {
          y_aux = y - (matriz_fil/2) + i;
          if( y_aux<0 || y_aux>=alto ) continue;
          for(j=0; j<matriz_col; ++j)
            {
              x_aux = x - (matriz_col/2) + j;
              if( x_aux<0 || x_aux>=ancho ) continue;
              sum = sum + mat(i,j);
              r = r + mat(i,j) * imagen_entrada[y_aux][x_aux];
            }
        }
      if( sum != 0 )
        {
          r = r / sum;
        }
      imagen_salida[y][x] = r;
    }
}

/*************************************************/
void error( char * mesg )
{
  if( hubo_error == 0 )
    {
      hubo_error = 1;
      fprintf( stderr, "Error: %s", mesg );
      printf( "Error: %s", mesg );
      exit(1);
    }
  else
    {
      fprintf( stderr,
           "Error al tratar de cerrar correctamente: %s", mesg );
      printf(
           "Error al tratar de cerrar correctamente: %s", mesg );
      exit(1);
    }
}
/*************************************************/
void uso()
{
  error("parametros incorrectos\nmodo de uso:
                        lab_1 -i <imagen> -o <salida> -f <filtro> \n");
}
```

## Pgmio.c

```
/* ---------------------------------------- */
/* PGMio functions                          */
/* Daniel Gomez 7/2/2003                    */
/* correcciones en lectura de imagen        */
/* Alvaro Gomez, Juan P. Oliver 27/9/2002   */
/* correcciones DOS file format             */
/* Alvaro Pardo 12/4/2000                   */
/* ---------------------------------------- */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "pgmio.h"
#include <fcntl.h>
/* ---------------------------------------- */
unsigned char** ReadPGM(char* input_file, int* rows,
                                                int* cols)
{
  FILE* fid;
  int fi;
  char PGMid[2];
  char line[256];
  int i,j;
  unsigned char* tmp;
  unsigned char** I;
  off_t pos;

  fid = fopen(input_file,"r");

  /* Read PGM id */
  fgets(line,254,fid);
  if( line[1] != '5')
     printf("formato de archivo incorrecto,
                                  tiene que ser PGM binario\n");
  /* Read extra information (comments) */
  /* if no comments this line contain the image dimensions */
  fgets(line,254,fid);
  if(line[0] == '#')
    /* Read Image dimensions */
     fgets(line,254,fid);
  sscanf(line,"%i %i", cols, rows);
  /* Read Max Grey Level */
  fgets(line,254,fid);

  /* Allocate space for image data */
  I = (unsigned char**)malloc((*rows)*sizeof(
                                          unsigned char*));
  tmp = malloc(1);
  if (I==NULL){fprintf(stderr,
              "Allocation Problems (ReadPGM)\n");exit(1);}

  for(i=0;i<(*rows);i++)
    {
     I[i] = (unsigned char*)malloc((*cols)*sizeof(
                                          unsigned char));
     if (I[i]==NULL){fprintf(stderr,
              "Allocation Problems (ReadPGM)\n");exit(1);}
    }
  pos = ftell(fid); // guardo posicion en el archivo
  fclose(fid);
  // abro archivo en formato binario
  fi = open(input_file,O_RDONLY | O_BINARY);
  lseek (fi,pos,SEEK_SET);  // posiciono donde quede antes
  /* Get Image pixels */
  for(i=0;i<*rows;i++)
    for(j=0;j<*cols;j++){
//      I[i][j]=(int)fgetc(fid);
      read(fi,tmp,1);
      I[i][j] = *tmp;
//      printf(" (%d,%d) = %X \n",i,j,I[i][j]);
//      if (I[i][j] == 0x1a) getch();
//      if (feof(fid)!=0){ printf(" Fin de archivo"); clearerr(fid);}
//      if (ferror(fid) != 0) printf("Ocurrio un error");
     }
//  fclose(fid);
  close(fi);
  return(I);
}
 /* ---------------------------------------- */
void WritePGM(char* output_file, unsigned char** I,
                                              int rows, int cols)
{
  FILE* fid;
  int i,j;
  unsigned char maxcol;

  /* Compute the max grey level */
  maxcol = I[0][0];
  for(i=0;i<rows;i++)
    for(j=0;j<cols;j++)
      if(I[i][j]>maxcol)
              maxcol = I[i][j];

  fid = fopen(output_file,"wb");
  if(fid == NULL){fprintf(stderr,"Imposible to open %s\t
                              (WritePGM)\n",output_file);exit(1);}
  /* Write PGM id */
  fprintf(fid,"P5\n");
  /* Write the image data */
  fprintf(fid,"%i %i\n",cols,rows);
  fprintf(fid,"%i\n",maxcol);
  for(i=0;i<rows;i++)
    for(j=0;j<cols;j++)
      fprintf(fid,"%c",I[i][j]);

  fclose(fid);
}
```

## Pgmio.h

```
#ifndef PGMio_h
#define PGMio_h
/*-----------------------------------------------*/
/* Alvaro Pardo. 12/4/00                         */
/* functions to read and write PGM files         */
/*-----------------------------------------------*/

unsigned char** ReadPGM(char* input_file, int* rows,
                                                int* cols);
void WritePGM(char* output_file, unsigned char** I,
                                          int rows, int cols);

#endif
```

## Programa en C que trabaja con el filtro de la placa

```
/*  Daniel Gomez para proyecto de grado con Handel C
  02/2003
  Basado en programa original de Juan P. Oliver
  El programa hace la convolucion discreta de una imagen
PGM binaria con una  matriz entera que se lee de un
archivo ascii.

  parte del codigo: Y. Casals, L. Betarte, G. Errandonea, R.
Grompone, mayo 2002
  parte del codigo: A. Gomez, A. Pardo*/

#include <stdio.h>
#include <sys/types.h>
#include <string.h>
#include <ctype.h>
#include "pgmio.h"
#include <dpmi.h>
#include <sys/farptr.h>
#include <sys/movedata.h>
#include <sys/segments.h>
#include <time.h>

// defino constantes para la placa ARCPCI
#define physical_address_vga_neckar 0xe0000000UL
#define physical_address_base0 0xF0C10000UL
#define physical_address_base1 0xf0000000UL
#define physical_address_base2 0xf0400000UL
#define physical_address_size_base1 1024*1024*4UL
// 4M
#define     physical_address_size_base2     1024*1024*4UL
#define physical_address_size_base0 16UL    // 16
#define largo 350000

unsigned long buffer[largo];

#define __DEBUGGING

/* parametros */
char *arch_filtro = NULL;
char *arch_im_in = NULL;
char *arch_im_out = NULL;

/* matriz con filtro a usar */
int matriz_fil;  /* cantidad filas */
int matriz_col;  /* cantidad columnas */
int *matriz;
/*  la siguiente macro se usa para hacer mas leible el codigo
 al darle apariencia de matriz a un array unidimensional de
enteros */
#define mat(i,j) (*(matriz+matriz_col*(i)+(j)))  /* i fila, j
                    columna, notacion estandar de matrices */
/* imagenes */
int ancho;
int alto;
unsigned char** imagen_entrada;
unsigned char** imagen_salida;

/* manejo de errores */
int hubo_error = 0;

/* declaracion de funciones */
int  leer_parametros( int, char** );
void leer_filtro();
void leer_imagen( unsigned char** I, char* arch_im_in,
                                        int* rows, int* cols );
void error( char * mesg );
void uso();
unsigned char** allocate_image( int rows, int cols );

main(int argc, char** argv)
{
  int    i,j,x,y,pixel, n, size;
  int    status;
  uclock_t tini,tfin;
  __dpmi_meminfo mi2;
  int selector2;

    //configuro dpmi para usar memoria de la placa ARCPCI
    __dpmi_meminfo mi1;
    int selector1;
     printf("Comienza set de DPMI para banco 1.\n");

    /* Map the physical device address to linear memory.  */
    mi1.address = physical_address_base1;
    mi1.size    = physical_address_size_base1;
    if( __dpmi_physical_address_mapping (&mi1) == -1 )
       printf ("Error en:
                __dpmi_physical_address_mapping base1\n");
    /* Now mi.address holds the linear address.  */
    /* Allocate an LDT descriptor and set it up to span the
       entire device on-board memory.  */
    if((selector1= __dpmi_allocate_ldt_descriptors (1))== -1)
       printf ("Error en: __dpmi_allocate_ldt_descriptor\n");
    if( __dpmi_set_segment_base_address (selector1,
                                          mi1.address) == -1 )
       printf ("Error en:
                      __dpmi_set_segment_base_address\n");
    if( __dpmi_set_segment_limit (selector1, mi1.size - 1)
                                                    == -1)
       printf ("Error en: __dpmi_set_segment_limit\n");
     printf("Sin errores! de __dpmi\n");

// para banco 2
     printf("Comienza set de DPMI para banco 2.\n");

    /* Map the physical device address to linear memory.  */
    mi2.address = physical_address_base2;
    mi2.size    = physical_address_size_base2;
    if( __dpmi_physical_address_mapping (&mi2) == -1 )
       printf ("Error en:
                __dpmi_physical_address_mapping base1\n");
    /* Now mi.address holds the linear address.  */
    /* Allocate an LDT descriptor and set it up to span the
       entire  device on-board memory.  */
    if((selector2= __dpmi_allocate_ldt_descriptors (1))== -1)
       printf ("Error en: __dpmi_allocate_ldt_descriptor\n");
    if( __dpmi_set_segment_base_address (selector2,
                                          mi2.address) == -1 )
       printf ("Error en:
                      __dpmi_set_segment_base_address\n");
    if( __dpmi_set_segment_limit (selector2, mi2.size - 1)
                                                    == -1)
       printf ("Error en: __dpmi_set_segment_limit\n");
     printf("Sin errores! de __dpmi\n");
    // fin de la configuracion

    leer_parametros(argc,argv);
    leer_filtro();
    /*leer imagen*/
    printf("Leyendo imagen...  ");
    imagen_entrada=ReadPGM(arch_im_in, &alto, &ancho);
    if( imagen_entrada == NULL )
       error("en la lectura de la imagen de entrada.\n");
    else printf(" OK. \n");
    imagen_salida = allocate_image( alto, ancho);

/* Grabar en memoria de la ACRCPCI, en base1 y offset 0,
los coeficientes del filtro,  el filtro esta en
mat(filas,columnas) comienza de (0,0) y el maximo es
(matriz_fil,matriz_col)  copia datos del filtro al buffer1 */

// vacio buffer por las dudas...
```

```
    for (n=0;n<(matriz_fil*matriz_col/4);++n){
        buffer[n]=0;
    }
    // cargo matriz de coeficientes en el buffer tipo long
    n=0;
   for(i=0; i<matriz_fil; ++i){
        for(j=0; j<matriz_col; ++j){
//      printf("agrego %X \n",(unsigned char)(mat(i,j)));
            buffer[n/4]= buffer[n/4] | (((unsigned char)mat(i,j))
                                                << 8*(n % 4));
            n=n+1;
        }
    }

/* Grabar en memoria de la ARCPCI, en base1 y offset 22,
la imagen a filtrar,  la imagen esta en imagen_entrada[y][x]
donde "y" son las filas y "x" las columnas, ambas
comienzan de 0, sus filas maximas son "alto" y sus
columnas maximas son "ancho" */

// vacio buffer por las dudas...
    for (n=22;n<(ancho*alto/4);++n){
        buffer[n]=0;
    }
    n = 88;
    for(pixel=0; pixel<(ancho*alto); ++pixel) {
        x = pixel % ancho;
        y = pixel / ancho;
        buffer[n/4] = buffer[n/4] | (imagen_entrada[y][x]
                                                << 8*(n % 4));
        n=n+1;
    }

// copio datos en la RAM de la placa
_movedatal(_my_ds(),(unsigned) buffer,selector1, 0, (n/4));

    // lanzar filtrado
    // reseto la varable de control
    _farpokel(selector1, 0x1FFFFcUL,0);
    printf("Comenzando filtrado de imagen...\n");
    tini=uclock();
    // escribir o leer en la direccion 2fffff
    _farpeekl(selector1, 0x2FFFf0UL);

// esperar finalizacion leyendo direccion base1 offset ffffc, si
esta el dato 0, no arranco aun, dato cccccccc esta en
procesamiento y con ffffffff termino el filtrado
    while (_farpeekl(selector1, 0xFFFfcUL)==0) {
        printf ("Esperando que diseño aranque... \n");
    };
    if (_farpeekl(selector1, 0xFFFfcUL)== 0xcccccccc)
        printf ("El diseño arrancó. \n");
    while (_farpeekl(selector1, 0xFFFFcUL)==0xcccccccc) {};
    tfin=uclock();
    if (_farpeekl(selector1, 0xFFFFcUL)== 0xffffffff)
        printf ("El diseño ha finalizado. \n");
    else printf ("Problema, no se estan leyendo los datos
                                        correctos en 0xffffc. \n");
     // reseto la varable de control
    _farpokel(selector1, 0x1FFFFcUL,0);
    printf(" Inicial %lld  Final %lld   Diferencia %d\n",
                                                tini,tfin,tfin-tini);

    // leer datos de imagen filtrada de base2 offset 0
    printf("\nArmando imagen de salida ... ");
    _movedatal(selector2,0 ,_my_ds(),(unsigned) buffer,
                                                (ancho*alto/4));
    n = 0;
    for(pixel=0; pixel<(ancho*alto); ++pixel) {
        x = pixel % ancho;
        y = pixel / ancho;
        imagen_salida[y][x]=  (buffer[n/4] &(0xFF<< 8*(n% 4)))
                                                >> 8*(n % 4));;
        n=n+1;
    }
    // escribimos el archivo de salida
    writePGM(arch_im_out, imagen_salida, alto, ancho);
    printf("  OK. \n\n");
    exit(0);
}

/*****************************************************/
int leer_parametros( int argc, char** argv )
{
  int i;

  for(i=1; i<argc; ++i)
    {
      if( strcmp(argv[i],"-i")==0 )
        {
          if( ++i >= argc ) uso(); /* aumenta i para apuntar al
                                      dato del parametro */
          arch_im_in = argv[i];
        }
      else if( strcmp(argv[i],"-o")==0 )
        {
          if( ++i >= argc ) uso(); /* aumenta i para apuntar al
                                      dato del parametro */
          arch_im_out = argv[i];
        }
      else if( strcmp(argv[i],"-f")==0 )
        {
          if( ++i >= argc ) uso(); /* aumenta i para apuntar al
                                      dato del parametro */
          arch_filtro = argv[i];
        }
      else
        uso();
    }

  if( arch_filtro==NULL || arch_im_in == NULL || arch_im_out
                                                    == NULL <= 0 )
    uso();
}

/*****************************************************/
void leer_filtro()
{
  FILE *filtro;
  int n, c, i, j;
  int aux_col;
  char buf[100];
  char *tok;

  if( (filtro=fopen(arch_filtro,"r")) == NULL )
    error("no puedo abrir el archivo con el filtro.\n");

  aux_col = 0;
  matriz_fil = 0;

  /* lee la primer fila para contar la cantidad de columnas */
  if( fgets(buf, 100, filtro) == NULL )
     error("el filtro no tiene filas.\n");
  for (tok = strtok(buf, " ,"); tok; tok=strtok(0, " ,")){
     aux_col = aux_col +1;
     printf("tok = `%s'\n", tok);
  }
  if( aux_col == 0 ) error("el filtro no tiene columnas.\n");
  matriz_col = aux_col;

  /* lee el resto de las filas para contarlas y chequear el
     formato */
  while( fgets(buf, 100, filtro) != NULL )
    {
     if( aux_col != matriz_col ) error("las filas del filtro no
                    tienen la misma cantidad de elementos.\n");
     ++matriz_fil;
     aux_col = 0;
     for (tok = strtok(buf, " ,"); tok; tok=strtok(0, " ,")){
       aux_col = aux_col +1;
```

```
    }
   }
  if( aux_col != 0 )
   {
    if( aux_col != matriz_col ) error("las filas del filtro no
                    tienen la misma cantidad de elementos.\n");
    ++matriz_fil;
   }

  /* pedido de memoria para guardar la matriz de
     coeficientes */
  matriz = (int*)calloc(matriz_col*matriz_fil, sizeof(int));
  if( matriz == NULL )
   error("no pude obtener memoria para guardar el filtro.\n");

  // se vuelve al archivo al principio para la segunda lectura
  rewind(filtro);

  /* lectura del la matriz, ya se sabe el tama~o y que el
     formato es correcto */
  for(i=0; i<matriz_fil; ++i){
   for(j=0; j<matriz_col; ++j){
    n = fscanf(filtro, "%d", &mat(i,j));
    if(n!=1) error("error en la lectura de la matriz");
   }
  }
  if( fclose(filtro) == EOF )
     error("no puedo cerrar el archivo del filtro.\n");

#ifdef __DEBUGGING
  printf("Matriz de coeficientes leida:\n");
  for(i=0; i<matriz_fil; ++i)
   {
    for(j=0; j<matriz_col; ++j)
     printf("%d ",mat(i,j));
    printf("\n");
   }
#endif

}
/*****************************************/
unsigned char** allocate_image( rows, cols )
{
  unsigned char** J;
  int i;

  /* Allocate space for image data */
  J =(unsigned char**)malloc((rows)*sizeof(unsigned char*));
  if (J==NULL){
     fprintf(stderr,"Allocation Problems (ReadPGM)\n)");
     exit(1);
  }

  for(i=0;i<(rows);i++)
    {
     J[i] = (unsigned char*)malloc((cols)*sizeof(
                                                  unsigned char));
     if (J[i]==NULL){
        fprintf(stderr,"Allocation Problems (ReadPGM)\n)");
        exit(1);
      }
    }
  return(J);
}
/*****************************************************/
void error( char * mesg )
{
  if( hubo_error == 0 )
   {
    hubo_error = 1;
    fprintf( stderr, "Error: %s", mesg );
    printf( "Error: %s", mesg );
    exit(1);
   }
  else
   {
    fprintf( stderr,
          "Error al tratar de cerrar correctamente: %s", mesg );
    printf(  "Error al tratar de cerrar correctamente: %s",
                                                        mesg );
    exit(1);
   }
}
/*****************************************************/
void uso()
{
  error("parametros incorrectos\nmodo de uso: lab_1 –i
                                  <imagen> -o <salida> -f <filtro> \n");
}
```

## Filtro en Handel C

```
// Diseño final
/* Filtra una imagen multiplicandola por una matriz
constante (3 x 3)
   Supongo que el número de filas es multiplo de 4
*/
#include "inc_fnl.h"

void main ( target=ArcPci,
   eram mem_dat[tam_ram_ext]=APmem: 32,
   eram mem_res[tam_ram_ext]=APmem2: 32,
   port (out) pido_bus1 : 1,
   port (in)  recibo_bus1: 1,
   port (out) pido_bus2 : 1,
   port (in)  recibo_bus2: 1,
   port (in)  inicio:1,
   port (out) ledn: 1,

   port (out) ram_coef_dir:ind_ram,
   port (in)  ram_coef_dot: 8,
   port (out) ram_coef_din: 8,
   port (out) ram_coef_we : 1
   )
{
   int   aux,aux1, pedido1, pedido2,   primera, prendido:1;
   int     suma_coef,  div8,  tmp,  tmp2:     8; /* 1) suma de
coeficientes del filtro 2)resultado de   la division 3)
coeficiente 4) pixel */
   int   sum16, mul16:16; /*  almacenan la suma (sum16) de
los productos parciales (mul16) de   pixel por coef */
   int     var32,  tmp32:32;  /*   en la primera se cargan los
datos de la imagen original   y   en la otra guardamos los
resultados temporales que se van a guardar   en la nueva
imagen */
   int   I: ind_ram; // indice a la ram de coeficientes
   int     cuenta,   tiempo1,  tiempo2,  tiempo3:  8;  /* se usan
para el  led */

   void division8(){ // calcula  div8 = sum16 / sum_coef
     int   ccnt0,ccnt1,ccnt2,ccnt3,ccnt4,ccnt5,ccnt6,ccnt7:1;
     int   ccnt8,ccnt9,ccnt10,ccnt11,ccnt12,ccnt13:1;
     int   d,z : 24; /* divisor y dividendo */
     int   s3,s4,s5,s6,s7:24;
```

```
    int   s8,s9,s10,s11,s12,s13,s14,s15:24;

    if (suma_coef.(1..7) .>. 0){ /* si suma_coef = 0 ó 1 no
hago la  división */
// agrego 8 ceros al dividendo y desplazo divisor  16 lugares
      z, d = 0 @ abs(sum16), abs(suma_coef) @ 0;
      /* comienzo a calcular de la posición 13, las pos. 14 y
      15 dan 0 porque es   división sin signo y ademas   en
      este  punto el divisor  es mayor que 1*/
      ccnt13 =((((z<<3)-d)<0) ? 0   : 1);
      s3=(z<<3)-((ccnt13==1) ? d : 0);
      ccnt12 =((((s3<<1)-d)<0) ? 0 : 1);
      s4=(s3<<1)-((ccnt12==1) ? d   : 0);
      ccnt11 =((((s4<<1)-d)<0) ? 0 : 1);
      s5=(s4<<1)-((ccnt11==1) ? d   : 0);
      ccnt10 =((((s5<<1)-d)<0) ? 0 : 1);
      s6=(s5<<1)-((ccnt10==1) ? d   : 0);
      ccnt9 =((((s6<<1)-d)<0) ? 0   : 1);
      s7=(s6<<1)-((ccnt9==1) ? d : 0);
      ccnt8 =((((s7<<1)-d)<0) ? 0   : 1);
      s8=(s7<<1)-((ccnt8==1) ? d : 0);
      ccnt7 =((((s8<<1)-d)<0) ? 0   : 1);
      s9=(s8<<1)-((ccnt7 ==1) ? d   : 0);
      ccnt6 =((((s9<<1)-d)<0) ? 0   : 1);
      s10=(s9<<1)-((ccnt6==1) ? d   : 0);
      ccnt5 =((((s10<<1)-d)<0) ? 0 : 1);
      s11=(s10<<1)-((ccnt5==1) ? d : 0);
      ccnt4 =((((s11<<1)-d)<0) ? 0 : 1);
      s12=(s11<<1)-((ccnt4==1) ? d : 0);
      ccnt3 =((((s12<<1)-d)<0) ? 0 : 1);
      s13=(s12<<1)-((ccnt3==1) ? d : 0);
      ccnt2 =((((s13<<1)-d)<0) ? 0 : 1);
      s14=(s13<<1)-((ccnt2==1) ? d : 0);
      ccnt1 =((((s14<<1)-d)<0) ? 0 : 1);
      s15=(s14<<1)-((ccnt1==1) ? d : 0);
      ccnt0 =((((s15<<1)-d)<0) ? 0 : 1);
      /* concateno los 8 resultados menos significativos y
      ajusto signo */
      div8= (((sum16.15 ^  suma_coef.7)==0) ? (ccnt7
            @ ccnt6 @  ccnt5 @  ccnt4 @  ccnt3 @  ccnt2
             @ ccnt1 @  ccnt0) : -(ccnt7 @ ccnt6 @ ccnt5
              @ ccnt4 @ ccnt3 @ ccnt2 @ ccnt1 @ ccnt0));
    }
    else div8= sum16.(0..7); // recorto a 8 bits
  }

  void multip8(){ /* calcula sum16 = tmp * tmp2 + sum16
(coeficiente *  pixel +  suma valores anteriores) */
    int   m41, m42: 14;
    int   m0, m1, m2, m3: 10;
    int   mr:8;

    par{  //calculo valor   absoluto
      if (tmp.7==1)  mr=   (~tmp)+1;
      else       mr=   tmp;
    }
    par{ // desplazamiento y 1er nivel de sumas
      m0=cond(mr.(0..1),0-> 0, 1->  0 @ tmp2, 2-> (0
               @ tmp2 @ cero1), 3-> ((cero2 @ tmp2.(1..7))
                 + (cero1 @ tmp2.(0..7))) @ tmp2.0);
      m1=cond(mr.(2..3),0-> 0, 1->  0 @ tmp2, 2-> (0
               @ tmp2 @ cero1), 3-> ((cero2 @ tmp2.(1..7))
                 + (cero1 @ tmp2.(0..7))) @ tmp2.0);
      m2=cond(mr.(4..5),0-> 0, 1->  0 @ tmp2, 2-> (0
               @ tmp2 @ cero1), 3-> ((cero2 @ tmp2.(1..7))
                 + (cero1 @ tmp2.(0..7))) @ tmp2.0);
      m3=cond(mr.(6..7),0-> 0, 1->  0 @ tmp2, 2-> (0
               @ tmp2 @ cero1), 3-> ((cero2 @ tmp2.(1..7))
                 + (cero1 @ tmp2.(0..7))) @ tmp2.0);
    }
    par{ // segundo   nivel de sumas
      m41 =((0 @ m0.(4..9))+ m2.(0..9)) @ m0.(0..3);
      m42 =((0 @ m1.(4..9))+ m3.(0..9)) @ m1.(0..3);
    }
    // último nivel de sumas y ajuste del signo
    mul16 = ((tmp.7==1) ? (~(((0 @ m41.(2..13))
          + m42.(0..13)) @ m41.(0..1))+1): (((0 @ m41.(2..13))
            + m42.(0..13)) @ m41.(0..1)));
    sum16 = mul16 + sum16;  // almaceno resultado
  }

  void filtrar_imagen(){
    int flag : 1;
    int  i,  m  :tam_ind_Icol; /*  indice  de  columnas  y
desplazamiento*/
    int j, k :tam_ind_Ifil; // indice de filas y desplazamiento
    int  tmp3  :  tam_ind_Ifil+1;  /*  var.  aux.    para  dividir  el
cálculo  de la dirección   de la ram en etapas */
    int  dir_ram2,  dir_ram,  dir_ram_ant,  tmp1  :
ind_ram_ext+1; //dir. de la ram y otras variables auxiliares
    //inicializo variables
    j,  primera,  i,tmp32=  Ffilas_div2,  1,
Fcolumnas_div2,0xffffffff;
    while (j .<. (Ifilas-Ffilas_div2)){ /* recorro filas de  imagen
de datos */
      while  (i  .<.  (Icolumnas-Fcolumnas_div2)){  /*  recorro
columnas de imagen de datos */
        par{ /*  pido bus, inicializo variables para el cálculo
de nuevo  pixel */
          pedido1, l, dir_ram_ant, sum16,  k =1, 0, 0, 0
                                        , -Ffilas_div2;
          /* flag indica que hay que escribir un nuevo dato
en ram 2 */
          flag = ((i.(0..1)==3 | (i==(Icolumnas
                              -Fcolumnas_div2-1))) ? 1 :   0);
        }
        while(k!=(Ffilas_div2+1)){  /*  recorro  filas  de
submatriz centrada en el   punto i,j  */
          // comienzo calculo de la dirección de la ram 1
          tmp3 = 0 @ (j + k);
          //  continuo calculo e inicializo m
          tmp1, m= (tmp3.*.(IcolW_div4)), -Fcolumnas_div2;
while(m!=(Fcolumnas_div2+1)){  /*  recorro    columnas  de
submatriz centrada en el punto i,j */
            dir_ram, dir_ram_ant=tmp1 + (0 @ ((i
                        + m).(2..(tam_ind_Icol-1)))), dir_ram;
            if ((dir_ram_ant!=dir_ram) | (primera==1)){
              par{ /* si no tengo el bus espero a recibirlo, si
ya lo tengo el par demora un solo ciclo   de reloj */
                {
                  recibo_bus1 ? aux;
                  while(aux){recibo_bus1 ? aux;};
                }
                dir_ram2 = Idir_base + dir_ram;  /* le agrego
offset  a direccion */
              }
              // leo dato en var32
              var32, primera=mem_dat[0 @ dir_ram2],0;
            };
            par{
              // elijo  el byte  que   me interesa
              case (i.(0..1)+   m.(0..1)){
                0: {tmp2 = var32.(0..7);}
                1: {tmp2 = var32.(8..15);}
                2: {tmp2 = var32.(16..23);}
                3: {tmp2 = var32.(24..31);}
              };
              ram_coef_dot ? tmp;  //  leo   coeficiente
              // incremento indice de coefs.   y de submatriz
              l,m=l+1, m+1;
            }
            par{
              multip8(); // calcula sum16= sum16+tmp* tmp2
              if (m==(Fcolumnas_div2 + 1))
                k=k+1;   // incremento indice de submatriz
            }
          };
        };
        par{
```

```
          division8();       /*  calcula  div8=  (sum16  /
suma_coef).(0..7) si suma_coef>=2
          si no div8 = sum16.(0..7) */
          pedido1  = 0; //  devuelvo bus 1
          pedido2  = flag;  // pido  bus   2 si es  necesario
        }
        if (flag == 1){   // escribo dato   en bus 2
          par   { // roto y concateno último dato a tmp32
            tmp32=cond((j.*.IcolW+(0 @ i)).(0..1),
                  0->  tmp32.(8..31) @ div8,
                  1->tmp32.(16..31) @ div8 @ tmp32.(0..7),
                  2->tmp32.(24..31) @ div8 @ tmp32.(0..15),
                  3->div8 @ tmp32.(0..23));
            recibo_bus2 ? aux;
          }
         // espero a recibir  el bus
          while(aux){recibo_bus2 ? aux;};


          // escribo dato   (tmp32)  en bus 2
          mem_res[0 @ ((j.*.IcolW_div4)+(0 @ i>>2))]
                                                  =  tmp32;
        }
        else //  roto y concateno dato a tmp32
          tmp32=cond((j.*.IcolW+(0 @ i)).(0..1),
                  0->tmp32.(8..31) @ div8,
                  1->tmp32.(16..31)  @ div8 @ tmp32.(0..7),
                  2->tmp32.(24..31)   @ div8 @ tmp32.(0..15),
                  3->div8 @ tmp32.(0..23));
        i, pedido2 = i+1, 0; /* incremento columna y
        devuelvo bus 2 si lo  había pedido */
      };
      // inicializo columna e incremento fila
      i, j= Fcolumnas_div2, j+1;
    };
    pedido1=1; // pido bus 1 para avisar que termino filtrado
    // espero a recibir  el bus
    recibo_bus1 ? aux;
    while(aux){recibo_bus1 ? aux;};
    /* escribo dato   de finalización   en la última dirección
       de la memoria */
    mem_dat[0x3ffff]=0xffffffff;
    pedido1=0;  // devuelvo bus   1
  }

  void configurar_filtro(){
    // Los elementos del filtro se ordenan por filas en la ram
    l, suma_coef, pedido1=0, 0,   1; /* inicializo vars. y pido
    bus 1 */
    // espero a recibir  bus   1
    recibo_bus1 ? aux;
    while(aux){recibo_bus1 ? aux;};
    /* escribo dato   de "en proceso"   en la última dirección
     de la memoria */
    mem_dat[0x3ffff]=0xccccccccc;
    while(l.<.Felementos){ // recorro la ram del filtro
      if (l.(0..1)==0) /*  si l es  multip.  de 4 leo nuevo dato
                        de ram 1 */
        var32=mem_dat[0   @ l.(2..(ind_ram-1))];
      case ((l).(0..1)){ /* según   ultimos  bits de  l selecciono
                          byte del dato */
        0: {tmp2 = var32.(0..7);}
        1: {tmp2 = var32.(8..15);}
        2: {tmp2 = var32.(16..23);}
        3: {tmp2 = var32.(24..31);}
      };
      par{ /* escribo  en la ram, acumulo suma_coef e
              incremento indice */
        ram_coef_din ! tmp2;
        ram_coef_we ! 1;
        suma_coef = suma_coef + tmp2;
        l =   l+1;
      }
    };
    pedido1= 0; // devuelvo bus   1
  }



// ----  Comienza programa principal

  par{
    while(1){
      par{ //  escrituras a puertos según variables
        pido_bus1 ! pedido1;
        pido_bus2 ! pedido2;
        ram_coef_dir ! l;
        ledn ! prendido;
      }
    };
    while(1){
      // espero señal   de comienzo
      inicio ? aux1;
      while(~aux1){
        par{
          inicio ? aux1;
          // reseteo los pedidos aunque deberían estar en 0
          pedido1  = 0;
          pedido2  = 0;
        }
      };
      configurar_filtro(); // cargo coefs. y calculo su  suma
      cuenta = 0; // inicializo variable para   el led
      par{
        filtrar_imagen();
        /* prendo led por ~1 seg. y luego lo apago por medio
         hasta que cuenta =  1+ abs(suma_coef) */
        while(cuenta !=   (1+abs(suma_coef))){
          prendido, cuenta, tiempo1=1, cuenta + 1, 0;
          while(tiempo1!=0xff){
            while(tiempo2!=0xff){
              while(tiempo3!=0xff){
                tiempo3=tiempo3+1;
              };
              tiempo2, tiempo3=tiempo2+1,   0;
            };
          tiempo1, tiempo2=tiempo1+1,   0;
          };
          prendido, tiempo1 =  0, 0;
          while(tiempo1!=0x7f){
            while(tiempo2!=0xff){
              while(tiempo3!=0xff){
                tiempo3=tiempo3+1;
              };
              tiempo2, tiempo3=tiempo2+1,   0;
            };
            tiempo1, tiempo2=tiempo1+1,   0;
          };

        };
      }
    };
  }
}
```

## **Inc_fnl.h** (include para el filtro en Handel C)

```
// anchos de palabra
const sw = 8;
const dw = 16;
const ddw = 32;

const Ifilas = 500;     // Número de filas de la imagen, debe ser múltiplo de 4
const Icolumnas = 500;  // Número de columnas de la imagen
const Ffilas = 3; // Número de filas del filtro
const Fcolumnas = 3; // Número de columnas del filtro
const Felementos = Ffilas * Fcolumnas;

const tam_ram_ext = 1<<18; // Tamaño de la memoria (o espacio ocupado)
const tam_ram_coef = log2(Ffilas * Fcolumnas)+1;

const Idir_base = 22;  // calculada a partir de un filtro de tamaño máximo 9 x 9
const ind_ram_ext = log2(((Ifilas*Icolumnas)>>2)+Idir_base)+1; // ancho de indices a ram
const ind_ram = log2((Ffilas * Fcolumnas))+1;

const tam_ind_Ifil = log2(Ifilas)+1;
const tam_ind_Icol = log2(Icolumnas)+1;
const Fcolumnas_div2 = Fcolumnas >> 1;
const Ffilas_div2 = Ffilas >> 1;
const IcolW = Icolumnas : tam_ind_Icol;
const IcolW_div4 = Icolumnas >> 2 : tam_ind_Icol-2;
const cero8 = 0:8;
const cero1 = 0:1;
const cero2 = 0:2;

const spec ArcPci = {
  fpga_type= "Xilinx4000",
  fpga_chip="xc4010xlpc84-3",
  clock_pad= "P13"
};
const spec APmem={
  addr = {"mem_dat_Addr<0>","mem_dat_Addr<1>","mem_dat_Addr<2>","mem_dat_Addr<3>",
        "mem_dat_Addr<4>","mem_dat_Addr<5>","mem_dat_Addr<6>","mem_dat_Addr<7>",
        "mem_dat_Addr<8>","mem_dat_Addr<9>","mem_dat_Addr<10>","mem_dat_Addr<11>",
        "mem_dat_Addr<12>","mem_dat_Addr<13>","mem_dat_Addr<14>","mem_dat_Addr<15>",
        "mem_dat_Addr<16>","mem_dat_Addr<17>"},
  data = {"mem_dat_Data<0>","mem_dat_Data<1>","mem_dat_Data<2>","mem_dat_Data<3>",
        "mem_dat_Data<4>","mem_dat_Data<5>","mem_dat_Data<6>","mem_dat_Data<7>",
        "mem_dat_Data<8>","mem_dat_Data<9>","mem_dat_Data<10>","mem_dat_Data<11>",
        "mem_dat_Data<12>","mem_dat_Data<13>","mem_dat_Data<14>","mem_dat_Data<15>",
        "mem_dat_Data<16>","mem_dat_Data<17>","mem_dat_Data<18>","mem_dat_Data<19>",
        "mem_dat_Data<20>","mem_dat_Data<21>","mem_dat_Data<22>","mem_dat_Data<23>",
        "mem_dat_Data<24>","mem_dat_Data<25>","mem_dat_Data<26>","mem_dat_Data<27>",
        "mem_dat_Data<28>","mem_dat_Data<29>","mem_dat_Data<30>","mem_dat_Data<31>"},
  ce   =  "mem_dat_ce",
  wb   =  "mem_dat_wb",
  en   =  "mem_dat_en"
};
const spec APmem2={
  addr = {"mem_res_Addr<0>", "mem_res_Addr<1>","mem_res_Addr<2>","mem_res_Addr<3>",
        "mem_res_Addr<4>","mem_res_Addr<5>","mem_res_Addr<6>","mem_res_Addr<7>",
        "mem_res_Addr<8>","mem_res_Addr<9>","mem_res_Addr<10>","mem_res_Addr<11>",
        "mem_res_Addr<12>","mem_res_Addr<13>","mem_res_Addr<14>","mem_res_Addr<15>",
        "mem_res_Addr<16>","mem_res_Addr<17>"},
  data = {"mem_res_Data<0>","mem_res_Data<1>","mem_res_Data<2>","mem_res_Data<3>",
        "mem_res_Data<4>","mem_res_Data<5>","mem_res_Data<6>","mem_res_Data<7>",
        "mem_res_Data<8>","mem_res_Data<9>","mem_res_Data<10>","mem_res_Data<11>",
        "mem_res_Data<12>","mem_res_Data<13>","mem_res_Data<14>","mem_res_Data<15>",
        "mem_res_Data<16>","mem_res_Data<17>","mem_res_Data<18>","mem_res_Data<19>",
        "mem_res_Data<20>","mem_res_Data<21>","mem_res_Data<22>","mem_res_Data<23>",
        "mem_res_Data<24>","mem_res_Data<25>","mem_res_Data<26>","mem_res_Data<27>",
        "mem_res_Data<28>","mem_res_Data<29>","mem_res_Data<30>","mem_res_Data<31>"},
  ce   =  "mem_res_ce",
  wb   =  "mem_res_wb",
  en   =  "mem_res_en"
};
```

## Interfaz VHDL

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;

entity final12ap is
  port (
  -- system signals
  system_half_clock : in std_logic;
  system_clock   : in std_logic;
  system_double_clock  : in std_logic;
  pld_ledn    : out std_logic;  -- external pull-up
  pld0_fast0n    : inout std_logic;   -- external pull-up
  pld0_fast1n    : inout std_logic;   -- external pull-up
  resetn        : in std_logic;   -- external pull-up DEV_CLRn
  -- bus one interface
  bus1_address      : inout std_logic_vector(19 downto 0);
  bus1_data      : inout std_logic_vector(31 downto 0);
  bus1_byte_enablen : inout std_logic_vector(4 downto 1);
  -- external pull-up
  bus1_write_enablen  : inout std_logic;   -- external pull-up
  bus1_output_enablen : inout std_logic;   -- external pull-up
  bus1_mion      : inout std_logic;   -- external pull-up
  bus1_fastn     : in std_logic;   -- external pull-up

  -- bus two interface
  bus2_address      : inout std_logic_vector(19 downto 0);
  bus2_data      : inout std_logic_vector(31 downto 0);
  bus2_byte_enablen : inout std_logic_vector(4 downto 1);
                -- external pull-up
  bus2_write_enablen   : inout std_logic;   -- external pull-up
  bus2_output_enablen : inout std_logic;   -- external pull-up
  bus2_mion      : inout std_logic;   -- external pull-up
  bus2_fastn     : in std_logic;   -- external pull-up
  -- cache-I/O bus interface
  cache_address     : inout std_logic_vector(19 downto 0);
  cache_data     : inout std_logic_vector(31 downto 0);
  cache_byte_enablen : inout std_logic_vector(4 downto 1);
              -- external pull-up
  cache_write_enablen  : inout std_logic; -- external pull-up
  cache_output_enablen : inout std_logic; -- external pull-up
  cache_mion     : inout std_logic_vector(2 downto 0);
              -- external pull-up
  cache_fastn    : in std_logic -- external pull-up
);
end final12ap;
architecture a1 of final12ap is
  signal GND, VCC, resetn_i, pld_ledn_i : std_logic;
  signal start, start1, start2, start3: std_logic;
  signal mem_dat_ce_buf,mem_dat_wb_buf : std_logic;
  signal mem_dat_en_buf,pido_bus1_buf : std_logic;
  signal bus1_fastn1, hab_tri1, hab_tri2: std_logic;
  signal mem_dat_Addr_buf, mem_res_Addr_buf:
                                  std_logic_vector(17 downto 0);
  signal mem_res_ce_buf,mem_res_wb_buf : std_logic;
  signal mem_res_en_buf: std_logic;
  signal pido_bus2_buf,bus2_fastn1: std_logic;

  component finalv12
  port (
    INPUT_CLOCK:in std_logic;
    globalReset: std_logic;
    inicio:in std_logic;
    mem_dat_Addr:out std_logic_vector(17 downto 0);
    mem_dat_Data:inout std_logic_vector(31 downto 0);
    mem_dat_ce:out std_logic;
    mem_dat_en:out std_logic;
    mem_dat_wb:out std_logic;
    mem_res_Addr:out std_logic_vector(17 downto 0);
    mem_res_Data:inout std_logic_vector(31 downto 0);
    mem_res_ce:out std_logic;
    mem_res_en:out std_logic;
    mem_res_wb:out std_logic;
    pido_bus1:out std_logic;
    pido_bus2:out std_logic;
    recibo_bus1:in std_logic;
    recibo_bus2:in std_logic;
    ledn: out std_logic
  );
  end component;

begin
  C0 : finalv12 port map ( input_clock => system_half_clock,
        globalReset => resetn_i, inicio => start3,
        mem_dat_Addr => mem_dat_Addr_buf,
        mem_dat_Data => bus1_data,
        mem_dat_ce => mem_dat_ce_buf,
        mem_dat_en => mem_dat_en_buf,
        mem_dat_wb => mem_dat_wb_buf,
        mem_res_Addr => mem_res_Addr_buf,
        mem_res_Data => bus2_data ,
        mem_res_ce => mem_res_ce_buf,
        mem_res_en => mem_res_en_buf,
        mem_res_wb => mem_res_wb_buf,
        pido_bus1 => pido_bus1_buf,
        recibo_bus1 => bus1_fastn,
        pido_bus2 => pido_bus2_buf,
        recibo_bus2 => bus2_fastn, ledn => pld_ledn_i
  );

  GND <= '0';
  VCC <= '1';
  resetn_i <= not resetn;

  process (pido_bus1_buf, bus1_fastn)
  begin
    if ((not pido_bus1_buf) or bus1_fastn)='0' then
      hab_tri1<='1';
    else
      hab_tri1<='0';
    end if;
  end process;

  process (pido_bus2_buf, bus2_fastn)
  begin
    if ((not pido_bus2_buf) or bus2_fastn)='0' then
      hab_tri2<='1';
    else
      hab_tri2<='0';
    end if;
  end process;

  process (system_clock,resetn)
  begin
    if (resetn='0') then
      start <= '0';
    else if rising_edge(system_clock) then
      if (bus1_address(19) and
                          not(bus1_output_enablen))='1' then
        start <= '1';
      else
        start <= '0';
      end if;
       end if;
    end if;
  end process;

 -- ensanchamos el pulso de start pues nuestra maquina
   corre con un reloj mas lento.
  process (system_clock)
  begin
    if rising_edge(system_clock) then
      start1 <= start;
      start2 <= start1;
    end if;
```

```
end process;

start3 <= start or start1 or start2;
-- system signals
pld_ledn <= not pld_ledn_i;
pld0_fast0n <= 'Z';
pld0_fast1n <= 'Z';
-- bus one interface
bus1_address(19 downto 0)<=
        ("00" &  mem_dat_Addr_buf)  when hab_tri1='1'
        else "ZZZZZZZZZZZZZZZZZZZZ";
bus1_byte_enablen(1)<=mem_dat_ce_buf
                                    when hab_tri1='1' else 'Z';
bus1_byte_enablen(2)<=mem_dat_ce_buf
                                    when hab_tri1='1' else 'Z';
bus1_byte_enablen(3)<=mem_dat_ce_buf
                                    when hab_tri1='1' else 'Z';
bus1_byte_enablen(4)<=mem_dat_ce_buf
                                    when hab_tri1='1' else 'Z';
bus1_output_enablen<=mem_dat_en_buf
                                    when hab_tri1='1' else 'Z';
bus1_write_enablen<=mem_dat_wb_buf
                                    when hab_tri1='1' else 'Z';
bus1_mion <= '0' when pido_bus1_buf='1' else 'Z';       --;
-- bus two interface
bus2_address(19 downto 0) <=
              ("00" & mem_res_Addr_buf) when hab_tri2='1'
              else "ZZZZZZZZZZZZZZZZZZZZ";
bus2_byte_enablen(1)<=mem_res_ce_buf
                                    when hab_tri2='1' else 'Z';
bus2_byte_enablen(2)<=mem_res_ce_buf
                                    when hab_tri2='1' else 'Z';
bus2_byte_enablen(3)<=mem_res_ce_buf
                                    when hab_tri2='1' else 'Z';
bus2_byte_enablen(4)<=mem_res_ce_buf
                                    when hab_tri2='1' else 'Z';
bus2_output_enablen<=mem_res_en_buf
                                    when hab_tri2='1' else 'Z';
bus2_write_enablen<=mem_res_wb_buf
                                    when hab_tri2='1' else 'Z';
bus2_mion <= '0' when pido_bus2_buf='1' else 'Z';
-- cache-I/O bus interface
cache_address(19 downto 0) <=
                "00000000000000000000" when GND='1'
                else "ZZZZZZZZZZZZZZZZZZZZ";
cache_data(31 downto 0) <=
               "00000000000000000000000000000000"
               when GND='1' else
               "ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ";
cache_byte_enablen(4 downto 1)   <= "0000"
            when GND='1' else "ZZZZ"; -- external pull-up
cache_write_enablen  <= 'Z';  -- external pull-up
cache_output_enablen <= 'Z';  -- external pull-up
cache_mion(2 downto 0)  <= "000"
            when GND='1' else "ZZZ";   -- external pull-up
end A1;
```

## Contenido del CD

En los siguientes puntos indicamos las rutas dentro del CD que se adjunta con el presente informe.

- Los compiladores: \Compiladores
- La documentación: \Documentación
- Los programas en C y VHDL: \Fuentes
- Informe final: \Informe_final
- Los Correctores y ejecutables (Perl, Awk): \Correctores
- La aplicación final: \Ap_final
- Información relativa al estado del arte: \Estado del Arte

# Referencias bibliográficas

## Bibliografía citada


[1] T. Thakur, D.Dinakar Srinivasa Rao, "SUIF to VHDL translator", Proyecto de grado, Indian Institute of Technology, 1999
http://www.cse.iitd.ac.in/esproject/homepage/docs/projects/dinkar_thakur/report/SUIF2VHDL_minip.ps

[2] M. Aubury, R. Watts, I. Page, "Handel C compiler, Release 1. Documentation and User's Manual", Oxford University, 1996

[3] B. W. Kernighan, D. M. Ritchie, "El Lenguaje de programación C", 2da. edición, Prentice-Hall, ISBN 968-880-205-0, 1991

[4] Xilinx Inc., "Xilinx Netlist Format (XNF) Specification", versión 6.1, 1995
ftp://ftp.xilinx.com/pub/documentation/xactstep6/xnfspec.pdf

[5] Xilinx Inc., "Xilinx Design Reuse Methodology for ASIC and FPGA Designers"
http://www.xilinx.com/ipcenter/designreuse/docs/Xilinx_Design_Reuse_Methodology.pdf

[6] D. Galloway, "Transmogrifier C", Universidad de Toronto, 1996
ftp://ftp.eecg.toronto.edu/pub/software/tmcc/tmcc3.1.tar.Z

[7] J. Poskanzer, "Portable Bitmap Utilities"
http://netghost.narod.ru/gff/graphics/summary/pbm.htm

[8] Lecturas de clases, Departamento de Ingenieria eléctrica y computación de la Brigham Young University
http://www.ee.byu.edu/ee/class/ee621/Lectures/L05.PDF

[9] F. Hill, G. Peterson, "Digital Systems", 3ra. edición, John Wiley & Sons Inc., ISBN 0-471-85936-2, 1987

[10] Material de clases, Facultad de Ingenieria de la Universidad Nacional de Comahue
http://fain.uncoma.edu.ar/TecnicasDigitales/unidades/TDI02Unidad5.pdf

[11] Material de clase, Universidad de Guadalajara
http://www.cucei.udg.mx/~jjme29/orale.pdf

[12] Lecturas de clases, Departamento de Ingenieria eléctrica y computación de la Brigham Young University
http://www.ee.byu.edu/ee/class/ee621/Lectures/L09.PDF

[13] Xilinx - Architectural Synthesis from Behavioral C Code to Implementation in a Xilinx FPGA (Xcell Journal 36 Q2 00)

[14] G. De Micheli, D. Ku, F. Mailhot, T. Truong, " The Olympus Synthesis System ", IEEE Design and Test of Computers, Octubre 1990, pág. 37.

## Bibliografía consultada.

D. Galloway, “The Transmogrifier C Hardware Description Language and Compiler for FPGAs”, Universidad de Toronto, 1995
http://www.eecg.toronto.edu/EECG/RESEARCH/tmcc/tmcc/paper.ps.gz

D. L. Perry ,“VHDL”, 2da. edición, McGraw-Hill, 1993
ISBN 0-07-049434-7

K. Skahill, “VHDL for Programable Logic”, Addison Wesley, 1996
ISBN 0-201-89573-0

L. Terés, Y. Torroja, S. Olcoz, “VHDL Lenguaje estándar de diseño electrónico”, 1er edición, McGraw-Hill, 1997
ISBN 84-481-1196-6

B. Stroustrup, “El lenguaje de programación C++”, 2da. edición, Addison-Wesley, 1995
ISBN 0-201-60104-4

D. Dougherty, A. Robbins, “Seed & Awk”, 2da. edición, O’Reilly, 1997
ISBN 1-56592-225-5

J. Levine, T. Mason, D. Brown, “Lex & Yacc”, 2da. edición, O’Reilly, 1992
ISBN 1-56592-000-7

L. Wall, T. Christiansen, J. Orwant,“Programming Perl”, 3ra edición, O’Reilly, 2000
ISBN 0-5960002-78

R. L. Schwartz, “Learning Perl”, 1ra edición, O’Reilly, 1993
ISBN 1-56592-0422

J. Hayes, "Introducción al Diseño Lógico Digital", 1ra edición, Addison-Wesley, 1996
ISBN 0-201-62590-3

Especificación 0.9 de DPMI en http://www.tenberry.com/web/dpmi/01.html