Universidad de Sevilla
Escuela Superior de Ingenieros
Ingeniería de Telecomuncación

MEMORIA DE PROYECTO FIN DE CARRERA:

Librería de periféricos de la familia PIC-16 en Handel-C

AUTOR: **JOSÉ LUIS CAMUÑAS TAGUAS**

TUTOR: **PROF. DR. JONATHAN NOEL TOMBS**

Este proyecto va dedicado a mis padres,
por su apoyo incondicional en los
momentos más difíciles,
y muy especialmente a Auxi,
por su paciencia y
comprensión.

TABLA DE CONTENIDOS

TABLA DE CONTENIDOS

INTRODUCCIÓN

Capítulo I. INTRODUCCIÓN

1. Título del proyecto

El proyecto objeto de este documento se titula: “Librería de periféricos de la familia PIC-16 en Handel-C”.

2. Antecedentes

Este proyecto está en la línea de investigación y estudio propia del área de tecnología electrónica departamento de ingeniería electrónica de la escuela de ingenieros de la Universidad de Sevilla y más concretamente del Pr. Dr. Jonathan Noel Tombs.

Mediante el desarrollo del proyecto titulado “Implementación en Handel-C del microcontrolador PIC16C54” realizado por D. Antonio Galán Vázquez se pretendió hacer una primera toma de contacto con el lenguaje de alto nivel para la descripción de hardware en FPGA´s llamado Handel-C. Con este fin se realizó un estudio del lenguaje mediante la realización del microcontrolador C54 de la popular familia PIC16 del fabricante Microchip del que existía un estudio previo realizado en VHDL puesto que permitía establecer comparaciones relacionadas con la complejidad del desarrollo del mismo y con el tamaño de puertas lógicas necesario para poder programar la FPGA. Con este proyecto precedente se abordaron aspectos propios del lenguaje Handel-C como herramienta de descripción de sistemas digitales síncronos así como aspectos relacionados con el rudimentario entorno de programación que facilitaba, en su versión 2.1, la empresa Embedded Solutions Ltd. Además se programó el microcontrolador con una serie de programas para probar su buen funcionamiento.

3. Motivación y objetivos del proyecto

La reciente aparición de versiones actualizadas del entorno de programación de Handel-C como son la versión 3.0 Beta de Embedded Solutions Ltd y la DK1 de Celoxica, en la que se incluye un entorno de programación completo, así como la necesidad de probar otros aspectos de descripción de hardware no abordados suficientemente en el estudio anterior, como son los relacionados con las comunicaciones externas, hacen pertinente la realización de este proyecto.

Los objetivos propuestos para la consecución del proyecto son:

- ✓ El estudio y comparación de los entornos de programación recientes.
- ✓ La realización de un ejemplo consistente en la elaboración de un conjunto de periféricos de la familia PIC-16 de Microchip en el que se pone de manifiesto la capacidad de Handel-C en lo referente a descripción de puertos y comunicaciones de procesos que se están ejecutando en paralelo.
- ✓ Una fase de pruebas en la que se realiza la programación de la FPGA Virtex de Xilinx y se comprueban los resultados.

ENTORNO DE PROGRAMACIÓN

Capítulo II. ENTORNO DE PROGRAMACIÓN

Handel-C es un lenguaje de programación diseñado para permitir la compilación de programas en hardware síncrono. Handel-C no es un lenguaje de descripción de hardware sino un lenguaje de programación que posibilita expresar algoritmos a alto nivel.

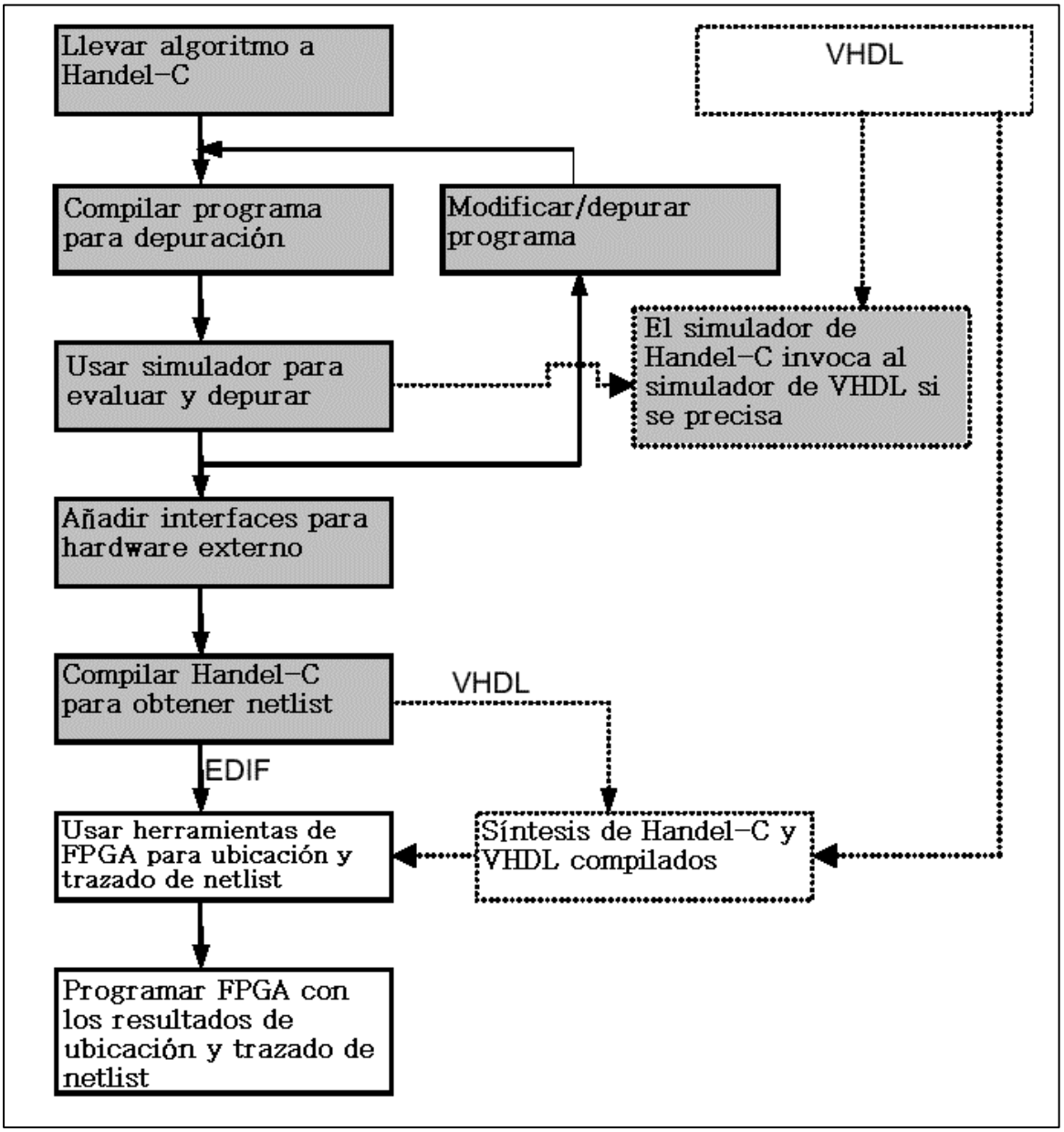


Figura 1. Diagrama de flujo del proceso de programación de algoritmos en Handel-C.

El proceso habitual que hay que seguir para implementar un algoritmo en Handel-C es el expresado en la figura. Los bloques señalados en color gris son posibles desde el entorno de programación (compilador+simulador) que a continuación será detallado y explicado. Las líneas punteadas indican pasos adicionales para el caso de utilizar código VHDL.

1. Introducción

El siguiente capítulo va a proporcionar una breve guía de cómo usar el precompilador, el depurador de código, el enlazador y el compilador de Handel-C incluidos en el entorno de programación de las versiones 3.0 Beta de Embedded Solutions Ltd y DK1 de Celoxica.

2. El entorno de desarrollo de Handel-C

El entorno de desarrollo de Handel-C es estándar para Windows. Contiene cuatro partes principales:

2.1 Ventana de espacio de trabajo (Workspace window)

Es el área donde se organizan los proyectos. Un proyecto consiste en un conjunto de archivos que son necesarios para poder realizar la compilación más información de la FPGA que se va a utilizar para la implementación de lo compilado. Una vez agrupada toda la información que se necesita para un proyecto éste se puede compilar. Cuando se inicia Handel-C la posición por defecto de esta ventana está en la parte izquierda de la pantalla.

2.2 Ventana/s de edición de código (Code editor)

Es donde se crean y editan los archivos de código Handel-C. Cuando se crea o abre un archivo, la posición original del editor de código es la parte derecha de la pantalla.

2.3 Ventana de salida (Output window)

Cuando se compila un archivo o un conjunto de archivos, los mensajes de errores y advertencias (warnigns) son mostrados en la ventana de salida. La posición por defecto de la ventana de salida está en la parte inferior de la pantalla. Esta ventana tiene etiquetas para mensajes de compilación y de depuración.

2.4 Ventanas de depuración (Debug windows)

Cuando se ha compilado el proyecto, se puede similar. La simulación ejecuta el programa permitiendo comprobar en todo momento en qué ciclo de reloj se encuentra ésta así como el valor de cualquier variable que se encuentre dentro del ámbito observable. Estas variables son mostradas en la **Ventana de variables (Variables window**). Se pueden seleccionar las variables que se van a mostrar en la **Ventana de observación en depuración (Debug Watch window).** La posición por defecto de esta ventana es en la parte inferior izquierda de la pantalla. La pila de llamadas (la ruta por la cual se ha llamado a una función es mostrada en la **Ventana de pila de llamadas (Call stack window)**. Se puede observar el ciclo de reloj en curso en la **Ventana de relojes (Clocks window)** y los hilos que actualmente se están ejecutando en la **Ventana de hilos (Threads window).**

Todas las ventanas y barras de herramientas son apilables y configurables como las estándar de Windows.

3. Secuencia lógica de desarrollo

La secuencia normal de desarrollo para proyectos que van a ser ubicados en un solo chip es la siguiente:

1. Creación de un nuevo proyecto.
2. Configuración del proyecto.

3. Adición al proyecto de la totalidad de los archivos de código fuente.
4. Creación del código fuente (en el caso de que este no haya sido escrito desde un editor externo).
5. Asociación (linking) del código con las librerías necesarias.
6. Configuración de los archivos para depuración (debugging).
7. Compilación del proyecto para depuración.
8. Depuración del proyecto.
9. Compilación del proyecto para obtener el integrado.
10. Exportación del archivo obtenido para herramientas de ubicación y rutado de vías (place&route).
11. Ubicación y rutado. No hay información para esto dentro de la documentación de Handel-C.

3.1 Activación del entorno

Handel-C comienza haciendo una de las siguientes acciones:

- Seleccionando la secuencia Inicio>Programas>Handel-C>Handel-C
- Haciendo doble click en un archivo de espacio de trabajo Handel-C (archivos con la extensión .hw). Estos archivos tienen el icono:

Figura 2. Icono de archivo .hw

Haciendo doble clic en el icono de Handel-C (v.3.0 Beta de Embedded Solutions Ltd o en el DK1 de Celoxica) identificables por los siguientes iconos respectivamente:

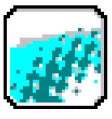



Figura 3. Iconos Handel-C, v3.0 Beta de Embedded Solutions Ltd y DK1 de Celoxica

Cuando comenzamos el programa con un espacio de trabajo vacío encontramos la siguiente pantalla indicada en la figura.

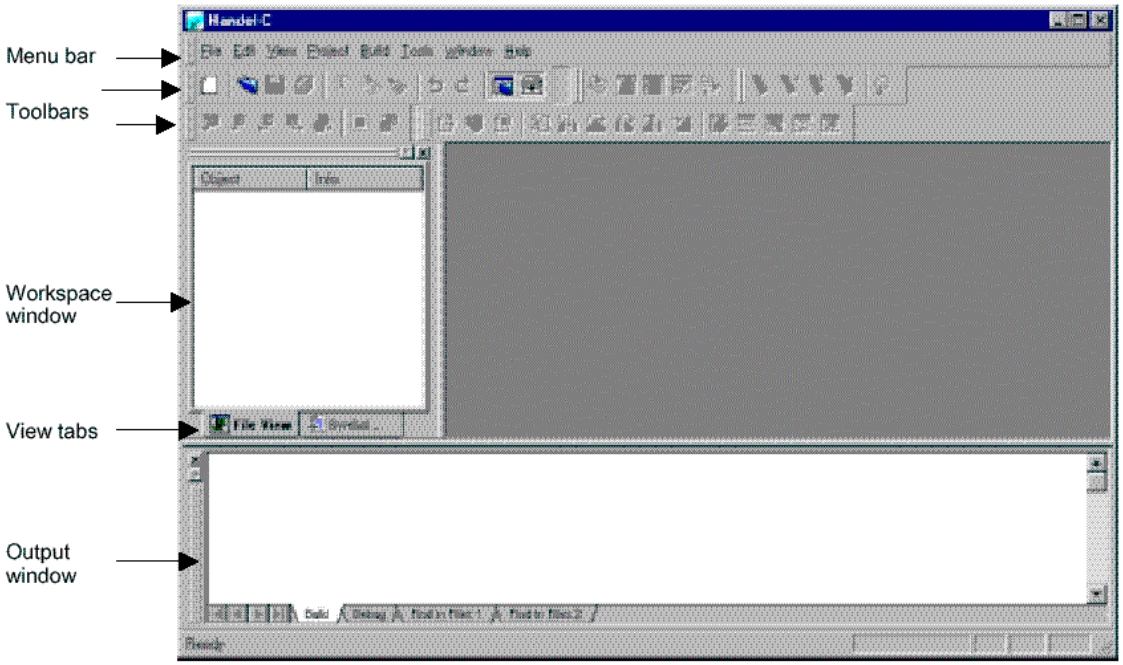


Figura 4. Pantalla inicial en el entorno de programación Handel-C v3.0 Beta.

3.2 Creación del proyecto

- Seleccionamos **New** del menú **File**.
- Seleccionamos la etiqueta **Project** en el cuadro de diálogo que aparece.
- Introducimos el nombre y localización del proyecto, es decir, la ruta de localización del directorio en el que vamos a almacenarlo. Para navegar dinámicamente por los directorios pulsamos el botón ... que está a la derecha del cuadro de localización (**Location**).

Por defecto, un espacio de trabajo nuevo es creado para el proyecto en el mismo directorio que el mismo. Los archivos de espacio de trabajo tienen la extension **.hw**. Los archivos de proyectos tienen la extension **.hp**.

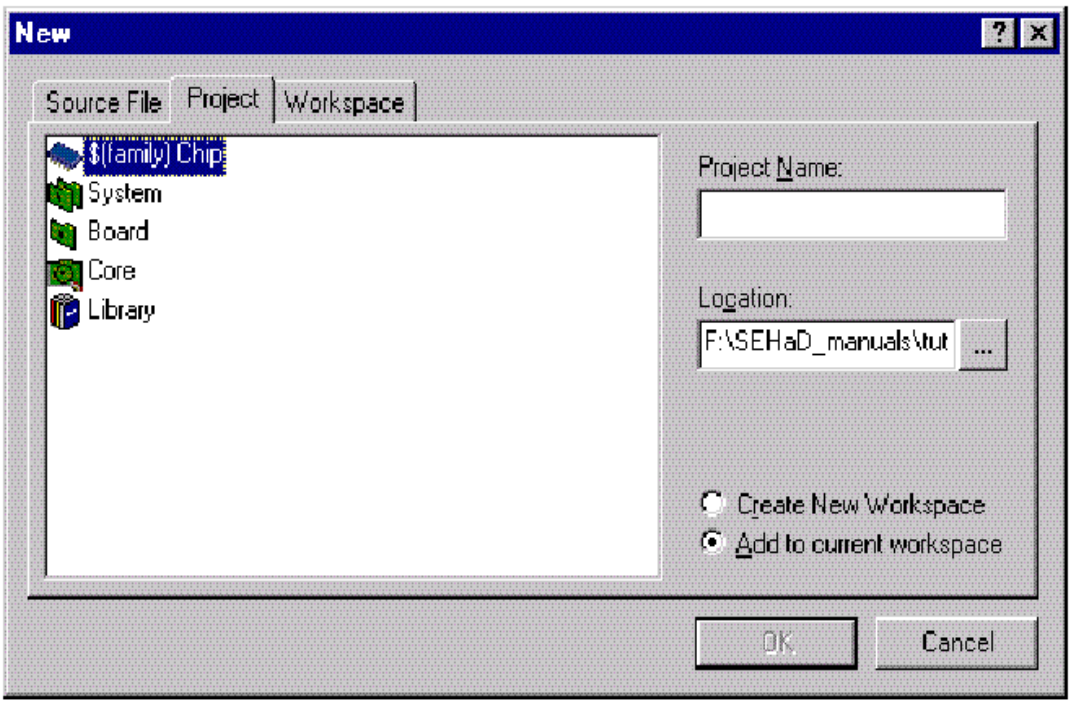


Figura 5. Cuadro de diálogo para la creación de un nuevo proyecto.

Cuando comienza un proyecto nuevo se ha de definir su tipo. Los tipos pueden ser los siguientes:

- Tipo **chip**, **system** o **board**: No orientados a un sistema o producto particular.

Figura 6. Iconos chip, system y board.

- Tipo **core**: Un trozo de código compilado para una arquitectura específica que puede ser utilizado para parte de un diseño mayor.

Figura 7. Icono core.

- Tipo **library**: Código Handel-C precompilado que puede ser reutilizado o vendido en cualquier lugar.

Figura 8. Icono library.

- Tipo **chip**, **system** o **board** predefinidos: Orientados a un producto conocido. Estos sistemas serán optimizados para esos productos y deben ser usados en ellos.

Algunos tipos de proyectos predefinidos comunes son suministrados en el entorno de programación de Handel-C. Para su utilización debemos de:

- Seleccionar el tipo de proyecto apropiado de entre los tipos enumerados en el cuadro **Project**.
- Pulsar **OK**.

3.3 Configuración del proyecto

Una vez creado el proyecto, se deben configurar las características del entorno hardware y software. Estas características definen qué tipo de integrado se va a usar y cómo el compilador, preprocesador y optimizador van a trabajar. Las configuraciones por defecto son apropiadas para un proyecto nuevo que se desee depurar.

3.4 Adición de archivos al proyecto

Los archivos de código Handel-C que se desean añadir al proyecto pueden ser algunos que se hayan escrito anteriormente (utilizando cualquier editor) o uno vacío nuevo.

Creación de un archivo nuevo

- Seleccionamos **File>New,** y pulsamos la etiqueta **Source File**.

- Seleccionamos en el panel izquierdo si se trata de un archivo cabecera (header) o fuente (source).
- Seleccionamos el proyecto al cual deberá pertenecer el archivo de la lista descendente de proyectos actuales.
- Establecemos la ubicación (ruta del directorio en donde se va a almacenar el archivo) bien escribiendo el nombre de la ruta en la casilla correspondiente, bien seleccionando el directorio pulsando previamente el botón [...]
- El editor de código se abrirá.

Adición de un archivo existente

Seleccionamos **Project**>**Add to Project**>**Files** y exploramos el árbol de directorios hasta localizar el archivo que queremos añadir. Podemos añadir varios archivos de un directorio seleccionándolos (usando la tecla Control + botón izdo. del ratón , etc...) o bien oprimiendo el botón derecho del ratón en el proyecto y seleccionando **Add Files to Folder** desde el menú rápido.

Borrado de archivos de un proyecto

Se pueden borrar archivos de un proyecto seleccionándolos y pulsando la tecla Supr o seleccionando **Edit>Delete**. Esto no borrará los archivos del disco duro.

> ***Al abrir un archivo de código existente no se incluye en el proyecto. No será construido ni compilado. Se han de añadir archivos al proyecto explícitamente.***

3.5 Redacción de código fuente

Se puede escribir código fuente Handel-C en el editor de código suministrado en el entorno de programación o bien desde un editor externo.

Tener un archivo abierto en el editor de código no significa que sea parte del proyecto. Los únicos archivos que serán compilados y construidos son aquellos que han sido añadidos al proyecto.

3.6 Configuración para la depuración de código

Hay varios métodos que pueden ayudar a depurar código Handel-C. Se pueden dividir en dos tipos:

- Código que será descartado automáticamente por el compilador si no se hace una compilación explícita para la depuración de código, por ejemplo la directiva **with {infile = "*file*"}.**

- Código que es creado para la depuración o para la creación de hardware exclusivamente y que se compila alternativamente para un caso o para el otro. En estos casos se puede usar las directivas del preprocesador: **#ifdef**, **#ifndef**, **#elif**, **#else** y **#endif**, como en el siguiente ejemplo:

```
#ifdef DEBUG
file_chan ? var; // Lee de un archivo
#elif SIMULATE
sim_chan ? var; // Lee del simulador
#else
HardwareMacroRead(var); // Interfaz HW real
#endif
```

Por defecto, **DEBUG** y **SIMULATE** son definidas para la compilación para depuración del código y **NDEBUG** para otras compilaciones.

Técnicas de programación usadas para depuración de código.

- Sustituir los canales del simulador por una interfaz de canales hardware.
- Usar las directivas **assert** para impedir la compilación de un determinado fragmento de código cuando la condición sea incierta.
- Sustituir ficheros de entrada por entradas desde canales externos.
- Escribir el contenido de las variables en archivos de salida.

3.7 Construir y compilar para la depuración de código

La compilación por defecto está configurada para la depuración de código. Inicialmente es improbable que se necesiten hacer cambios en este tipo de compilación. El compilador crea un archivo que es compilado a código autóctono del PC usando el programa Microsoft Visual C++. Esto crea la simulación del integrado.

Para construir y compilar el proyecto, únicamente hay que seleccionar **Build** desde el menú **Build,** o bien pulsar el icono correspondiente (ver figura).

Figura 9. Icono Build.

Los mensajes del compilador aparecerán en la ventana de salida pulsando la pestaña **Build.**

3.8 Depuración y simulación

Seleccionamos **Start debug** desde el menú **Build** o bien pulsamos el icono **Step into** de la barra de herramientas relacionadas con la depuración de código. El menú **Debug** remplazará al menú **Build**. Se puede avanzar a lo largo del código ejecutándolo sentencia a sentencia, ciclo a ciclo de reloj, desde el

punto actual hasta un determinado punto de ruptura o indefinidamente hasta pulsar el botón de interrupción del proceso de depuración como detallaremos más adelante.

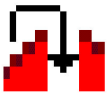


Figura 10. Botón de comienzo del proceso de depuración (Step into).

Las expresiones que son completadas al final del ciclo de r eloj en curso son marcadas con un puntero.

El código de colores para los punteros es el siguiente:

- **Amarillo**: Punto actual.
- **Blanco**: Otros puntos en este hilo se están ejecutando en el ciclo de reloj en curso.
- **Gris**: Otros puntos en otros hilos se están ejecutando en el ciclo de reloj en curso.

Para establecer un punto de ruptura (breakpoint), hacemos click en el editor de código sobre la línea en la que queremos que se produzca la ruptura y después pulsamos el botón de breakpoint.

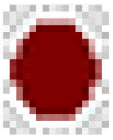


Figura 11. Botón de ruptura del proceso de depuración (breakpoint).

Un círculo rojo aparecerá al comienzo de esa línea. Cuando el depurador alcanza esa línea, el proceso se detendrá.

3.9 Optimización del código

Se puede conocer la profundidad y velocidad del código creado compilándolo con la opción -e seleccionable en la etiqueta Compiler en el cuadro de diálogo Project Settings. Esto genera:

- Un archivo HTML para el proyecto, *proyecto.html*
- Un archivo HTML para cada archivo perteneciente al proyecto *archivos_c.html.* Estos archivos resaltan el código conforme al código de área y temporización.

3.10 Compilación final (Tipo Release)

Cuando se ha realizado el proyecto con éxito se puede seleccionar Build>Set Active Configurations y elegir el tipo de construcción requerida de entre las configuraciones disponibles.

El tipo Release permite simular el proyecto sin los retrasos inherentes del proceso de depuración. También permite compilar las librerías Handel-C sin la información de depuración para proteger la propiedad intelectual.

Se obtiene un fichero de VHDL y otro de EDIF. Son archivos que están dispuestos para su ubicación y rutado (place&route) en la FPGA. Por defecto la mayoría de las optimizaciones están activadas.

4. Configuraciones del proyecto

Las configuraciones del proyecto definen cómo éste va a ser compilado y construido. Seleccionando **Project>Settings** aparecerá el cuadro de diálogo llamado **Project Settings**. Las diferentes configuraciones de aspectos: generales, integrado, preprocesador, compilador, optimizador, enlazador, depurador y comandos para la construcción son visibles haciendo click en la pestaña correspondiente.

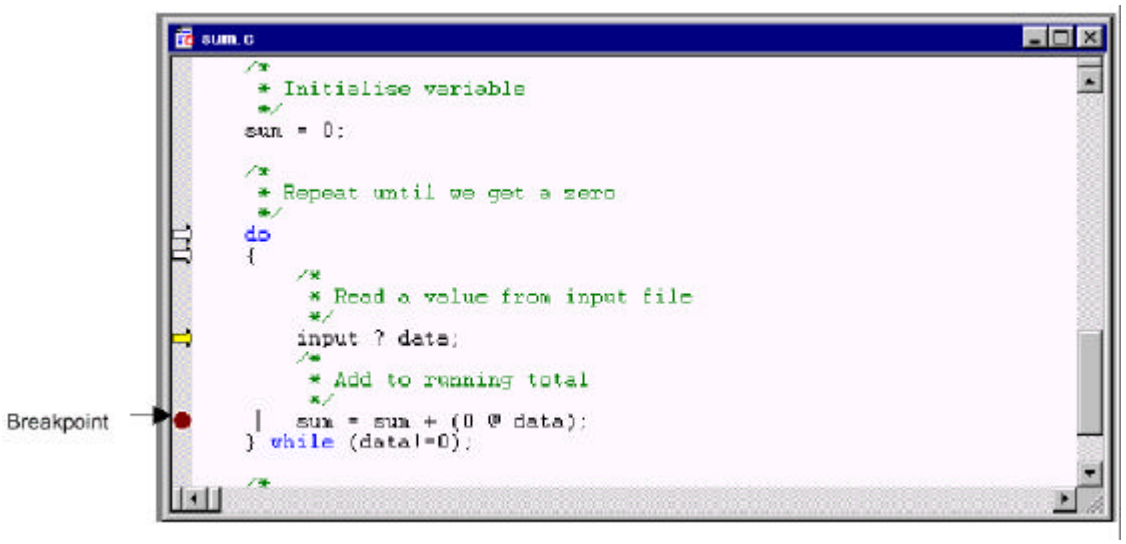


Figura 12. Cuadro de diálogo de configuración del proyecto.

ETIQUETA	OBJETO	SIGNIFICADO	VALOR	POR DEFECTO
General				
	Intermediate files	El subdirectorio donde se almacenan los archivos intermedios	Nombre de la ruta del directorio relativa al directorio del proyecto	*configuration name*
	Output files	El subdirectorio donde se almacenan las salidas finales (.dll, netlist, etc.)	Nombre de la ruta del directorio relativa al directorio del proyecto	*configuration name*
Chip				
	Family	La familia que contiene el componente que va a ser implementado	Seleccionar la familia desplegando la lista hacia abajo	Valor deseado

	Part	Número de componente a implementar	Escribir el número de componente	Depende del proyecto
Preprocessor				
	Preprocessor definitions (-D)	Es equivalente a la directiva #define	Configuración según uso	**DEBUG, NDEBUG, SIMULATE**
	Additional include directories (-l)	Añade directorios a la ruta de búsqueda de directorios include	Configuración según uso	Ninguno
	Additional preprocessor options (-cpp)	Añade comandos cpp	Ver ayuda	Ninguno
Compiler				
	Generate debug information (-g)	Hace que el compilador genere información para el degugger	Comprobar que está activado.	Activado
	Generate warning messages (-W)	Hace que el compilador genere mensajes de aviso	Comprobar que está activado.	Activado
	Generate estimation information (-e)	Hace que el compilador genere archivos HTML dando información de la profundidad y la velocidad del código.	Comprobar que está activado.	Desactivado

Optimisations				
	Various levels of optimizations	Comprobar el nivel necesario	Depende de la implementación	
Linker				
	Output format	Seleccionar la implementación para el compilador	Determinado por la configuración	Valor deseado
	Save browse info	Almacenar información necesaria para exploración de símbolos	Activado	Activado
	Additional Library Path	Ruta de directorios para buscar librerías	Configurado según uso	
	Command line for building simulator DLL (-cl)	Enlazar opciones definidas en el archivo cl.cf	Define cómo el compilador C es llamado para compilar simulator.dll	Configuraciones de cl.cf instaladas (sólo Debug y Release)
	Object/library modules	Señala módulos que deben ser incluidos en la construcción del proyecto	Configuración según uso	
Debugger				
	Working directory	Directorio que el simulador usa como directorio de trabajo en curso	Nombre de ruta del directorio relativa al directorio del proyecto	Directorio del proyecto en curso (.)

Build commands				
	Compilation commands	No implementado actualmente		Ninguno
	Output files	No implementado actualmente		Ninguno

4.1 Configuraciones independientes para archivos

Se pueden crear configuraciones independientes para un archivo. Esto está orientado para cambiar el nivel de optimización de un determinado archivo. Las configuraciones de proyecto para un archivo sobrescriben las configuraciones generales del proyecto.

- Para crear configuraciones para un archivo hay que abrir el cuadro de diálogo **Project Settings** (o hacer click con el botón derecho en File View y seleccionar **Settings**, o bien seleccionar **Project>Settings**).
- Seleccionar el nombre del fichero en el que se pretende actuar en el cuadro de archivos dentro del cuadro de diálogo **Project Settings.**
- Realizar los cambios oportunos.

4.2 Configuraciones predefinidas

Hay tres tipos de configuraciones que se pueden seleccionar para construir la aplicación

Debug (la que está por defecto).

Release

Target (VHDL, EDIF, etc.)

Debug se usa para crear una configuración que permita simular y depurar el proyecto en el PC. En modo depuración se puede ver el contenido de los registros y avanzar a lo largo del código de programa.

El modo **Release** se usa para crear propiedad intelectual (librerías) en Handel-C. Se crea código compilado que no tiene mensajes de depuración y que puede ser utilizado en otros programas. El modo **Release** también puede ser usado para simulación a gran velocidad.

En modo **Target** se puede obtener un conjunto de puertas que está dispuesto para ser ubicado en la FPGA. ver figura.

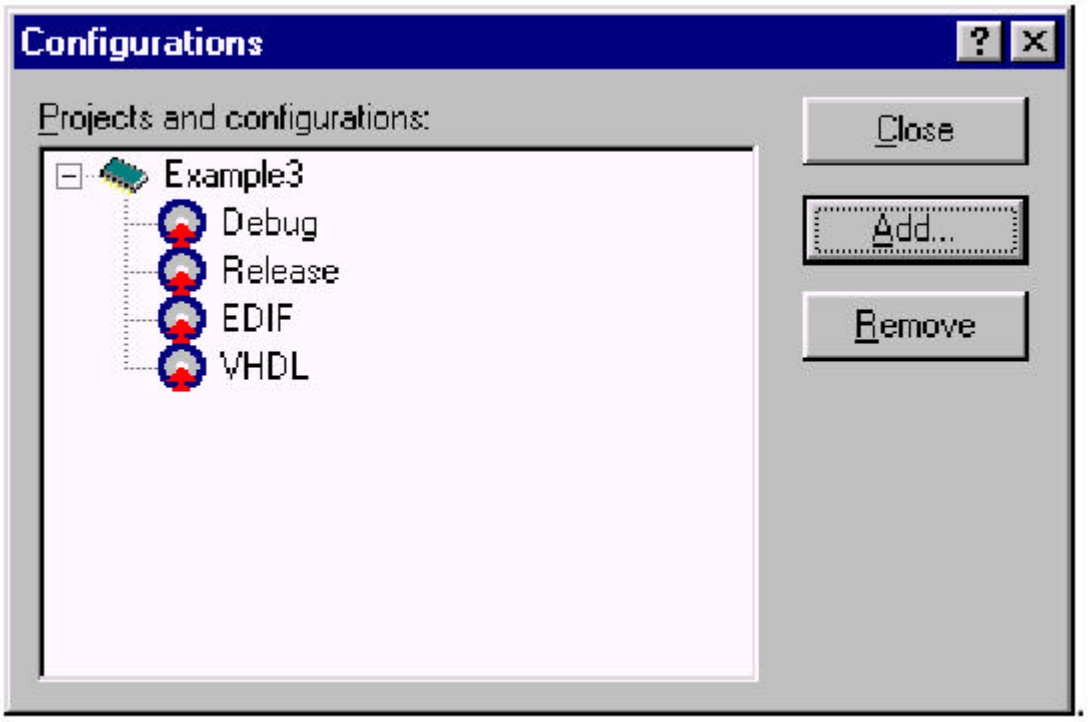


Figura 13. Cuadro de diálogo para las configuraciones.

4.3 Definir configuraciones

Se puede grabar una combinación particular de configuraciones como configuraciones de proyecto usando el menú **Build>Configurations.** Estas configuraciones definidas por el usuario pueden ser utilizadas solamente en el proyecto.

Handel-C tiene cuatro configuraciones posibles: Build, Debug, VHDL y EDIF. Se puede copiar una de ellas y entonces hacer modificaciones sobre la misma. Para ello:

Seleccionamos **Build>Configurations...**
Hacemos click en el botón **Add** en el diálogo que aparece.
Introducimos un nombre para la nueva configuración y seleccionamos el tipo de configuración que queramos usar como base en el cuadro **Copy setting from.**

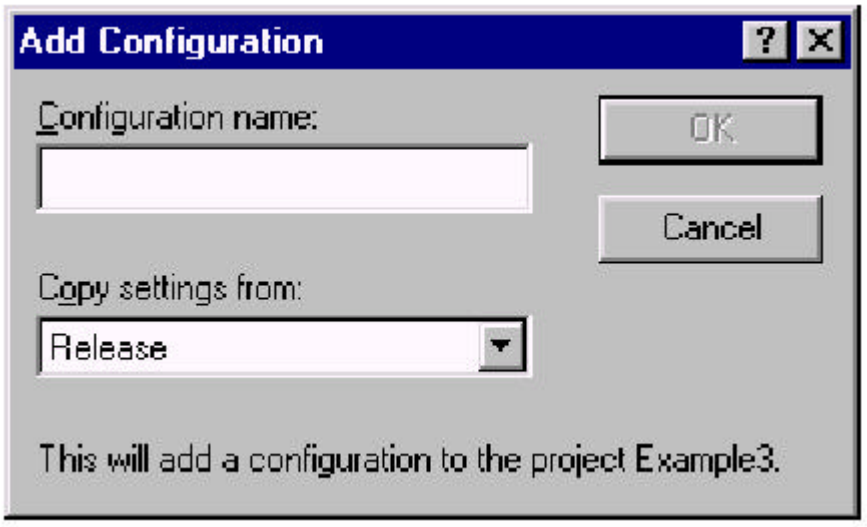


Figura 14. Cuadro de diálogo para crear configuraciones particulares.

4.4 Configuraciones más complejas

Si se van a tener múltiples proyectos (quizá se necesiten tener varios circuitos independientes en el mismo integrado), es mejor crear un espacio de trabajo primero y después añadirle proyectos.

Si se tiene un espacio de trabajo ya creado hay que abrirlo primero. Aparte de eso, se debe seleccionar **New** del menú **File**. Creamos un nuevo espacio de trabajo para almacenar el/los proyecto/s. Después introducimos su nombre y localización (nombre de la ruta del directorio en donde será

guardado). Para esto, o bien se escribe el nombre de la ruta en el cuadro **Location**, o bien se usa el botón [...] para acceder al directorio. Los archivos de espacio de trabajo tienen la extensión .hw.

Adición de un proyecto existente en un espacio de trabajo

Para esto sólo hay que seleccionar **Insert Project into Workspace** del menú **Project.**

Creación de un proyecto complejo

Cuando el proyecto es una tarjeta o un sistema probablemente contendrá subproyectos. Cuando se crea un tipo nuevo de proyecto complejo (editando un archivo nuevo con extensión .cf), aparece un cuadro de diálogo cuando pulsamos **OK.** El cuadro de diálogo **New Project Components** pregunta por los proyectos que se van a usar como componentes del proyecto complejo. Se puede entonces crear un nuevo proyecto o seleccionar uno dentro del espacio de trabajo de la lista descendente que aparece. Si el proyecto existe pero no en el espacio de trabajo se puede añadir usando el botón **Insert Project**.

Para asegurar que los subproyectos están incluidos cuando se construye el proyecto complejo, se pueden establecer los subpreyectos como dependientes. Para esto hay que seleccionar **Project>Dependencies**... Aparecerá entonces una lista con los proyectos en el espacio de trabajo. Seleccionamos entonces aquellos que deseamos reconstruir con construyamos el proyecto complejo.

5. Dependencias

Las dependencias son usadas para asegurar que archivos que no son parte del proyecto sean actualizados durante la construcción. También especifican el orden en que los archivos deben ser compilados y construidos.

Hay tres tipos de dependencias usadas en Handel-C:

Dependencias de proyecto.
Dependencias de archivo.
Dependencias externas.

Las únicas que se pueden cambiar directamente son las de proyecto. Las otras muestran información calculada por el compilador.

5.1 Dependencias de proyecto

El diálogo que comienza con **Project>Dependencies...** permite seleccionar otros proyectos dentro del espacio de trabajo que van a depender del proyecto. Los proyectos enumerados en esa lista serán reconstruidos tantas veces como el proyecto sea reconstruido.

Si se está construyendo un proyecto complejo, tal como una tarjeta o un sistema que tiene varios integrados dentro, se puede crear un proyecto separado para cada integrado y hacer que dependan del proyecto del sistema.

5.2 Dependencias de archivo

Las dependencias de archivo son enlistadas en las propiedades de archivo. Estas especifican los archivos anexos de usuario que no son incluidos en el proyecto y que son necesarios para compilar y construir un archivo seleccionado. También especifican qué archivos dentro del proyecto deben ser compilados antes que ese archivo.

Estas dependencias se generan cuando se compila un archivo. Se pueden examinar seleccionando un archivo en el cuadro **File View** de la ventana de espacio de trabajo y pulsar **Alt + Intro** o pulsar el botón derecho del ratón sobre el nombre de archivo y seleccionar **Properties** en el menú rápido.

5.3 Dependencias externas

La carpeta de dependencias externas aparece en la ventana de trabajo después de que éste haya sido construido. Contiene una lista de archivos cabecera requeridos por el proyecto y que no están incluidos en él.

6. La ventana de espacio de trabajo

La ventana de espacio de trabajo contiene espacios de trabajo y proyectos. Un espacio de trabajo es simplemente una zona en la que conservamos proyectos. Esto permite organizar los archivos que se necesitan para cada proyecto. Normalmente se va a usar un espacio de trabajo por sistema (que será nuestro objetivo).

Un proyecto consiste en todo aquello que se necesita para crear uno o más archivos de componentes *(netlist)* dispuestos a ser ubicados y rutados *(place&route)* en una FPGA, junto con las configuraciones del proyecto. Estas configuraciones suministran información de dónde se localizan los archivos que conforman el proyecto. el integrado objetivo para el proyecto, cómo va a trabajar el compilador así como requisitos para la optimización. Los proyectos pueden ser librerías (Handel-C compilado que no está pensado para una aplicación en particular), núcleos (fragmentos de código como funciones compiladas), listas completas de componentes para un integrado, tarjetas (listas de componentes para varios integrados con una configuración específica) o sistemas (combinación de tarjetas, etc.)

La ventana de espacio de trabajo tiene dos posibles vistas:

Vista de archivo (File View)

Vista de símbolos (Symbol View)

6.1 Vista de archivo

La vista de archivo muestra el espacio de trabajo, sus proyectos, sus archivos fuente y sus carpetas. Si hay varios proyectos en un solo espacio de trabajo el nombre del proyecto en curso estará en negrita.

La vista de archivo muestra las estructura de los archivos en el proyecto. Esta estructura no guarda relación alguna con la forma en que los archivos están almacenados en el disco duro. Esto permite crear dependencias (qué archivos se necesitan para este proyecto y de qué archivos o proyectos depende) y gestionar el proyecto viendo qué archivos se están usando en el mismo.

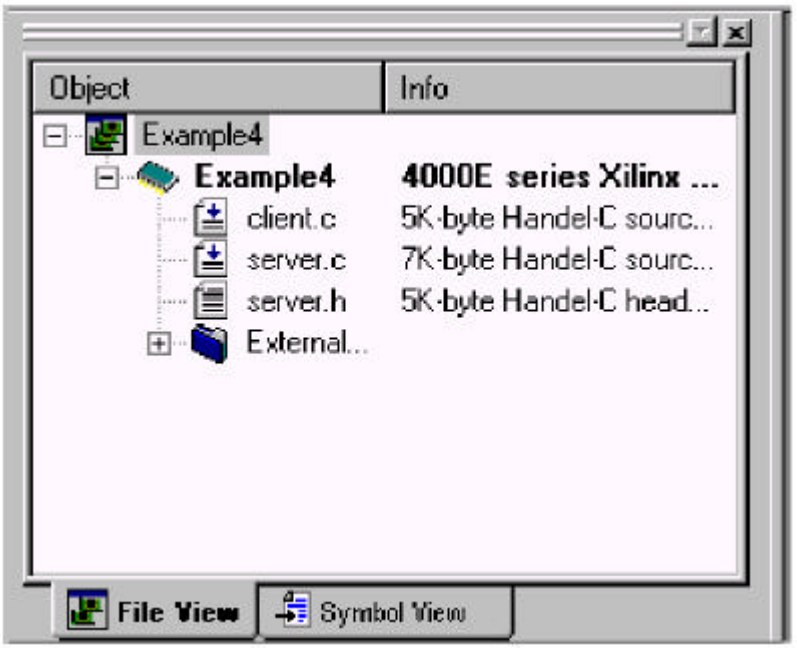


Figura 15. Vista de archivo de la ventana de espacio de trabajo.

Se puede ajustar el ancho de las columnas destinadas a **Object** e **Info** desplazando el filo de la columna cabecera.

Haciendo doble click en un archivo fuente se abrirá el mismo en el editor de código. Haciendo doble click en cualquier otro sitio extenderá o contraerá esa rama del árbol del espacio de trabajo.

Si pulsamos el botón derecho del ratón sobre el nombre de un archivo o sobre un directorio nos mostrará un lista de los comandos usados comúnmente.

Propiedades de archivo

Cuando seleccionamos un archivo o directorio en la ventana de espacio de trabajo haciendo **View>Properties** aparece lo representado en la figura.

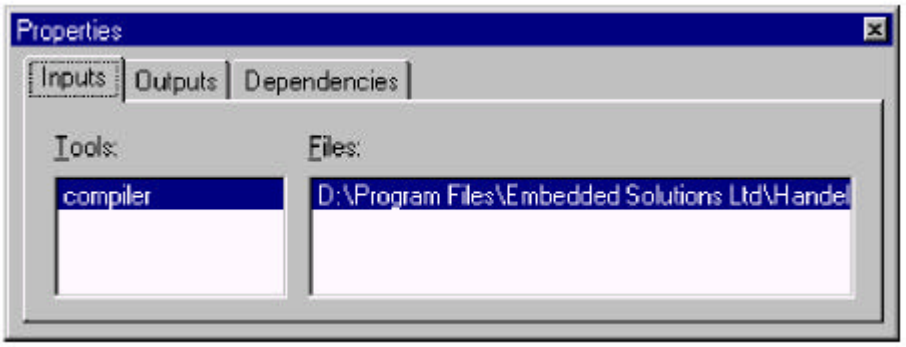


Figura 16. Propiedades de archivo.

En la pestaña **Inputs** se pueden ver herramientas usadas y el archivo que se requiere para su uso.

En la pestaña **Outputs** se muestran los archivos generados por la herramienta específica.

Con la pestaña **Dependencies** podemos ver los archivos cabecera (dependencias) que necesita este archivo.

Gestión de los archivos del proyecto

Se pueden ordenar los archivos dentro del proyecto en carpetas. Estas carpetas se usan únicamente para organizar los archivos. No existen como carpetas en el disco duro y no tienen efecto alguno en la estructura de directorios.

Para organizar el espacio de trabajo hacemos lo siguiente:

Seleccionamos **Project>Add to Project>New Folder.**

Escribimos el nombre de la carpeta en el cuadro de diálogo que aparece.
Escribimos la extensión de los archivos que deberá contener. Se puede dejar este espacio en blanco.
Hacemos click en OK.
Aparecerá una nueva carpeta en la ventana de vista de archivo.
Arrastramos los archivos que queremos incluir en la carpeta sobre ésta.

6.2 Vista de símbolos

Un símbolo es cualquier cosa definida por el usuario (funciones, variables, macros, typedef, enums, etc.).

La vista de símbolos permite ver lo que se tiene en un proyecto. Antes de construir un proyecto está vacía. Cuando construimos un proyecto con la información para exploración habilitada (*browse information* habilitada por defecto en la opción **Debug)** se crea una tabla de símbolos que permite examinar los símbolos definidos y usados en el proyecto.

Icono	**Significado**
	Función compartida, procedimiento o expresión
	Función no compartida o macro
	Variable
	Memoria (RAM, ROM, WOM o MPRAM)
	Canal
	Interfaz externa

	Señal
	Posiciones apiladas que contienen al objeto asociado (p. ejemplo: macros recursivas)
	Posición en el archivo que contiene la definición del objeto

Seleccionando la etiqueta **Symbol View** de la ventana de espacio de trabajo podremos ver iconos que representan variables lógicas y estructurales, funciones y procedimientos.

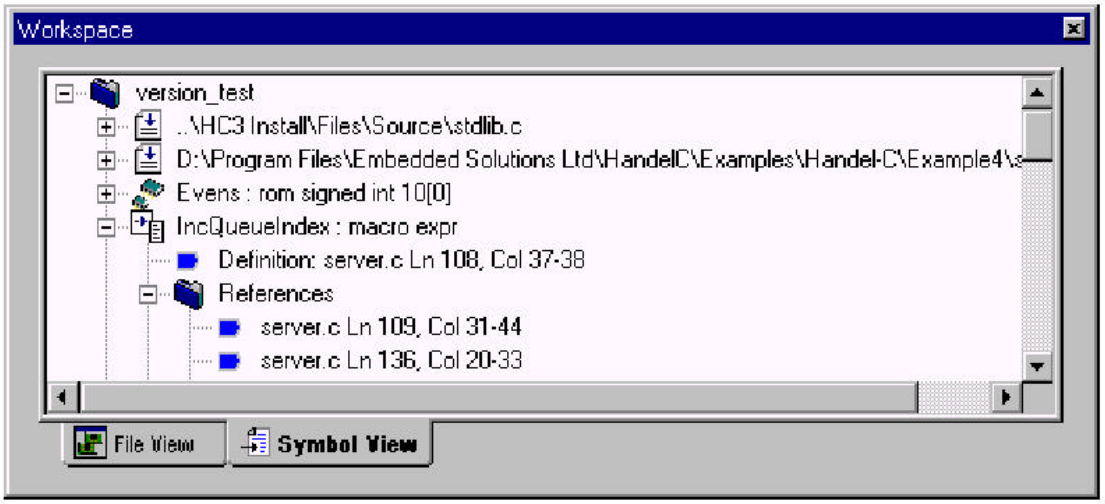


Figura 17. Vista de símbolos de la ventana de trabajo.

Cada icono está identificado por su definición y uso (referencia). Los símbolos externos (variables externas y nombres de funciones) aparecen en orden alfabético.

Haciendo doble click en un símbolo hará que se expanda (si es posible). Si no es así abre el archivo de código fuente pertinente con la línea etiquetada conveniente.

Los símbolos locales aparecen en orden alfabético dentro de la función o procedimiento donde están definidos.

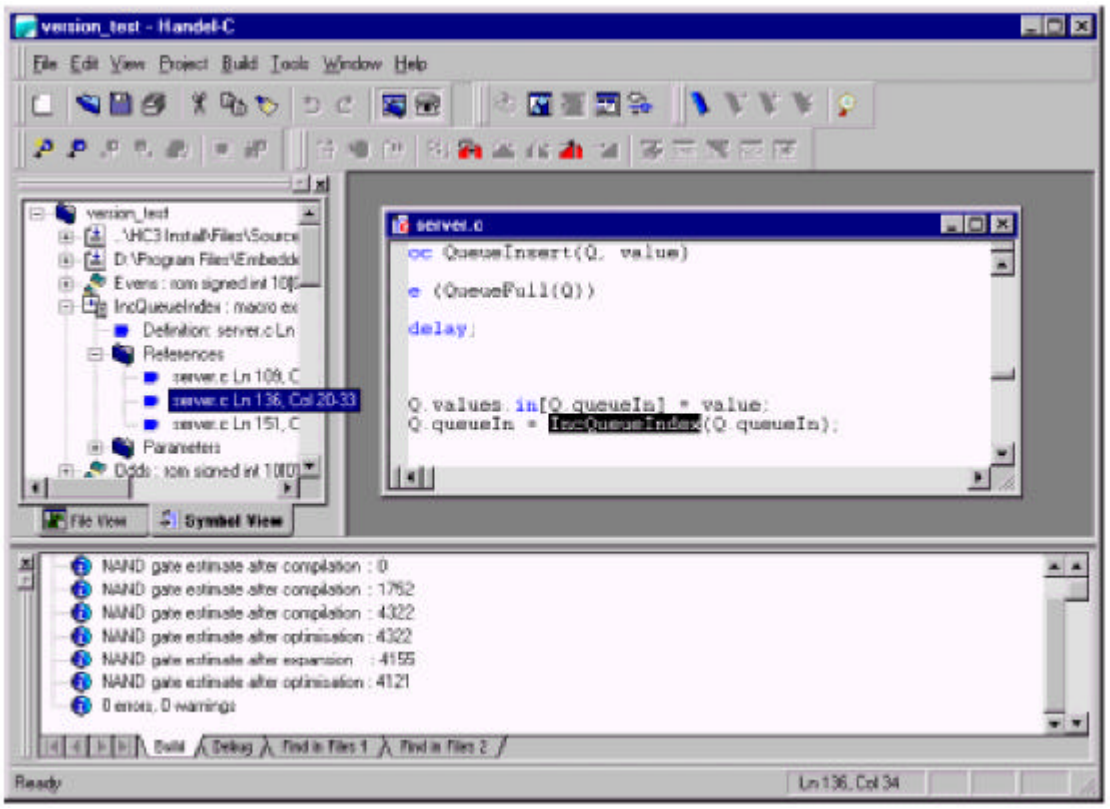


Figura 18. Utilización de la vista de símbolos.

7. El explorador de código fuente

Se puede hacer una exploración de definiciones y referencias sin usar la vista de símbolos. Cuando seleccionamos **Source Browser** del menú de herramientas **Tool** aparecerá un cuadro de diálogo como el mostrado en la figura.

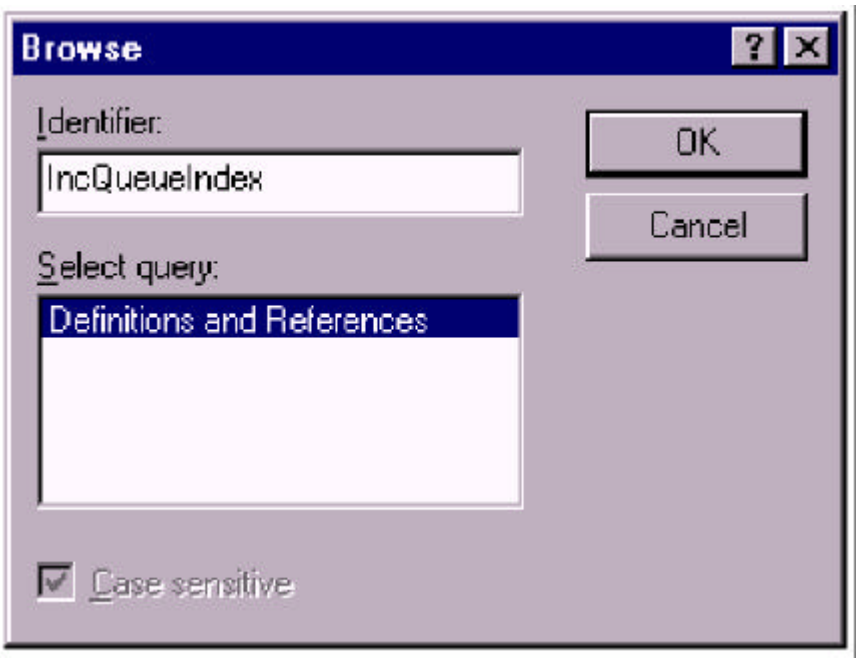


Figura 19. El explorador de código fuente.

Introduciendo el símbolo que estamos buscando podremos ver un cuadro de diálogo en el que se muestran sus definiciones y las referencias a dicho símbolo.

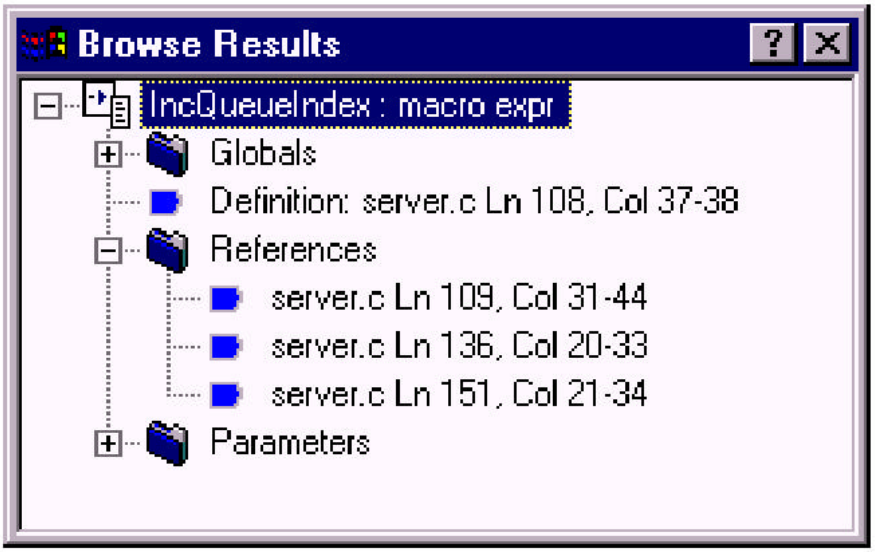


Figura 20. Resultados de la búsqueda de un símbolo en el explorador.

Si seleccionamos el nombre de un símbolo en un archivo fuente, podemos usar el explorador de comandos y botones para encontrar su definición y referencias en todos los archivos usados en un proyecto.

Botón	Comando	Función
	***Go to definition***	Salta a la línea de código fuente donde la variable se ha definido.
	***Go to reference***	Salta a la primera línea de código fuente donde la variable es usada.
	***Previous Definition/Reference***	Salta a la definición o referencia previa.
	***Next Definition/Reference***	Salta a la definición o referencia siguiente.
	***Pop context***	Retorna a la posición original antes de que empezara el navegador de fuente.

8. Edición

8.1 El editor de código

El editor de código es un editor simple que reside en su propia ventana. La sintaxis tiene códigos de color. Se puede cambiar este código de color seleccionando la etiqueta **Tab** del cuadro de diálogo **Tools>Options.** Los valores por defecto son:

Se pueden usar comandos de edición estándares dentro de la ventana de código. Éstos son accesibles desde el menú **Edit.**

OBJETO	COLOR
Comentarios	Verde
Palabras reservadas de Handel-C	Azul
Números	Negro
Cadenas de caracteres	Negro
Operadores	Negro

La siguiente tabla resume estos comandos:

Comando	Método abreviado	Descripción de la función
Undo	*Ctrl.+Z*	Deshace el último cambio del documento activo o del espacio de trabajo.
Redo	*Ctrl+Y*	Deshace el último Undo.
Cut	*Ctrl+X*	Copia la selección actual y la borra.
Copy	*Ctrl+C*	Copia la selección actual y la copia en el portapapeles.
Paste	*Ctrl+V*	Copia el portapapeles en la selección actual.
Delete	*Supr*	Borra la selección actual.
Find	*Ctrl+F*	Encuentra una cadena o una expresión común en el archivo en curso.
Find in files		Encuentra una cadena o una expresión común en los archivos seleccionados.
Replace	*Ctrl+H*	Sustituye una cadena o una expresión común por otra en el archivo en curso.

El menú **Edit** también tiene los submenús **Bookmarks** y **Browse** y el comando de puntos de ruptura.

8.2 Comandos de búsqueda

Handel-C tiene comandos simples de búsqueda y sustitución (**Find** y **Replace**) que permiten búsqueda de texto en el archivo en curso, y el comando **Find in Files** que permite buscar una cadena en todos los archivos de un directorio. La salida de ese comando puede ser enviada en dos tipos distintos de ventana, permitiendo ver los resultados de dos búsquedas. Para elegir qué ventana está seleccionada hay que seleccionar, o no, la etiqueta **Output to pane 2** en el diálogo **Find in Files**.

Estas búsquedas funcionan línea a línea, lo que quiere decir que no se puede localizar texto que ocupe más de una línea.

También se pueden buscar expresiones comunes. Para hacer esto, habilitar **Regular expresión** en los cuadros de diálogo **Find** y **Find in Files**. Las expresiones comunes soportadas son las descritas en la lista.

Expresiones Comunes	Descripción
(x)	Los caracteres o las expresiones entre paréntesis.
.	Cualquier carácter individual.
^	Comienzo de línea.
$	Fin de línea.
\t	Tabulador.
x|*y*	Correspondencia con *x* o *y*. Por ejemplo, la búsqueda *a(team|class)* se corresponderá con *ateam* o con *aclass*.
x*	Ninguna, una o más copias de x. Por ejemplo ba*c se corresponderá con bc, bac, baac, baaac, ...
x?	Ninguna o una x. Por ejemplo, ba?c se corresponde con bc o con bac.
x+	Al menos una x. Por ejemplo ba+c se corresponde con bac, baac, baaac, ..., pero no con bc.

[xyz] [x-y]	Se corresponderá con un carácter de los que están entre corchetes. Se usa - para incluir todos los caracteres de un determinado rango (p. ej., [_A-Za-z] se corresponderá con un subguión o con cualquier letra del abecedario mayúscula o minúscula). Para incluir el carácter '-', '[' o ']' se han de poner al principio o al final de la lista, p. ej, [xyz-], []xyz].
[^xyz]	Se corresponderá con un carácter que no esté entre los corchetes. Por ejemplo *x[^0-9]* se corresponderá con *xa* pero no con *x0* ó *x2*.
\x	Se corresponderá con el carácter *x*, incluso si *x* es uno de los caracteres reservados ^$[].*+? referidos arriba. Por ejemplo *^pig* se corresponderá con la cadena *pig* a comienzo de una línea, pero *\^pig* se corresponderá con la cadena *^pig* en cualquier lugar de la línea.

Submenú de *Bookmarks*

El submenú de Bookmarks permite poner y quitar *bookmarks* dentro de los archivos. Una vez puestas una o más *bookmarks* en el archivo, nos podemos mover de una a otra seleccionando **Next Bookmark** (F2) o **Previous Bookmark** (Shift+F2).

Para poner una *bookmark*, hay que:

- Seleccionar la línea en la que queremos ubicarla.
- Apretar el botón de *toggle bookmark* o bien apretar el botón derecho del ratón en la línea y seleccionar **Toggle bookmark** del menú rápido que aparece, o también seleccionar **Edit>Bookmarks>Toggle Bookmark** (Ctrl F2).

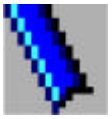


Figura 21. Botón de toggle bookmark

Para moverse a una *bookmark*:

- Seleccionar **Edit>Bookmarks>Next Bookmark** (F2) o pulsar el botón *next bookmark* para moverse hacia la próxima marca.

Figura 22. Botón de next bookmark.

- Seleccionar **Edit>Bookmarks>Previous Bookmark** (Shift F2) o pulsar el botón de *previous bookmark* para moverse hacia la marca anterior.

Figura 23. Botón de previous bookmark.

Para quitar una *bookmark*:

- Seleccionamos la línea en la que queremos quitar la marca.
- Presionamos el botón *toggle bookmark* o bien,
- Presionamos el botón derecho del ratón sobre la línea en cuestión y seleccionamos **Toggle Bookmark** del menú rápido que aparece o bien,
- Seleccionamos **Edit>Bookmarks>Toggle Bookmarks** (Ctrl F2).

Para quitar todas las *bookmarks*:

- Seleccionamos **Edit>Bookmarks>Clear All Bookmarks (**Ctrl Shift F2) o presionamos el botón *clear all bookmarks* para borrar las marcas.

Figura 24. Botón clear all bookmarks.

Comando de puntos de ruptura

El comando de puntos de ruptura o **Breakpoints (** Alt+F9) permite poner, habilitar e inhabilitar puntos de ruptura. Como explicaremos cuando describamos el depurador.

Submenú explorador

El submenú explorador permite encontrar definiciones y referencias de variables seleccionadas u otros símbolos. Si se hace un cambio en una variable, esta es una manera rápida de encontrar cualquier lugar en donde sea usada. Su uso es el explicado en el apdo. 3.7.

Guardar los cambios

Si no se han guardado los cambios realizados en un archivo, aparecerá un asterisco después del nombre del archivo en barra de título. El entorno preguntará si se desean guardar los cambios cuando se cierra un archivo.

No se pueden hacer ediciones en archivos de sólo lectura.

9. Archivos y rutas

El directorio en curso es el directorio que contiene el archivo de proyecto en curso con extensión .hp. Todos los nombres de rutas relativas son calculados desde ese directorio.

9.1 Archivos de proyecto generados

Cuando se crea un espacio de trabajo, se crea un directorio para el mismo. Los proyectos dentro del espacio de trabajo pueden estar en el mismo directorio o en un subdirectorio. Cuando se construye un proyecto, se crea un directorio para los resultados de esa construcción. El nombre del directorio por defecto es el nombre del tipo de construcción (Debug, Release, VHDL o EDIF).

Se puede cambiar este estableciendo los valores de **Output Directory** en la etiqueta **General** del cuadro de diálogo **Project Settings.**

9.2 Rutas de búsqueda

Los archivos de código que se han añadido al espacio de trabajo del proyecto serán compilados y construidos. Los archivos de cabecera serán localizados únicamente por el preprocesador, si es que existen, en una ruta conocida.

Los directorios buscados serán los siguientes en este mismo orden:

1. Directorios que contengan archivos Handel-C que tengan la directiva #*include* (si está entre comillas).
2. Directorios enumerados en **Project>Settings>Preprocessor> Additional include directories** (en el orden especificado).
3. Directorios listados en el cuadro **Directories** del diálogo **Tools>Options** (en el orden especificado).
4. Directorios en el entorno de variable **HANDELC_CPPFLAGS**(en el orden especificado).

10. Ventanas y barras de herramientas

El interfaz de usuario de Handel-C tiene ventanas desplazables estándar y barras de herramientas configurables.

Se puede configurar:

- La manera en que son expuestos el editor y el entorno de desarrollo (posiciones del espacio de trabajo y de las ventanas de salida).
- La manera en que las ventanas de documentos son exhibidas (esta forma es particularizable para cada espacio de trabajo).
- El esquema de depuración (la manera en que aparecen las ventanas cuando estamos en la fase de depuración).

Estos diseños son almacenados. Los esquemas de edición y construcción y los de depuración son conservados en su copia de Handel-C. Si se cambian, afectarán a todos los proyectos. El diseño de ventanas de documentos se conserva en su espacio de trabajo y puede cambiar cuantas veces se cambie el espacio de trabajo en curso.

10.1 Tipos de ventana

Las ventanas de documentos son desplazables dentro de la ventana de Handel-C. Se pueden redimensionar y arrastrar.

Las ventanas pueden ser apiladas en uno de los márgenes de la ventana de Handel-C o bien estar flotando sobre otras ventanas. Cuando una ventana está apilada carece de barra de título. Si lo que se ha apilado es una ventan del editor de código, el nombre del archivo aparece entre corchetes tras el título del proyecto en la barra de título de Handel-C.

Para hacer que aparezca flotando una ventana se ha de hacer doble click en su contorno.

Para apilar una ventana flotante o bien se hace doble click en su contorno o bien se arrastra su barra de título hacia una posición apilable.

Fraccionar ventanas

Se pueden fraccionar ventanas arrastrando el cuadro pequeño que aparece justo encima de la barra vertical de desplazamiento.

Figura 25. Cuadro para fraccionamiento de ventanas.

10.2 El menú de ventanas

El menú de ventanas permite controlar el tamaño y presentación de las ventanas de edición. Este menú tiene los siguientes comandos:

Comando	Descripción
New window	Crea una copia de la ventana actual.
Split	Divide la ventana en dos o cuatro vistas.
Docking View	Habilita/inhabilita la vista apilada de las ventanas apilables seleccionadas.
Close	Cierra la ventana actual.
Close All	Cierra todas las ventanas.
Next	Se mueve hacia el siguiente cuadro de una ventana fraccionada.
Previous	Se mueve hacia el cuadro anterior de una ventana fraccionada.
Cascade	Ordena en forma de cascada todas las ventanas abiertas dejando visibles sus barras de título.
Tile Horizontally	Exhibe todas las ventanas, dividiendo la zona de visión horizontalmente.
Tile Vertically	Exhibe todas las ventanas, dividiendo la zona de visión verticalmente.
Arrange Icons	Ordena los iconos de ventanas minimizadas en la parte baja de la zona de visión.
Windows...	Abre el diálogo de ventanas.

El diálogo de ventanas **Windows** proporciona el nombre de todas las ventanas de edición abiertas. Podemos hacer que cualquiera de ellas sea la ventana actual o seleccionar un grupo de ventanas para guardarlas, cerrarlas o embaldosarlas.

10.3 Presentación en pantalla completa

El comando **Full screen** del menú **Edit** muestra el cuadro del editor de código con el tamaño máximo. Las barras de menú normales y las barras de

herramientas no están a la vista. Para volver a la vista normal hay que pulsar el botón de fin de pantalla completa mostrado en la figura.

Figura 26. Icono de fin de pantalla completa.

10.4 Barras de herramientas

Cuando comienza Handel-C, las barras de herramientas aparecen bajo la barra de menú. Las barras que aparecen son:

Figura 27. La barra de herramientas estándar.

Figura 28. Mini-barra de construcción.

Figura 29. Mini-barra de exploración.

Figura 30. Mini-barra de depuración.

Figura 31. Mini-barra de *bookmarks*.

Cambiar las barras de herramientas

Las barras de herramientas son apilables. Pueden ser apiladas en uno de los bordes de la ventana de Handel-C o pueden estar flotando. Se puede cambiar

una barra de herramientas de su posición apilada a flotante y viceversa haciendo doble click sobre ella. Se puede mover arrastrándola pinchando en la barra de título o en la barra doble.

La barra de estado

La barra de estado es visible al fondo de la ventana de Handel-C. Muestra información sobre objetos cuando el puntero del ratón está sobre ellos.

11.El menú de herramientas

El menú de herramientas tiene el comando **Source Browser** y comandos para configurar personalmente nuestra copia de Handel-C.

11.1 El comando Source Browser

El comando **Source Browser** permite buscar nombres de variables y funciones dentro de nuestro código. Nos muestra sus definiciones así como todas las referencias a las mismas, tal como se ha explicado previamente.

11.2 Personalizando el interfaz

El comando **Customize...** hace que comience el diálogo **Customize**. La etiqueta **Toolbar** permite cambiar la apariencia de las barras de herramientas.

Activando la casilla correspondiente o desactivándola en el cuadro **Toolbars** haremos que una barra de herramientas se muestre o se oculte.

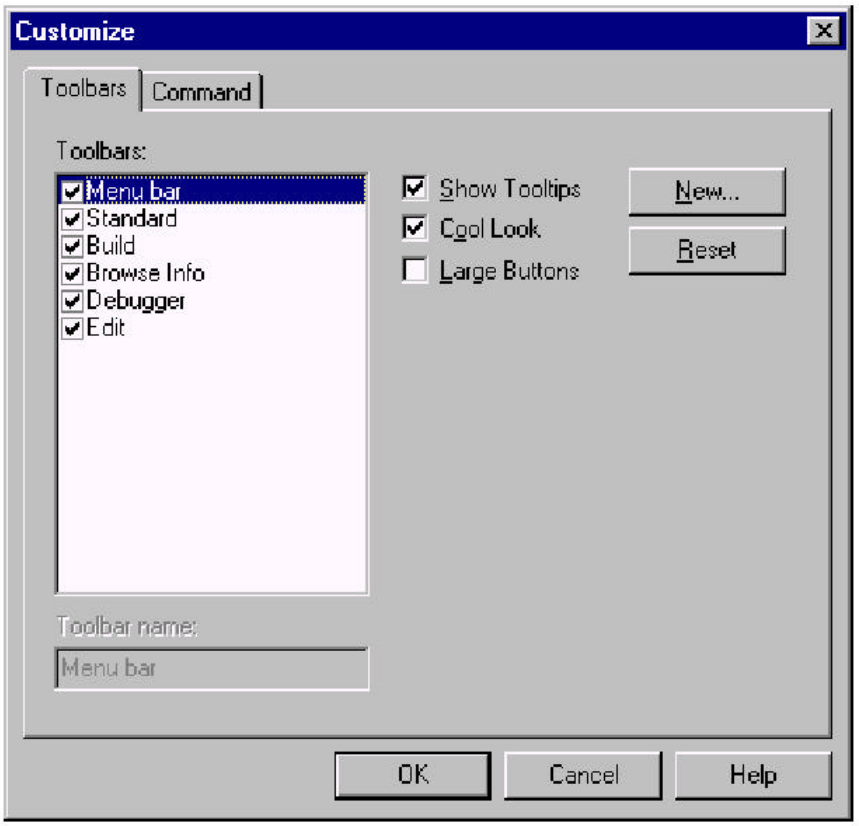


Figura 32. Cuadro de diálogo *Customize* para la personalización del interfaz.

Además tenemos las siguientes opciones:

Show Tooltips	Activar esta casilla para resaltar la acción del botón cuando el puntero del ratón está sobre el mismo.
Cool Look	Hace que los botones aparezcan en dos dimensiones.
Large Buttons	Aumenta el tamaño de los botones.
Large Icons	Incrementa el tamaño de los iconos que están sobre botones grandes.

La etiqueta **Command** permite añadir menús y botones a la barra de herramientas y a la barra menú. A la derecha del cuadro se exhiben los botones y comandos de menú disponibles (ver figura).

Seleccionamos el botón o el menú que queramos añadir y lo arrastramos hacia la barra de herramientas o la barra de menú. Si se arrastra un comando de

menú a una barra de herramientas, aparecerá como un botón. Si se arrastra hacia una zona vacía, aparecerá como una nueva ventana flotante.

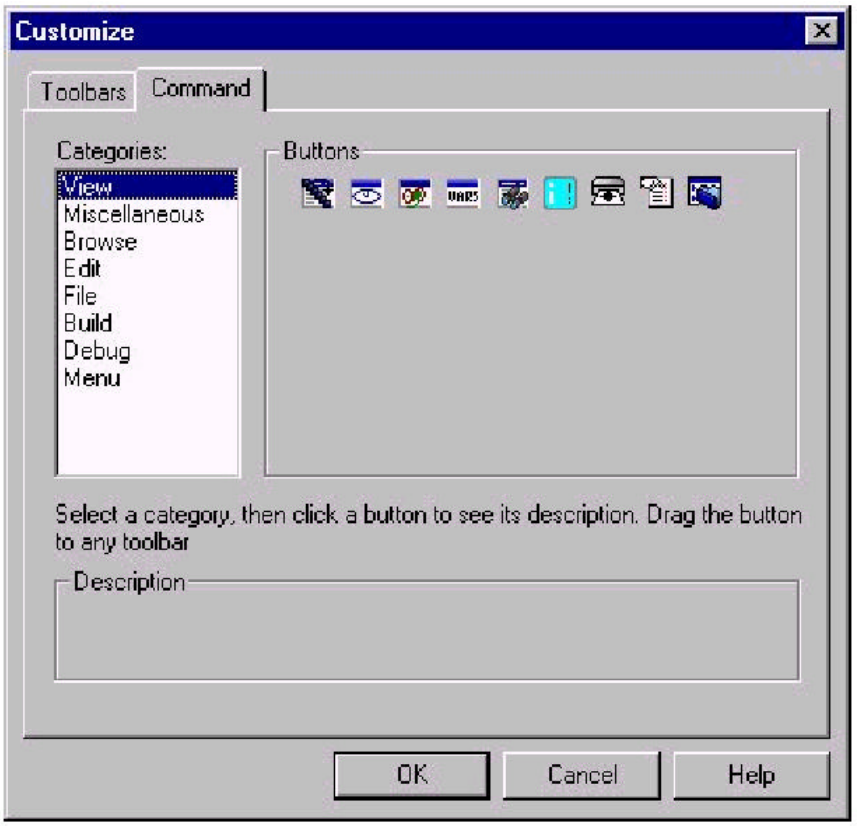


Figura 33. Cuadro de diálogo *Customize.* Personalización de comandos.

Borrado de botones y menús

Se pueden quitar los botones de una barra de herramientas abriendo el diálogo **Tools>Customise** y luego arrastrarlos hacia fuera de la barra de herramientas.

También se pueden eliminar menús de la barra de menú abriendo el diálogo **Tools>Customise** y arrastrarlos hacia fuera de la barra de herramientas.

Para recuperar el estado previo de una barra de herramientas, seleccionamos la etiqueta de la barra de herramientas del diálogo **Tools>Customise**. Seleccionamos la barra de herramientas (bajo la etiqueta **Toolbar**) o el menú (bajo la etiqueta **Commands**).

11.3 Opciones

El comando **Tools>Options** permite establecer opciones en los siguientes apartados:

Editor	Establece las opciones de ventana para el editor. Define cuándo van a ser guardados los archivos.
Tabs	Define cómo las etiquetas van a ser usadas y si se usa o no la auto-indentación.
Debug	Configura cuál va a ser la base por defecto que va a ser usada para la representación de números en las ventanas de depuración.
Format	Define el color y la fuente de texto y los marcadores en las ventanas.
Workspace	Establece el número de espacios de trabajo recientemente abiertos en la lista de espacios de trabajo.
Directories	Establece los directorios que van a ser explorados cuando hay directivas *include* y para las librerías usadas en los proyectos.

A continuación detallamos las posibilidades de configuración en los apartados indicados anteriormente.

Editor

Vertical scroll bar	Activar para mostrar la barra vertical de desplazamiento.
Horizontal scroll bar	Activar para mostrar la barra horizontal de desplazamiento.
Automatic window recycling	Muestra archivos abiertos por el IDE (*Integrated Development Environment)* en una ventana existente.
Selection margin	Se utiliza la selección de margen en la ventana del editor para permitir la selección de párrafos, etc.
Drag and drop text editing	Se puede editar seleccionando y arrastrando una zona a una nueva posición.
Save before running	Guarda archivos antes de ejecutar herramientas

tools	definidas en el menú **Tools**.
Prompt before saving files	Pregunta antes de guardar.
Automatic reload of externally modified files	Si se abre un archivo en Handel-C y se modifica en algún editor externo a Handel-C se cargan automáticamente desde el disco duro.

Tabs

File type	Define las configuraciones para tipos de archivo específicos o define las configuraciones por defecto
Tab size	Define el número de espacios equivalente por tabulador.
Insert spaces/ Keep tabs	Selecciona cuándo usar espacios o tabuladores en archivos.
Auto indent	Activar texto auto-indentado.

Debug

Base for numbers	Selecciona la base numérica de representación por defecto en las ventanas de depuración.

Format

Category	Selecciona el tipo de ventana para modificarlo.
Font	Selecciona la fuente con la que se mostrará el texto.
Size	Selecciona el tamaño de exposición de la fuente.
Colours	Selecciona el tipo de texto para modificarlo ***Foreground:*** Selecciona el color de primer plano. ***Background:*** Selecciona el color de fondo.
Sample	Exhibe texto de ejemplo en las configuraciones seleccionadas.
Reset All	Vuelve a la configuración inicial.

Workspace

Default workspace list	Selecciona el número de espacios de trabajo reciente en el comando **File>Recent Workspaces**

Directories

Show directories for:	Selecciona las listas de rutas de *include* y las rutas de librerías. Añade o quita rutas de directorios para archivos *include* o archivos librería.

12.El compilador de Handel-C

El compilador Handel-C compila y optimiza el código fuente Handel-C en un archivo apropiado para la simulación o un archivo tipo *netlist* que puede ser ubicado y rutado en una FPGA real.

El compilado es normalmente llamado automáticamente cuando el usuario selecciona una opción del menú **Build**.

Una vez que el compilador ha finalizado su labor, se muestra en la ventana de salida una estimación del número de puertas NAND estimado necesario par la implementación del diseño.

El compilador utiliza el preprocesador GNU. Los *flags* pueden pasarse al preprocesador usando la etiqueta **Preprocessor** del cuadro de diálogo **Project>Settings**.

Si se desea ejecutar el compilador desde una línea de comando, se puede hacer usando el comando **handelc**.

12.1 El proceso de construcción

Ejecutamos una construcción cuando:

Apretamos el botón de construcción.

Tenemos archivos no compilados aún y seleccionamos el comando **Start Debug** del menú **Build**.

Seleccionamos **Build** o **Rebuild All** del menú **Build**.

Figura 34. Icono de construcción del proyecto (*build*)

Esta ejecución conlleva:

- ✓ Preprocesar los archivos cabecera y compilar los archivos cabecera dependientes.
- ✓ Compilar aquellos archivos que ha sido añadidos, cambiados y guardados desde la última compilación y también compilar todos aquellos archivos que dependen de éstos.
- ✓ Enlazar los archivos compilados unos con otros.
- ✓ Calcular el número de puertas a utilizar.
- ✓ Construir una tabla de símbolos.
- ✓ Generar un archivo simulable o uno tipo *netlist*.

Si se cambia la configuración para un proyecto determinado, será necesario compilar todos los archivos. Seleccionando el comando **Build>Rebuild All** aseguramos que todos los archivos son recompilados.

Los resultados de la compilación y construcción son mostrados en la ventana de construcción. Haciendo doble click sobre un error el editor de código nos llevará a la línea adecuada en el archivo fuente.

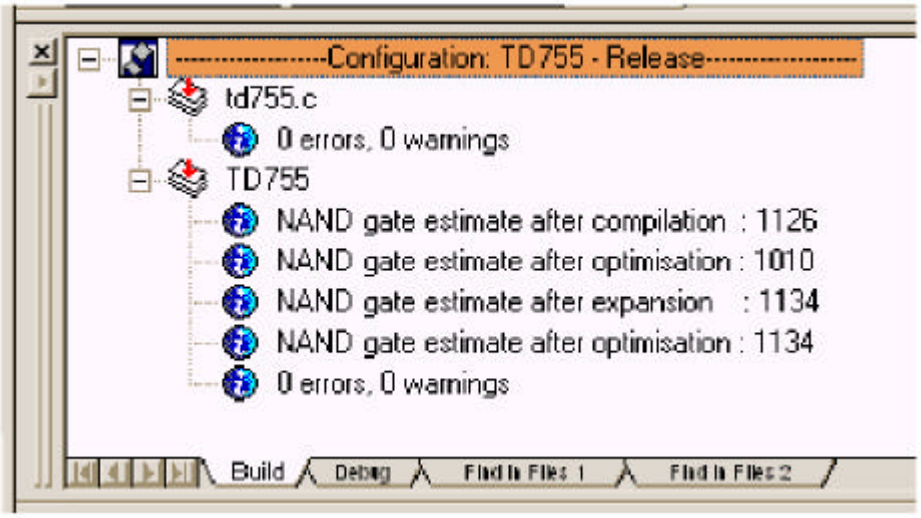


Figura 35. Ventana de salida tras la compilación de un proyecto.

12.2 Comprobación de la profundidad y la velocidad del código

Se puede examinar la profundidad y la velocidad del código compilándolo con la opción **-e.** Esto genera:

- Un archivo HTML para el proyecto, *proyecto*.**html**
- Un archivo HTML para cada archivo del proyecto*archivos*_c.**html**. Estos archivos resaltarán las zonas de código en función del área o retraso que necesiten para su implementación.

Se pueden examinar estos archivos abriéndolos en el navegador de Internet.

proyecto.html

El archivo *proyecto*.**html** tiene enlaces a todos los archivos *archivos*_c.**html** que resaltan el código fuente. También tiene enlaces a las 5 mayores áreas y a los 5 mayores retrasos del proyecto.

FILE: Release\Prueba.dll

Source files:

- C:\Archivos de programa\Celoxica\DK1 Evaluation\Examples\Handel-C\ExampleC\Edge_v4\edge_v4.c

Top Areas:

- C:\Archivos de programa\Celoxica\DK1 Evaluation\Examples\Handel-C\ExampleC\Edge_v4\edge_v4.c (1048869 nand gates)
- C:\Archivos de programa\Celoxica\DK1 Evaluation\Examples\Handel-C\ExampleC\Edge_v4\edge_v4.c (1048771 nand gates)
- C:\Archivos de programa\Celoxica\DK1 Evaluation\Examples\Handel-C\ExampleC\Edge_v4\edge_v4.c (8363 nand gates)
- C:\Archivos de programa\Celoxica\DK1 Evaluation\Examples\Handel-C\ExampleC\Edge_v4\edge_v4.c (6407 nand gates)
- C:\Archivos de programa\Celoxica\DK1 Evaluation\Examples\Handel-C\ExampleC\Edge_v4\edge_v4.c (1 nand gates)

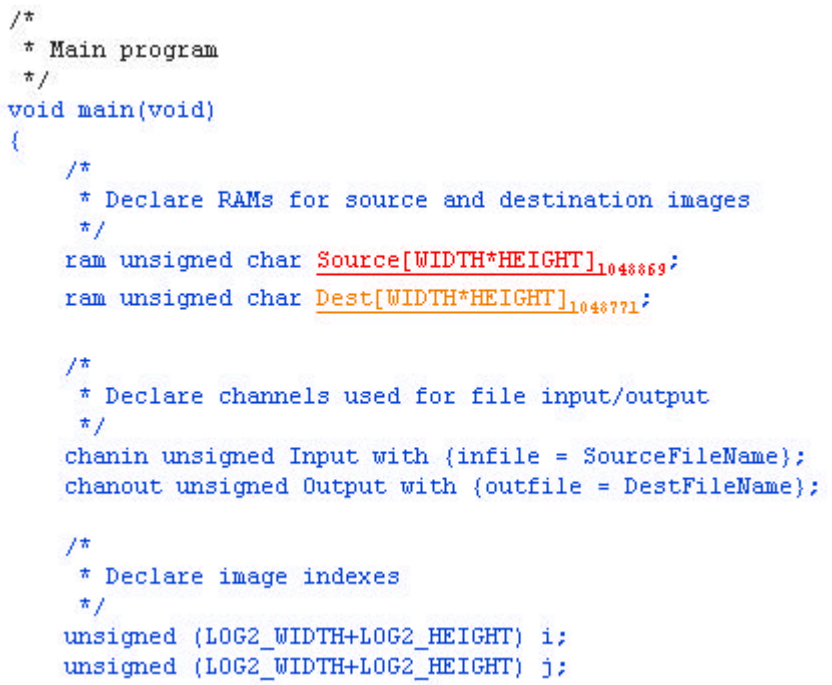
Top delays:

- C:\Archivos de programa\Celoxica\DK1 Evaluation\Examples\Handel-C\ExampleC\Edge_v4\edge_v4.c (38 gate delays)
- C:\Archivos de programa\Celoxica\DK1 Evaluation\Examples\Handel-C\ExampleC\Edge_v4\edge_v4.c (37 gate delays)
- C:\Archivos de programa\Celoxica\DK1 Evaluation\Examples\Handel-C\ExampleC\Edge_v4\edge_v4.c (36 gate delays)
- C:\Archivos de programa\Celoxica\DK1 Evaluation\Examples\Handel-C\ExampleC\Edge_v4\edge_v4.c (35 gate delays)
- C:\Archivos de programa\Celoxica\DK1 Evaluation\Examples\Handel-C\ExampleC\Edge_v4\edge_v4.c (6 gate delays)

Figura 36. Archivo HTML del proyecto Prueba.hw

archivo_c.html

La versión html del archivo fuente muestra dos versiones del código fuente. La primera tiene varios colores de acuerdo con el área precisada para la implementación del código.



```
/*
 * Main program
 */
void main(void)
{
    /*
     * Declare RAMs for source and destination images
     */
    ram unsigned char Source[WIDTH*HEIGHT]1048869;
    ram unsigned char Dest[WIDTH*HEIGHT]1048771;

    /*
     * Declare channels used for file input/output
     */
    chanin unsigned Input with {infile = SourceFileName};
    chanout unsigned Output with {outfile = DestFileName};

    /*
     * Declare image indexes
     */
    unsigned (LOG2_WIDTH+LOG2_HEIGHT) i;
    unsigned (LOG2_WIDTH+LOG2_HEIGHT) j;
```

Figura 37. Archivo edge v4 c.HTML, visión del archivo en función del área precisada.

La segunda, de acuerdo con la cantidad de retardo generado. Colores fríos (azules y verdes) indican un área o retraso pequeño; los colores cálidos (rojos y amarillos) muestran que se están consumiendo una gran cantidad de área o retardo.

```
/*
 * Define parameters of image and thresold for edges
 */
#define LOG2_WIDTH 8
#define WIDTH 256
#define LOG2_HEIGHT 8
#define HEIGHT 256
#define THRESHOLD 16


/*
 * Edge detector procedure
 */
void edge_detect(ram unsigned char (*Source)[WIDTH*HEIGHT],
                 ram unsigned char (*Dest)[WIDTH*HEIGHT])
{
    /*
     * Declare RAM access macros
     */
    macro expr ReadRAM(a, b) =
                ((unsigned 1)0) @
                    (*Source)[(0@a) + ((0@b) << 8)];
    macro proc WriteRAM(a, b, c)
                (*Dest)[(0@a) + ((0@b)<<8)] = c;
```

Figura 38. Archivo edge v4 c.HTML, visión del archivo en función del retardo generado.

Hay tablas completas de colores al final de cada sección.

Highlight Colours Table:

=	0.00
<	104886.89
<	209773.78
<	314660.67
<	419547.56
<	524434.45
<	629321.33
<	734208.22
<	839095.11
<	943982.00
<	1048868.89
=	1048868.88

Figura 39. Tabla de colores en función de las puertas generadas.

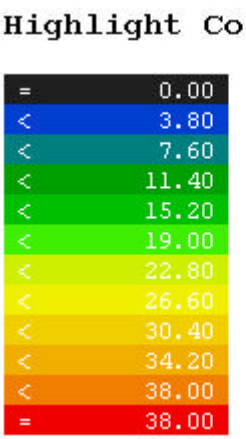


Figura 40. Tabla de colores en función de los retardos generados.

Los cinco retardos y las cinco áreas mayores están subrayados y etiquetados con el número de puertas o número de niveles lógicos generados. Estas estimaciones son solamente una guía porque se necesita un proceso exhaustivo de ubicación y rutado (*place&route*) para obtener información exacta del área lógica y el retardo generados.

12.3 El menú de construcción

Comando	Teclado abreviado	Función
Compile	Ctrl+F7	Ejecuta el compilado en el documento activo (que debe tener la extensión **.c**), para generar su correspondiente archivo **.obj**.
Build project	F7	Construye el proyecto: Ejecuta el compilador en todos los archivos **.c** que son más recientes que sus archivos **.obj** y después ejecuta el enlazador sobre los archivos **.obj** para crear el archivo **.dll**, EDIF o VHDL.
Rebuild All		Reconstruye todos los archivos del proyecto en curso. Es igual que *Build* salvo que todos los archivos **.c** son compilados sin

		excepción.
Clean		Borra todos los archivos que son generados con el comando *Build.*
Start Debug		Menú de saltos.
Go	F5	(Construye el proyecto si no está construído aún). Ejecuta el simulador a velocidad máxima (hasta que alcanza un punto de ruptura u otra causa de parada).
Step Into	F11	(Construye el proyecto si no está construído aún). Ejecuta el código hasta la primera sentencia en la función o macro llamada en la línea en curso. Si la línea en curso no es un función o una macro, se ejecuta hasta la próxima sentencia.
Run to Cursor	Ctrl+F1	(Construye el proyecto si no está construído aún). Ejecuta hasta la línea que contiene el siguiente cursor.
Set Active Configuration		Muestra un cuadro de diálogo donde el usuario puede elegir la configuración activa.
Configurations...		Muestra un cuadro de diálogo donde el usuario puede añadir, quitar o editar configuraciones.

13.El simulador y el depurador de Handel-C

El simulador permite comprobar el programa sin usar *hardware* real. Permite ver el estado de cada variable (registros) en el programa a cada ciclo de reloj. Se puede seleccionar qué variables van a ser exhibidas usando las ventanas de observación (**Watch)** y de variables (**Variable).** Se pueden ver los hilos en curso ejecutando la ventana de hilos (**Threads)** y los relojes actuales usados en la ventana de relojes (**Clocks)**. Por último, podemos observar una determinada función y lo que la función necesita llamar para poder ejecutarse en la ventana de pila de llamadas (**Call Stack)**.

Se puede ejecutar el código en el simulador de varias formas:

Ejecutar hasta el final (nunca finaliza en un bucle infinito de programa).

Ejecutar hasta que llega a la posición actual del cursor.

Ejecutar hasta que alcanza los puntos de ruptura definidos por el usuario.

Ejecución paso a paso.

Cuando se usa el depurador de código se puede estar ejecutando la simulación (*run mode*) o pausar la simulación (*break mode*). Cuando la simulación está pausada (de alguna de las maneras explicadas anteriormente o usando el comando **Break**) se pueden examinar fácilmente las variables, cambiar la presentación de las ventanas o establecer puntos de ruptura. Cuando la simulación está en modo ejecución sólo podemos observar.

En el momento en que usamos el depurador de código aparece el menú **Debug**. Este menú tiene los siguientes comandos.

Comando	Teclado abreviado	Función
Go	*F5*	Ejecuta el simulador a velocidad completa (hasta un punto de ruptura u otra parada).
Restart	*Ctrl+Shift+F5*	Ejecuta el simulador comenzando en la primera línea de programa.
Stop Debugging	*Shift+F5*	Detiene la simulación.
Break		Pausa la simulación (cuando se está ejecutando).
Show Next Statement	*ALT+(Tecl. num)**	Muestra la sentencia en el hilo en curso que será ejecutada en el siguiente ciclo de reloj.
Step Into	*F11*	Ejecuta hasta la primera sentencia en la función llamada en la línea actual. Si la línea actual no llama a ninguna función, ejecuta hasta la siguiente sentencia en un ciclo de reloj completo.
Step Over	*F10*	Ejecuta hasta el comienzo de la siguiente sentencia (salta una función).
Step Out	*Shift+F11*	Ejecuta hasta el comienzo de la sentencia posterior a la línea que llama a la función en curso (sale de una función).
Run to Cursor	*Ctrl+F10*	Ejecuta hasta que llega a la línea que contiene el siguiente cursor.
Advance	*Ctrl+F11*	Avanza dentro de un mismo ciclo de reloj línea a línea de código.

También se pueden establecer puntos de ruptura en líneas de código válidas. Cuando el depurador de código alcanza un punto de ruptura se detendrá hasta que el usuario le solicite la reanudación.

14.El interfaz del depurador de código

El interfaz del depurador de código posee las siguientes ventanas:

Ventana	Teclado abreviado	Función
Ventana del editor	Aparece por defecto	Es la ventana del editor de código del código fuente que se está depurando, aparece marcada con los puntos de ejecución actuales y los puntos de ruptura.
Watch	Alt+F9	Es una ventana con cuatro etiquetas, en las que se puede ver el contenido de las variables seleccionadas.
Call Stack	Alt+7	Es una ventana que muestra la ruta de llamada a la función que se está ocupando.
Variables	Alt+4	Ventana de variables.
Clocks	Alt+9	Muestra todos los relojes en curso y sus valores.
Threads	Alt+5	Muestra todos los hilos en curso con un único identificador y permite seleccionar uno para su seguimiento.

14.1 Símbolos en la ventana del editor

Las sentencia asociadas con el instante actual de reloj son marcadas con punteros. Todas esas sentencias se ejecutan juntas.

Si hay una sentencia **par** en el código, la ejecución será segmentada en varios hilos, uno para cada rama de la sentencia **par**. Los hilos se ejecutan en paralelo. Cuando se está haciendo un proceso de depuración de código sólo se puede observar un hilo en un instante determinado.

El hilo en curso tiene punteros amarillos y blancos. Los punteros blancos muestran código combinacional que será ejecutado en el siguiente ciclo de reloj. Los amarillos indican código que se está ejecutando en ese mismo momento.

Los otros hilos tienen punteros grises que indican código que se está ejecutando en ese mismo instante.

Con la ejecución paso a paso podremos observar la evolución de esos punteros.

La ventana de variables

La ventana **Variables** siempre muestra las variables actuales. Cuando sus valores cambian, el color cambia de negro a rojo. La ventana tiene dos etiquetas, **Auto** y **Locals**.

La etiqueta **Auto** muestra las variables que han sido seleccionadas automáticamente. Son variables utilizadas en la sentencias previa y en curso en el hilo en curso. También muestra los valores devueltos por una función cuando la saltamos.

La etiqueta **Locals** muestra las variables que son locales para la función o la macro en curso.

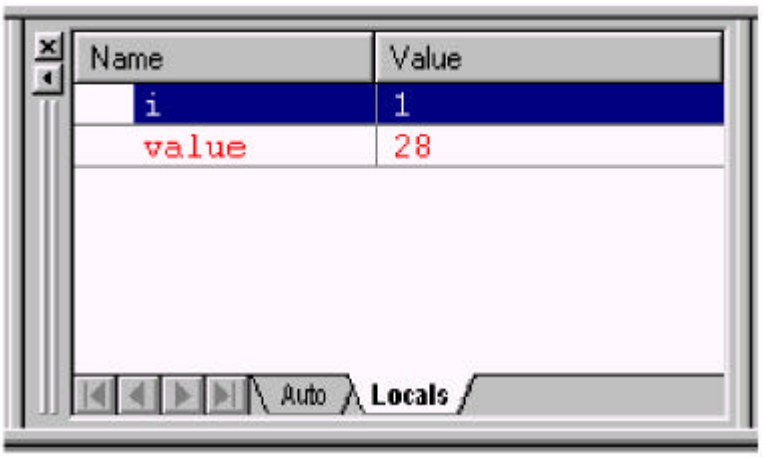


Figura 41. Ventana de variables locales.

14.2 La ventana de observación

Existen cuatro ventanas de observación. Se pueden seleccionar variables para ser exhibidas en cada ventana, y observar sus valores en cada punto de ruptura o en cada paso a medida que se ejecuta el programa.

Se puede añadir una variable a la ventana de observación tecleando su nombre en la ventana de observación.

La ventana de observación tiene un evaluador de expresiones. Si se le teclea una expresión, el resultado será evaluado.

14.3 La ventana de pila de llamadas

Las funciones llamadas en la función en curso son mostradas en la ventana de pila de llamadas **Call Stack**. Esta ventana muestra la función actual en la parte superior de la misma y, debajo, las funciones que no se han completado aún.

La función en curso en el hilo en desarrollo se marca con un puntero amarillo. Si hay varios hilos que están ejecutando distintas funciones, las otras funciones en desarrollo se marcan con punteros grises.

En la figura podemos ver la función en curso **blib()** y la ventana de pila de llamadas correspondientes.

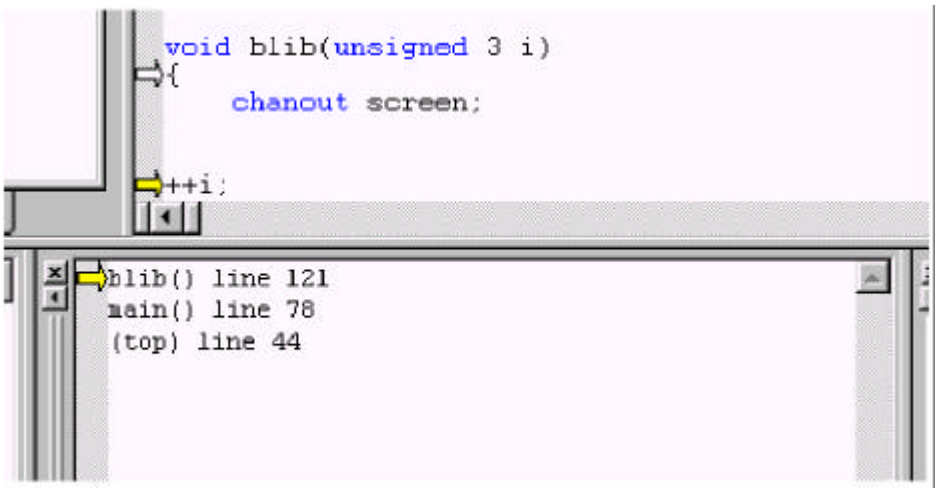


Figura 42. Ventana de pila de llamadas.

14.4 La ventana de hilos

Todos los hilos son mostrados en la ventana de hilos. Ver figura.

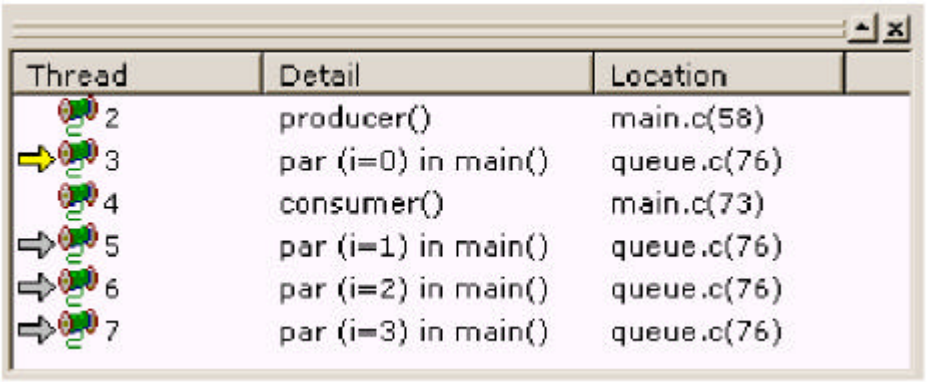


Figura 43. Ventana de hilos.

La columna con el indicativo **Thread** muestra los hilos que se están ejecutando en el programa.

El puntero amarillo (hilo nº 3) indica el hilo actual.

Los punteros grises indican hilos que se están ejecutando en el mismo ciclo de reloj que el hilo actual.

La columna **Detail** proporciona una reseña de la procedencia de este hilo.

La columna **Location** da información del número de línea en el programa de ese hilo.

Con el botón derecho del ratón podemos desplegar un menú de la ventana de hilos que muestra los comandos:

Comando	Función
Show Location	Muestra el archivo fuente y desplaza a la derecha.
Follow	Hace que el depurador de código siga ese hilo (lo convierte en el hilo actual.

14.5 La ventana de relojes

Todos los relojes utilizados se muestran en la ventana de relojes **Clocks**. El reloj en curso es marcado con un puntero amarillo. Se identifica con el nombre completo de la ruta del archivo que lo referencia.

Figura 44. Ventana de relojes.

También se muestra la cuenta del número de ciclos de reloj ya ejecutados en la columna **Cycles**. Haciendo doble-click en un reloj podemos ver su definición.

15. Configuración

En modo depuración, la configuración del proyecto está establecida por defecto. La configuración que aparece es la siguiente:

Herramienta	Configuración
Preprocesador	Se definen las varibles **DEBUG** y **SIMULATE**. Esto nos permite configurar el código en función de si se va a usar el simulador o se van a usar interfaces *hardware* reales.
Compilador	Se activan las casillas de **Generate Debug** y **Generate warning**.
Enlazador	**Output format** puesta como **Simulator**. Activada la casilla **Save browse info**. Desactivada la opción **Generate estimation information** (generación de archivos html).
Depurador	El directorio de trabajo **Working directory** para el depurador es el actual (.)
Optimizaciones	Activada la optimización a alto de alto nivel.

NUEVOS ASPECTOS DE LA VERSIÓN 3.0 DE HANDEL-C

Capítulo III. NUEVOS ASPECTOS DE LA VERSIÓN 3.0 BETA DE HANDEL-C

En este capítulo se recogen todos aquellos aspectos nuevos del lenguaje Handel-C que han sido incorporados en la versión 3.0 Beta de Embedded Solutions Ltd. respecto de la versión anterior, la 2.1 (y que también están presentes en la versión DK1 de Celoxica), así como aquellos que han resultado modificados o ampliados.

1. Nuevos operadores

1.1 El operador de indirección (*) y " dirección de" (&)

Los punteros en Handel-C son similares a los de C convencional. Proporcionan la dirección de una variable o de un trozo de código. Esto permite acceder a una variable por referencia en vez de por valor.

El operador de indirección (*) es igual que en ISO-C. Se utiliza para declarar punteros a objetos y para redireccionar a otros punteros.

El operador "dirección de" (&) también es igual que en ISO-C (aunque técnicamente las variables Handel-C no son habitualmente almacenadas en posiciones de memoria que necesiten ser direccionadas).

2. Operadores modificados o ampliados

2.1 Operador de selección de bit ([...])

Se pueden seleccionar bits individuales o cadenas de bits de un valor usando el operador **[...]**. El bit 0 es el menos significativo y el bit n-1 es el más significativo en donde n es la anchura en bits el valor. Por ejemplo:

```
unsigned int 8 x;
unsigned int 1 y;
unsigned int 5 z;

x = 0b01001001;
y = x[4];
z= x[7:3];
```

Esto hará que la variable y valga 0 (como el bit nº 4 de la variable x) y que la variable z valga 9 (como la cadena 01001 de la variable x). También es posible la selección de bits en RAMs, ROMs y elementos *array*. Por ejemplo:

```
ram int 7 w[23];
int 5 x[4];
int 3 y;
unsigned int 1 z;

y = w[10][4:2];
z = x[2][0];
```

En la asignación a la variable *y* el número 10 selecciona la undécima entrada a la ram *w* de 23 elementos y la selección 4:2 son 3 bits intermedios (de los 7 que consta) de ese valor que se almacenarán en la variable *y*. Análogamente, en la asignación a la variable *z* seleccionamos el bit 0 de la tercera entrada del *array* x.

2.2 Operador división (/)

La utilización del operador división se ha extendido en la versión 3.0 Beta al uso del mismo con variables. Su utilización no se recomienda debido al gran consumo de puertas lógicas que requiere. En su lugar podemos construir una función que se base en el desplazamiento de bits para ahorrar *hardware* en la implementación.

2.3 Operador módulo (%)

Al igual que con el operador división (/), el uso del operador módulo (%) se ha ampliado al manejo de variables.

2.4 Operadores de desplazamiento

El operador de desplazamiento a la izda. (<<) y el de desplazamiento a la drcha. (>>) pueden desplazar un número de bits indicado por el valor de una variable. Esta es la extensión de se uso que se ha hecho en la versión 3.0 Beta.

3. Nuevas declaraciones

*3.1 Variable **signed***

El tipo lógico básico es el tipo **int**. Este tipo puede ser calificado como **signed** (con signo) o **unsigned** (sin signo).

*3.2 Tipo **enum***

El tipo **enum** especifica una lista de valores enteros constantes, por ejemplo:

```
enum weekdays {MON, TUES, WED, THURS, FRI};
```

El primer nombre (en este caso MON) tiene el valor 0, el siguiente 1, y así sucesivamente, a menos que se especifiquen valores explícitamente. Si no se especifican todos los valores éstos se incrementan desde el último valor especificado.

Para especificar valores **enum** debemos hacerlo como en el ejemplo:

```
enum weekdays {MON = 9, TUES, WED, THURS, FRI};
```

En la versión beta no se pueden declarar variables tipo enum, (por ejemplo, `enum weekdays x;` no está permitido). Se pueden asignar valores **enum** a variables (por ejemplo `int x = MON;`).

*3.3 Variable **struct***

El tipo **struct** define una estructura de datos, un grupo de variables unidas bajo un mismo nombre. El formato de la estructura puede ser identificado por el nombre del tipo. Las variables miembro de la estructura pueden ser del mismo tipo o de tipos diferentes. Una vez declarada una estructura, su nombre de tipo puede ser utilizado para definir otras estructuras del mismo tipo. Los miembros de la estructura pueden ser accesibles usando la construcción:

```
nombre_de_estructura.nombre_de_miembro
```

La sintaxis de este tipo es la siguiente:

```
struct[nombre_de_tipo]
{
      lista_de_miembros;
}[nombres_de_instancias];
```

En donde:

lista_de_miembros es una lista de definiciones de variables terminadas por punto y coma.

nombres_de_instancias declara variables con el mismo tipo de estructura.

Alternativamente podemos definir variables **struct** como:

```
struct nombre_de_tipo nombres_de_instancias;
```

Ejemplo:

```
struct human          //Declara el tipo de estructura human
{
 unsigned int 8 age;//Declaración de los miembros
```

```
 int 1 sex;
 char name[25];
};                  //Define el tipo human

struct human sister;
sister.age = 25;
```

*3.4 Objeto **signal***

Una señal **signal** es un objeto que toma el valor que se le asigna durante un único ciclo de reloj. El valor asignado a esta señal puede ser leído en el mismo ciclo de reloj en que es asignado. Durante el resto del tiempo la señal toma su valor de inicialización. El valor de inicialización por defecto es 0. El operador clarificador <> puede ser usado para aclarar definiciones de señales complejas.

La sintaxis es la siguiente:

```
signal [<tipo ancho_dato>] nombre_de_señal;
```

Ejemplo:

```
int 15 a, b;
signal <int> sig;

a = 7;
par
{
      sig = a;
      b = sig;
}
```

sig es asignada y leída en el mismo ciclo de reloj, de manera que *b* toma el valor de *a*. Como la señal solamente mantiene el valor asignado un ciclo de

reloj, si se lee un ciclo antes o después de su asignación obtendremos su valor de inicialización.

*3.5 Variable **const***

El tipo **const** define una variable o *array* de variables que no pueden ser asignadas. Esto quiere decir que mantienen su valor de inicialización siempre. Pueden ser inicializadas en la sentencia de su declaración. La declaración tipo **const** puede ser usado en vez de **#define** para definir constantes.

Ejemplo:

```
const int i = 5;
i = 10;          // Error.
i++;             // Error.
```

*3.6 Parámetro **volatile***

En ISO-C, **volatile** se usa para declarar variables que pueden ser modificadas por otra cosa que no es el mismo programa.

Se usa principalmente para registros cableados. **volatile** controla la optimización forzando una relectura de la variable. El valor inicial de **volatile** está indefinido.

Handel-C no hace nada cuando lee la etiqueta **volatile**. Se utiliza por motivos de compatibilidad.

3.7 Clarificador de tipo (<>)

< > es una extensión de Handel-C para aclarar declaraciones complejas de tipos de arquitecturas. No se puede usar en tipos lógicos. Es una buena

práctica usarlos allí donde se utilicen canales, memorias o señales, para aclarar el formato de dato transferido o almacenado en esas variables.

Ejemplo:

```
struct fishtank
{
      int 4 koi;
      int 8 carp;
      int 2 guppy;
} bowl;

signal <struct fishtank> drip;
chan <int 8 (*runwater)()> tap;
```

Debe ser usado para aclarar la declaración: `chan int *x;`
¿Puntero a canal o canal de punteros?

En su lugar pondremos: `chan <int *> x;` //canal de punteros
o `chan <int> *x;` //puntero a canal

*3.8 Variable **auto***

auto define una variable automática local. En Handel-C, todas las variables locales se definen por defecto como **auto**. No se puede inicializar una variable **auto**, pero se le puede asignar un valor. El estado de inicialización de una variable **auto** es indefinido.

Ejemplo:

```
auto pig;
pig = 15;
```

*3.9 Variable **extern***

extern declara una variable que puede ser accedida por nombre desde cualquier función. Las variables externas deben ser definidas una vez fuera de toda función. (Por defecto, cualquier variable definida fuera de toda función se considerará como **extern**).

Si la variable va a ser usada en varios archivos fuente, es una buena práctica agrupar todas las declaraciones externas en un archivo cabecera incluido al comienzo de cada archivo fuente usando la directiva **#include *nombre_archivo_cabecera***.

No se puede acceder a una variable (aunque esté definida como **extern**) desde diferentes dominios de reloj.

Ejemplo:

```
extern int 16 global_fish;
int global_frog = 1234;

main()
{
      global_fish = global_frog;
      ...
}
```

Sintaxis:

```
extern declaraciónVariable;
nombreFunción (lista_de_parámetros)
```

*3.10 Funciones **inline***

inline hace que una función sea expandida cada vez que sea llamada. Se generará *hardware* cada vez que se llame a la función **inline**. Esto asegura que la función no es accedida al mismo tiempo por brazos paralelos de código.

Por defecto las funciones se asumen como **shared** (no **inline**).

Sintaxis:

```
inline declaración_de_función;
```

*3.11 Variable **register***

register ha sido implementada por razones de compatibilidad con ISO-C. **register** define una variable que tiene ámbito local. Su valor inicial es indefinido.

Ejemplo:

```
register int 16 fish;
fish = 0xffff;
```

*3.12 Variable **static***

Con la declaración de **static** damos a una variable almacenamiento permanente (su valor es conservado en todo momento). Esto asegura que el valor de una variable se conserva en las llamadas a funciones. También afecta al ámbito de una variable o función. Las funciones y las variables declaradas fuera de toda función como **static** sólo pueden ser usadas en el archivo en donde aparecen. Las variables **static** declaradas dentro de una función **inline** o en un *array* de funciones sólo pueden ser usadas en la copia de la función en la que aparecen.

Las variables **static** son las únicas variables locales (excluyendo las tipo **const**) que pueden ser inicializadas.

Ejemplo:

```
static int 16 local_function (int water, int
weed)
static int 16 local_fish = 1234;
```

```
main ()
{
      int fresh, pondweed;
      local_fish   =   local_function   (fresh,
pondweed);
      ...
}
```

Sintaxis:

```
static declaración_de_variable;
static nombre_de_función (lista_de_parámetros)
```

*3.13 Definidor de tipo **typedef***

typedef define otro nombre para un tipo de variable. Esto permite aclarar el código. El nuevo nombre es sinónimo del tipo de variable.

Ejemplo:

```
typedef int 4 SMALL_FISH;
```

Si se usa **typedef** en varios archivos fuente, es una buena práctica agrupar todas las definiciones de tipo en un archivo cabecera, incluido al comienzo de cada archivo fuente usando la directiva #**include** *nombre_archivo_cabecera*. Esto es usual para diferenciar nombres **typedef** de nombres de variables estándar, de manera que sean fácilmente reconocibles.

Ejemplo:

```
typedef int 4 SMALL_FISH;
extern SMALL_FISH stickleback;
```

*3.14 Retornos y listas de parámetros tipo **void***

Las funciones son similares a las de ISO-C. Handel-C ha extendido su uso para proporcionar *arrays* de funciones y funciones **inline**. Los *arrays* de funciones facilitan varias copias de una determinada función. Se puede seleccionar la copia que se está usando en un determinado momento. Las funciones **inline** se parecen a las macros en que son expandidas cada vez que son usadas.

Las funciones toman argumentos y devuelven valores. Una función que no devuelve ningún valor es del tipo **void**. El tipo de devolución por defecto es **int undefined**.

Cuando una función es declarada o definida hay que especificar su lista de parámetros. Esta lista describe el tipo de argumentos que esta función espera recibir. Las funciones que no esperan ningún argumento tienen su lista de parámetros declarada como **void**.

Ejemplo:

```
void main(void)
```

*3.15 Declaración **mpram***

Se puede crear una **mpram** o **RAM multi-puerto** construyendo algo parecido a una *union* ISO-C. Las **mprams** se pueden usar para conectar dos bloques de código independientes. El reloj del puerto **mpram** se toma de la función en que se utiliza. La declaración normal de una **mpram** sería para crear una RAM de doble puerto declarando dos puertos de igual anchura: para Altera un puerto debería ser de sólo lectura y otro de sólo escritura, para Xilinx 4000 un puerto debería ser de lectura-escritura y otro de sólo lectura y para Virtex los dos puertos deberían ser de lectura-escritura.

Sintaxis:

```
mpram nombre_de_MPRAM
```

```
{
      tipo _RAM tipo _var nombre_RAM[anchura];
      tipo _RAM tipo _var nombre_RAM[anchura];
}
```

*3.16 Declaración **wom***

Se puede declarar una memoria de sólo escritura usando la palabra **wom**. El único uso de una memoria de sólo escritura sería para declarar un elemento dentro de una RAM multi-puerto. Como las **woms** sólo existen dentro de rams multi-puerto, es ilegal declarar una fuera de la declaración de una **mpram**.

Sintaxis:

```
wom tipo_var tamaño_var nombre_wom [anchura] = valor_inicialización [with {especificaciones}];
```

Ejemplo:

```
mpram connect
{
 wom <unsigned 8> Writeonly[256]; //Puerto sólo-escritura
 rom <unsigned 8> Read[256];    //Puerto sólo-lectura
};
```

4. Declaraciones modificadas o ampliadas

*4.1 Declaración **interface***

Con la versión 3.0 Beta se pueden declarar tipos de *interfaces* que van a ser conexiones con objetos que no son propios de Handel-C:

Código objeto autóctono para el PC usado en la simulación.

Los programas que se ejecutan el PC para la simulación y se conectan a **interfaces** Handel-C son conocidos como *plugins*. Hay especificaciones

especiales de puertos que permiten conectar *interfaces* definidas por usuario con un *plugin* para simulación. Estos son: **extlib, extfunc, extpath** y **extinst**.

Descripciones *hardware* escritas en otro lenguaje.

Actualmente sólo se usan VHDL y EDIF. Para una *interface* con código VHDL, la clase de *interface* debería ser la misma que la entidad VHDL.

*4.2 Declaración **ram***

En la versión 3.0 Beta de Handel-C se ha ampliado la definición de **ram** a bloques de memoria específicos de Xilinx.

*4.3 Declaración **rom***

En la versión 3.0 de Handel-C se ha ampliado la definición de **ram** a bloques de memoria específicos de Xilinx.

5. Nuevas sentencias

*5.1 Sentencia **continue***

continue se va directo a la siguiente iteración de un bucle **for, while** o **do**. Para **do** o **while**, esto quiere decir que el test es ejecutado inmediatamente. En una sentencia **for** se ejecuta el paso incremental. Esto permite evitar sentencias **if...else** profundamente anidadas dentro de lazos.

Ejemplo:

```
for (i=100; i>0; i--)
{
      x = f(i);
      if (x == 1)
           continue;
      else
           y += x*x;
}
```

ⓘ No se pueden usar **continue** para saltar dentro o fuera de bloques **par**.

*5.2 Sentencia **goto***

Las etiquetas **goto** se van directamente a la sentencia especificada por la etiqueta. La etiqueta tiene el mismo formato que el nombre de una variable y debe estar en la misma función que el **goto**. Las etiquetas tienen el mismo ámbito que la función. Formalmente, **goto** nunca es necesario. Podría ser útil para salirse de etiquetas de código profundamente anidadas en caso de error.

Ejemplo:

```
for (... )
{
      for(... )
      {
            if (disaster)
                  goto Error;
            else
                  continue;
      }
}

Error:
      output ! error_code;
```

ⓘ No se pueden usar **goto** para saltar dentro o fuera de bloques **par**.

*5.3 Sentencia **return***

La sentencia **return** es utilizada para volver de una función a su llamada. **return** termina la función y devuelve el control a la función llamante. La ejecución se reanuda en la línea inmediatamente posterior a la llamada de la función. **return** puede devolver un valor a la función llamante. El valor devuelto es del tipo declarado en la declaración de la función. Las funciones que no devuelven ningún valor deberían ser declaradas como tipo **void**.

Ejemplo:

```
int power(int base, int n)
{
      int i, p;

      p = 1;
      for (i=1; i<=n; i++)
           p=p*base;
      return(p);
}
```

ⓘ No se puede usar **return** para salir de bloques **par**.

*5.4 Sentencia **typeof***

El tipo de operador **typeof** permite determinar el tipo de un objeto en tiempo de compilación. El argumento de **typeof** debe ser una expresión. Al usar **typeof** nos aseguramos que las variables ligadas mantienen su relación. Esto hace fácil modificar código, simplificando el proceso de evitar conflictos de tipo y anchura de los datos.

Una construcción del tipo **typeof** puede ser utilizada en cualquier sitio en donde se pueda utilizar el nombre de algún tipo. Por ejemplo, se puede utilizar en declaraciones, en *casts* o dentro de **typeof**.

Sintaxis:

```
typeof (expresión)
```

Ejemplo:

```
unsigned 9 ch;
typeof(ch @ ch) q;
struct
{
      typeof(ch) cha, chb;
}s1;
typeof(s1) s2;

ch = s1.cha + s2.chb;
q = s1.chb @ s2.cha;
```

Si el ancho de la variable *ch* se cambia en este ejemplo no sería necesario modificar todo el código.

También esto es útil para pasar parámetros a **macro proc**. El ejemplo de abajo muestra cómo usar la definición de **typeof** para tratar con varios tipos de parámetros.

```
macro proc swap (a, b)
{
      typeof(a) t;
      t=a;
      a=b;
      b=t;
}
```

*5.5 Sentencia **assert***

assert permite generar mensajes en tiempo de compilación si se cumple una condición. Puede ser usada para comprobar constantes en tiempo de compilación y ayuda a la protección de posibles alteraciones problemáticas de código. El usuario usa una expresión para comprobar el valor de una constante en tiempo de compilación y, si la expresión evaluada resulta ser falsa, se manda un mensaje de error a el canal de error estándar en el formato:

filename:(número de línea):(número de columna)::Asertion failed: *cadena_de_error_definida_por_usuario*

Sintaxis:

`assert (`*`condición, [cadena con especificaciones de formato, {argumentos}]`*`);`

El formato de las especificaciones hace que se muestre según lo indicado en la siguiente tabla:

%c	Como un carácter	%s	Como una cadena
%d	Como decimal	%f	Como punto flotante
%o	Como octal	%x	Como hexadecimal

Ejemplo:

```
int f(int x)
{
     assert(width(x)==3,"La anchura de x no es  3
(es %d)", width(x));
      return x+1;
}
void main(void)
{
      int 4 y;
```

```
        y = f(y);
    }
```

El ancho de *x* será inferido al valor 4, así que el mensaje siguiente será mostrado:

```
    F:\proj\test.c(4)(2) : La anchura de x no es 3
(es 4).
```

*5.6 Sentencia **ifselect***

Sintaxis:

```
ifselect (condición)
 sentencia 1
else
 sentencia 2
```

ifselect comprueba el resultado de una expresión constante en tiempo de compilación. Si la condición es verdadera, se compila la siguiente sentencia o bloque de código. Si es falsa, se compila otra sentencia o bloque de código (tras **else)** en caso de que existan. De esta forma, sentencias completas pueden ser seleccionadas o descartadas en tiempo de compilación en función de la evaluación de la expresión.

La construcción **ifselect** permite construir macros recursivas, de una forma parecida a **select**. También es útil dentro de códigos de bloque contradictorios ya que la condición está basada en constantes en tiempo de compilación. Consecuentemente, puede usarse **ifselect** para detectar el primer objeto y el último en un bloque de código contradictorio y crear *pipelines***.**

Ejemplo:

```
int 12 a;
int 13 b;
```

```
int undefined c;

ifselect (width(a) >= width(b))
      c=a;
else
      c=b;
```

Se asigna a *c* el valor de *a* o *b* dependiendo de el ancho de estas variables.

6. Sentencias modificadas o ampliadas

*6.1 Reproducción de bloques **par** y **seq***

Se pueden reproducir los bloques **par** y **seq** usando un bucle enumerado (algo parecido a un bucle **for**). La cuenta se define con un punto de comienzo (base_índice descrita en la sintaxis de abajo), un punto final (límite_índice) y un tamaño de paso (cuenta_índice). El cuerpo del bucle es reproducido tantas veces como pasos haya entre el punto de comienzo y el punto final. Si es un bucle **par,** el proceso de reproducción se ejecutará en paralelo; si es tipo **seq**, se reproducirá en serie.

Sintaxis:

```
par|seq (base_índice;limite_índice;cuenta_índice)
{
      Cuerpo;
}
```

En donde, *base_índice, limite_índice* y *cuenta_índice* son **macro expr** que están declaradas implícitamente. No necesitan ser expresiones únicas, por ejemplo se puede declarar **par** (i=0, j=23; i != 76; i++, j--).

Ejemplo:

```
par (i=0; i<3; i++)
```

```
{
      a[i] = b[i];
}
```

Se expande de la siguiente forma:

```
par
{
      a[0] = b[0];
      a[1] = b[1];
      a[2] = b[2];
}
```

7. Nuevas declaraciones en macros

*7.1 Declaración **let...in***

Las construcciones Handel-C **let** e **in** permiten declarar **macro expr** dentro de otras **macro expr**. En este sentido, macros complejas pueden ser descompuestas en otras más sencillas, en tanto que permanecen agrupadas en un único bloque de código. También permite crear fácilmente macros recursivas tipo **shared**.

La palabra **let** comienza la declaración de un macro local; la palabra **in** finaliza la declaración y define su ámbito.

Ejemplo:

```
macro expr Fred(x) =
 let macro expr y = x*2; in
      y+3;            //Devolverá (x*2)+3
```

La línea superior define el nombre de la macro y su parámetro. La segunda línea define *y* dentro de la definición de la macro. La última línea muestra el valor de la macro completa.

En el siguiente ejemplo se muestra una macro recursiva que utiliza la declaración **let**...**in** en la definición de una macro compartida (**shared**).

```
shared expr mult (p, q) =
   let macro expr multiply(x,y) = select (width(x)==0, 0,
     multiply(x\\1,y<<1)+(x[0]==1?y:0)); in
          multiply(p, q);
```

8. Nuevas declaraciones para relojes

8.1 Reloj en uso actualmente (__clock)

El reloj usado actualmente por una función puede ser referenciado usando la palabra clave **__clock**. Esto permite a la función pasar el reloj actual a una **interface** externa. El valor de la variable de sistema **__clock** será el valor posterior a alguna división. El reloj puede ser interno o externo (**internal clock** o **external clock**).

Ejemplo, en el que se define como parámetro en la declaración de una **interface** el reloj actual **__clock**:

```
interface reg32x1k() registers(address, data_in, __clock,
    write) with {extlib="PluginModelSim.dll", extinst="1;
    model=reg32x1k_wrapper; clock=ck:25"};
```

9. Declaraciones para relojes modificadas o ampliadas

*9.1 Declaraciones de reloj **internal** e **internal_divide***

Se pueden tener varios relojes interactuando en un determinado diseño. Cada función **main()** debe estar asociada con su correspondiente reloj.

Con la versión 3.0 Beta de Handel-C se puede hacer que el reloj sea cualquier expresión o expresión dividida por un determinado factor. Para chips de la serie Xilinx 4000 se puede hacer que el reloj sea un valor leído del generador de reloj que reside en el propio integrado.

La sintaxis es la siguiente:

```
set clock = internal <Expresión>;
set clock = internal_divide <Expresión> factor;
```

Esto permite establecer el reloj de nuestra función **main()** con un determinado valor leído desde un puerto *software*.

Obsérvese el siguiente ejemplo:

```
interface port_in (unsigned 1 clk) ClockPort();
set clock = internal ClockPort.clk;
```

9.2 Comunicación entre dominios de reloj

No es legal acceder a la misma variable desde dominios de reloj distintos. En vez de eso, se deben transmitir datos entre distintos dominios de reloj usando canales o puertos.

Los canales que se conectan entre distintos dominios de reloj deben ser unidireccionales y punto a punto. De esta manera, con el primer uso que se le de al canal definiremos su dirección y los dominios que van a transmitir y recibir por el mismo. Si se intenta reutilizar el canal en la dirección opuesta o entre otros dominios de reloj, el compilador generará un error.

Los canales usados entre dominios de reloj deben estar declarados en un archivo y luego hacer referencia a éstos como tipo **extern** desde otro archivo.

La temporización entre dominios no está especificada, pero la transmisión está garantizada y ambos lados esperarán a que ésta haya finalizado. Como ejemplo, ver el siguiente código:

```
/*      Archivo Transmit.c          */
chan 8 c;           // El canal debe tener ámbito global

main()
{
      int 8 x, y;
      c ! x;      //El programa esperará a que la transmisión
                  //se haya completado satisfactoriamente.
      c ! y;
}

/*      Archivo Receive.c           */
extern chan c;

main()
{
      int 8 p, q;

      c ? p;
      c ? q;
}
```

10. Cambios en enlazador

Con la versión 3.0 Beta de Handel-C se pueden enlazar juntos y cargar en una misma FPGA varios archivos. Esto permite crear y acceder a archivos de librería.

Se puede cargar un mismo chip con varias funciones **main()**. Esto quiere decir que se pueden tener bloques lógicos independientes que usan relojes distintos funcionando dentro de una misma FPGA. Los relojes pueden ser tanto internos como externos y éstos últimos pueden ser especificados por el usuario.

Análisis y descripción de periféricos de la familia PIC16 de Microchip

Capítulo IV. ANÁLISIS Y DESCRIPCIÓN DE PERIFÉRICOS DE LA FAMILIA PIC16 DE MICROCHIP.

En este capítulo se pretende dar una descripción somera de los periféricos de la familia PIC-16 del fabricante Microchip que se han utilizado como ejemplo de desarrollo de código Handel-C.

La utilización de periféricos, que tendrán un funcionamiento en paralelo y que utilizarán numerosos interfaces *hardware* reales para comunicarse, nos permitirá hacer un estudio de la capacidad del lenguaje Handel-C para hacer una descripción eficaz (en cuanto a que se consiga el funcionamiento de lo descrito) y eficiente (en cuanto a los recursos físicos y temporales que disponga) de los distintos procesos y de los dispositivos *hardware* que a continuación se van a exponer.

1. Watchdog Timer (WDT)

El *Watchdog Timer* es un oscilador RC que funciona autónomamente del integrado y que no precisa ningún componente externo. Este oscilador RC está separado del oscilador RC que genera el reloj del sistema. Esto quiere decir que el WDT funcionará incluso si se detiene por algún motivo el reloj del sistema (por ejemplo en la ejecución de la instrucción SLEEP). Durante un funcionamiento normal o una operación SLEEP, el reset del WDT provocará un reset general del sistema.

Cuando el *Watchdog Timer* provoca un reset afectará al registro STATUS, de manera que la causa que generó este reset queda monitorizada en dicho registro.

Este dispositivo puede estar permanentemente inhabilitado mediante una programación adecuada del registro CONFIG.

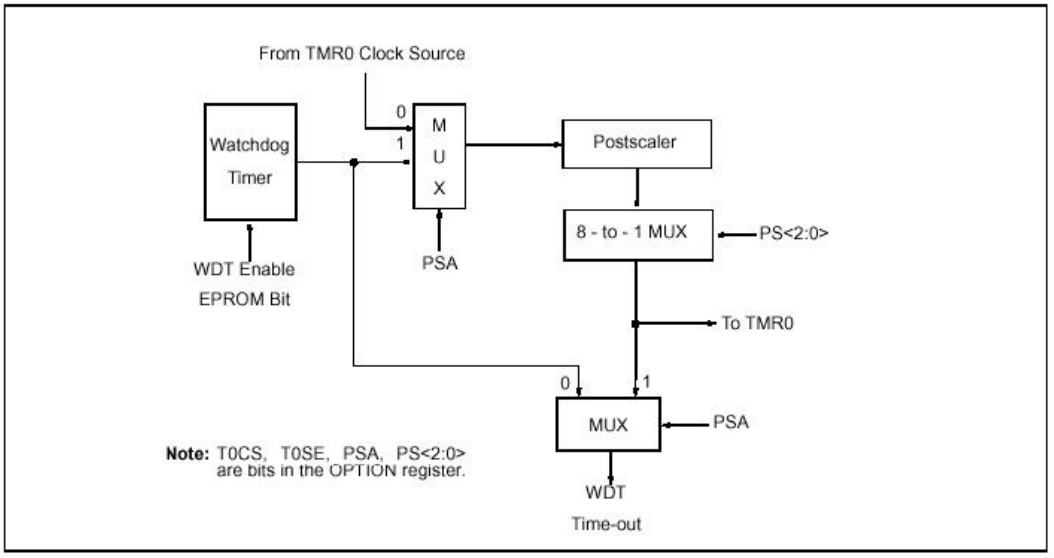


Figura 45. Diagrama de bloques del *Watchdog Timer*.

El WDT tiene un período de expiración nominal (sin *scaler*). Si se necesita un período mayor puede asignársele al WDT un *postscaler* con un coeficiente de división 1:128. Esta asignación es *software*, escribiendo la configuración adecuada en el registro OPTION.

La instrucción CLRWDT pone a cero el WDT y el *postscaler* (en el caso de estar asignado a éste) e impide que el tiempo de expiración concluya y genere el correspondiente reset general.

2. Timer0

El módulo **Timer0** es un contador-temporizador de 8-bits. La fuente de reloj puede ser el reloj interno del sistema (Fosc/4) o un reloj externo. Cuando la fuente de reloj es un reloj externo, el módulo **Timer0** puede ser configurado para que se incremente con los flancos de subida o con los de bajada.

Este módulo también tiene la opción de utilizar un *prescaler* programable. Este *prescaler* puede ser asignado o al módulo **Timer0** o al **Watchdog Timer**. El bit PSA (nº 3 del registro OPTION) asigna el *scaler* y los bits PS2, PS1 y PS0 (nº 2, 1 y 0 del registro OPTION) determinan el valor del *prescaler*. El **Timer0** puede ser incrementado con las siguientes tasas: 1:1, cuando el *scaler* está asignado al **Watchdog Timer**, 1:2, 1:4, 1:8, 1:16, 1:32, 1:64, 1:128 y 1:256.

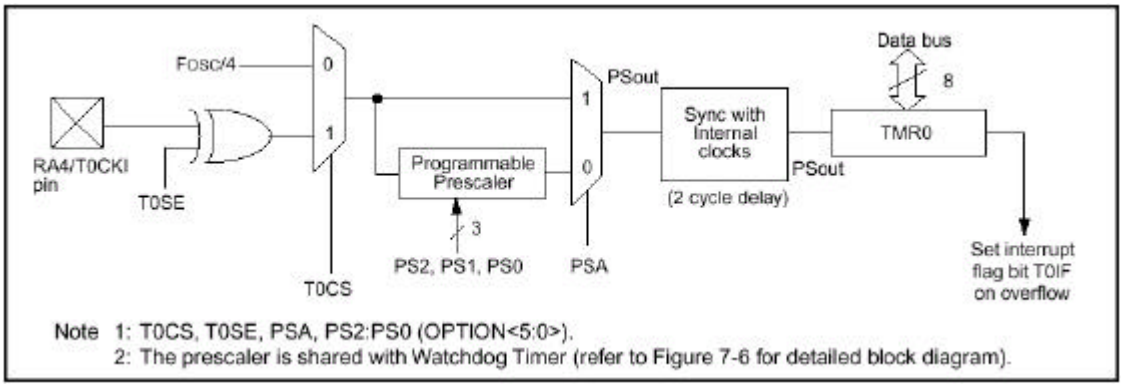


Figura 46. Diagrama de bloques del Timer0.

La sincronización con el reloj externo ocurre tras el *prescaler*. Cuando se utiliza el *prescaler*, la frecuencia del reloj externo debe ser mayor que la frecuencia del dispositivo.

3. Timer1

El **Timer1** es un contador-temporizador de 16 bits. La fuente de reloj puede ser el reloj interno del sistema (Fosc/4), un reloj externo o un cristal externo. El **Timer1** puede operar como temporizador o como contador. Cuando funciona como contador (fuente de reloj externa) puede o trabajar sincronizado con el dispositivo o de forma asíncrona. La forma asíncrona permite al **Timer1** funcionar durante el modo SLEEP.

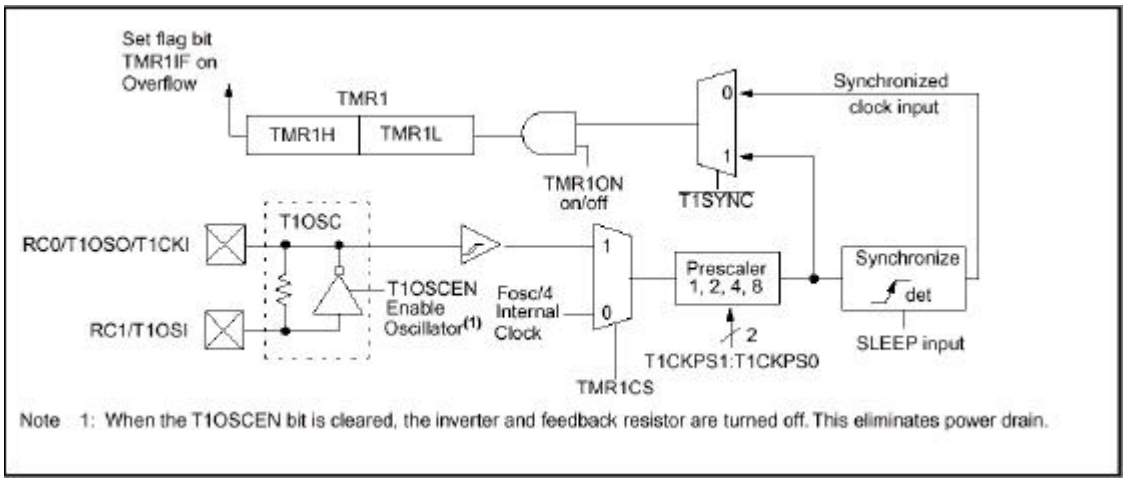


Figura 47. Diagrama de bloques del Timer1.

El **Timer1** también tiene la opción de usar un *prescaler* cuyas tasas de incremento son las siguientes: 1:1, 1:2, 1:4 y 1:8. El **Timer1** puede ser usado conjuntamente con el módulo **Capture/Compare/PWM**. Cuando se usa junto al módulo **CCP** el **Timer1** es la base de tiempo para capturas de 16 bits o comparaciones de 16 bits y debe ser sincronizado con el dispositivo. El oscilador del **Timer1** es también una de las fuentes de reloj del módulo **LCD**.

4. Timer2

El **Timer2** es un temporizador de 8 bits con un *prescaler* y un *postscaler* programables, además tiene un registro de período de 8 bits (PR2). Este temporizador puede ser usado en conjunto con el módulo **CCP** (en el modo **PWM)** y también como fuente de reloj para el Puerto Serie Síncrono (*Synchronous Serial Port,* **SSP)**. La opción de prescaler permite incrementos del **Timer2** con tasas: 1:1, 1:4 y 1:16.

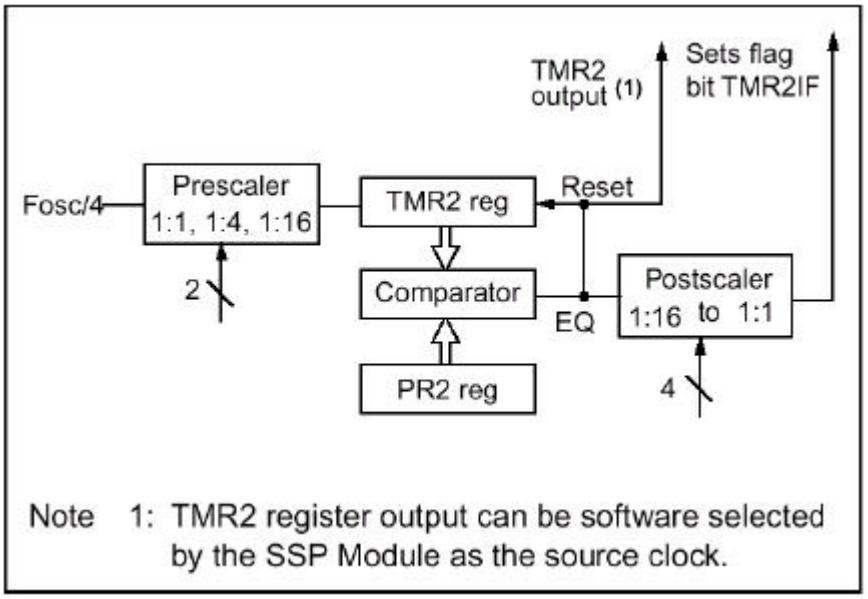


Figura 48. Diagrama de bloques del Timer2.

El *postscaler* permite la coincidencia del registro TMR2 con el registro de período (PR2) un número de veces programable antes de generar una interrupción. El *postscaler* puede ser programado desde la tasa 1:1 a la 1:16 inclusive.

5. Módulo Capture/Compare/Pulse_Width_Modulation (CCP)

El módulo **CCP** puede operar en cualquiera de los siguientes modos: Capturador de 16 bits, Comparador de 16 bits y Modulador de Ancho de Pulsos ampliable a 10-bits.

5.1 Modo Capture

En modo *Capture* retiene valores de 16 bits del **TMR1** en el par de registros de 8 bits CCPR1H:CCPR1L. El evento de captura puede ser programado para que ocurra en los siguientes casos: flanco de bajada, flanco de subida, cuarto flanco de subida o décimo sexto flanco de subida de la señal que entra por el pin CCP1.

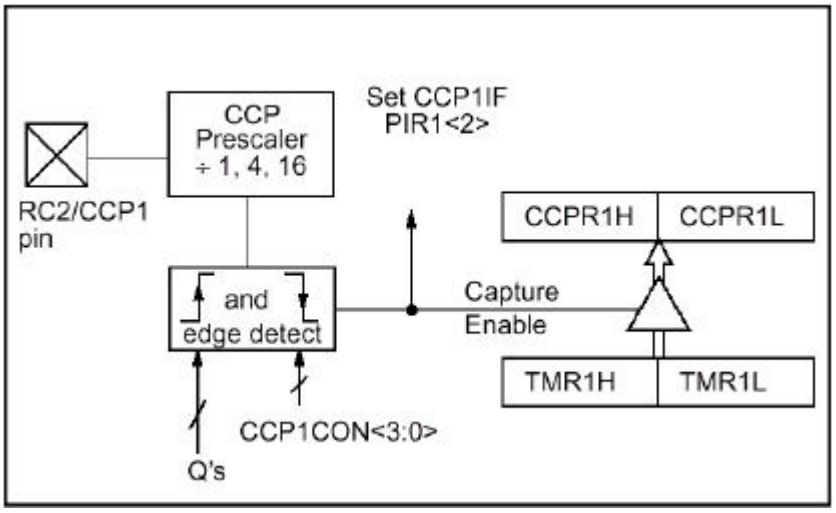


Figura 49. Diagrama de bloques del módulo CCP, modo *Capture.*

5.2 Modo Compare

En el modo *Compare* se hará una comparación entre los pares de registros TMR1H:TMR1L y CCPR1H:CCPR1L. Cuando la comparación resulta positiva se puede generar una interrupción y la salida por el pin CCP1 puedeser forzada un valor determinado (alto o bajo). El TMR1 puede ser reseteado y comenzar una conversión A/D. Esto se configura con los bits CCP1M3 a CCP1M0 del registro CCPCON.

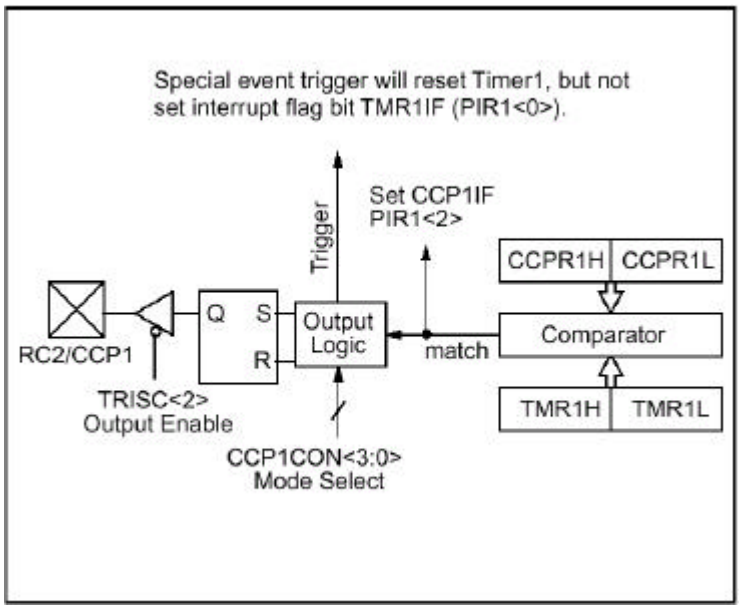


Figura 50. Diagrama de bloques del módulo CCP, modo *Compare.*

5.3 Modo Pulse Width Modulation

En el modo *Pulse Width Modulation* se compara el registro TMR2 con un registro de ciclo completo de 10 bits (CCPR1H:CCPR1L<5:4>) así como con un registro de período de 8 bits (PR2). Cuando el registro TMR2 se iguala al registro de ciclo completo, el pin CCP1 será forzado a salida a nivel bajo. Cuando el TMR2 se iguala al PR2 se pone a cero (00h) el TMR2, el pin CCP1 tendrá una salida forzada a nivel alto y se puede generar una interrupción.

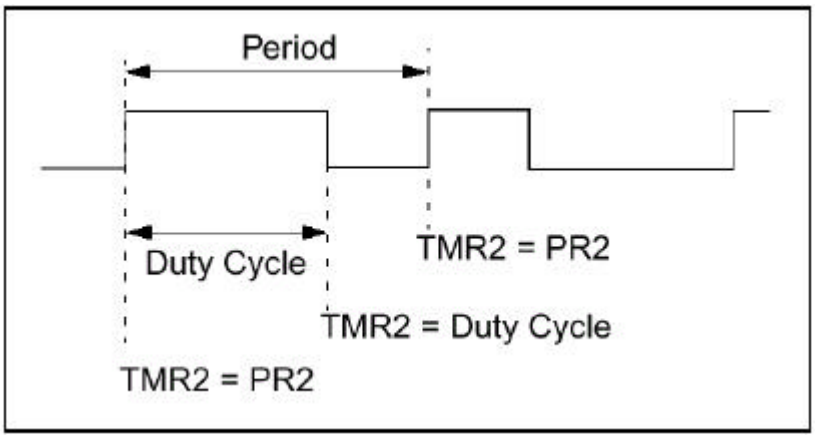


Figura 51. Salida del pin CCP1 en modo *Pulse Width Modulation.*

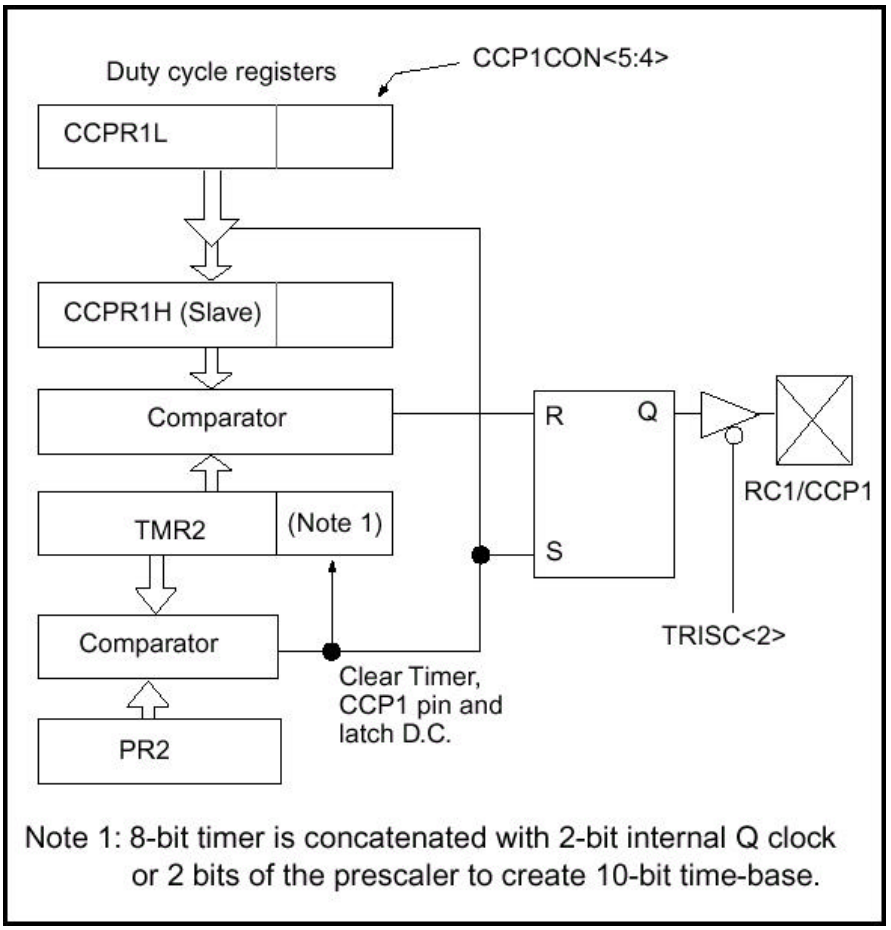


Figura 52. Diagrama de bloques del módulo CCP, modo *Pulse Width Modulation.*

6. Módulo Puerto Serie Síncrono (SSP)

El módulo **Puerto Serie Síncrono (SSP)** es una interfaz serie útil para la comunicación con otros periféricos o dispositivos microcontroladores. Estos dispositivos periféricos pueden ser EEPROMs serie, registros de desplazamiento, controladores de *displays*, convertidores A/D, etc. El módulo **SSP** puede trabajar en los siguientes modos:

SPI (*Serial Peripheral Interface*).
I²C (*Inter-Integrated Circuit*).

6.1 Modo SPI

El modo **SPI** permite la transmisión síncrona de 8 bits de datos y la recepción simultánea. Para realizar la comunicación son necesarios los siguientes pines:

- Salida de datos serie (*SDO*).
- Entrada de datos serie (*SDI*).
- Reloj serie (*SCK*).
- Selección de *slave* ($\overline{SS}$).

Programando el registro de control SSPCON podemos conseguir varios modos de funcionamiento. Los bits de control de este registro permiten especificar:

- Modo *Master* (*SCK* es la salida de reloj).
- Modo *Slave* (*SCK* es la entrada de reloj).
- Polaridad del reloj (estado de reposo de *SCK*).
- Flanco de reloj (salida de datos en flanco de subida o bajada de *SCK*).
- Tasa de reloj (sólo en modo *Master*).
- Modo de selección de *slave* (sólo en modo *Slave*).

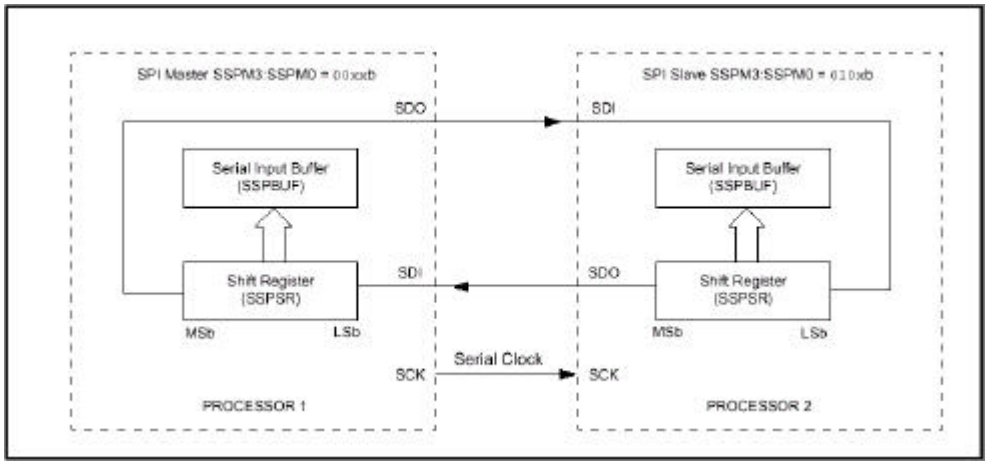


Figura 53. Conexión *Master-Slave* en modo SPI.

El módulo **SSP** consiste en un registro de desplazamiento transmisor/receptor (SSPSR) y un registro *buffer* (SSPBUF). El SSPSR desplaza el dato de entrada y lo saca del dispositivo, comenzando por el bit más significativo. El registro SSPBUF mantiene el dato que fue escrito en el SSPSR hasta que el dato recibido esté completo. Una vez leídos los 8 bits de datos que se han recibido, se copian al registro SSPBUF.

En este modo se controlarán además:

- Las colisiones, que se producen si se escribe en el registro SSPBUF mientras se está transmistiendo el dato anterior.
- *Overflows*, que se producen si se recibe un nuevo dato mientras el registro SSPBUF todavía está manteniendo el dato anterior, es decir, cuando el usuario aún no ha leído el dato previo. En este caso se perderá el dato nuevo contenido en SSPSR. Esto sólo puede ocurrir en modo *slave.*
- Las interrupciones, que indicarán cuando se ha producido una transmisión o recepción.
- La temporización en modo *Master*, que puede tener una fuente de reloj que proceda del reloj interno del sistema (Tcy) y que puede ser escalada con tasas 1:4, 1:16 ó 1:64, o bien puede proceder del **Timer2** escalada con una tasa 1:2.

En la figura podemos observar el correspondiente diagrama de bloques del modo SPI, los pines utilizados y las señales.

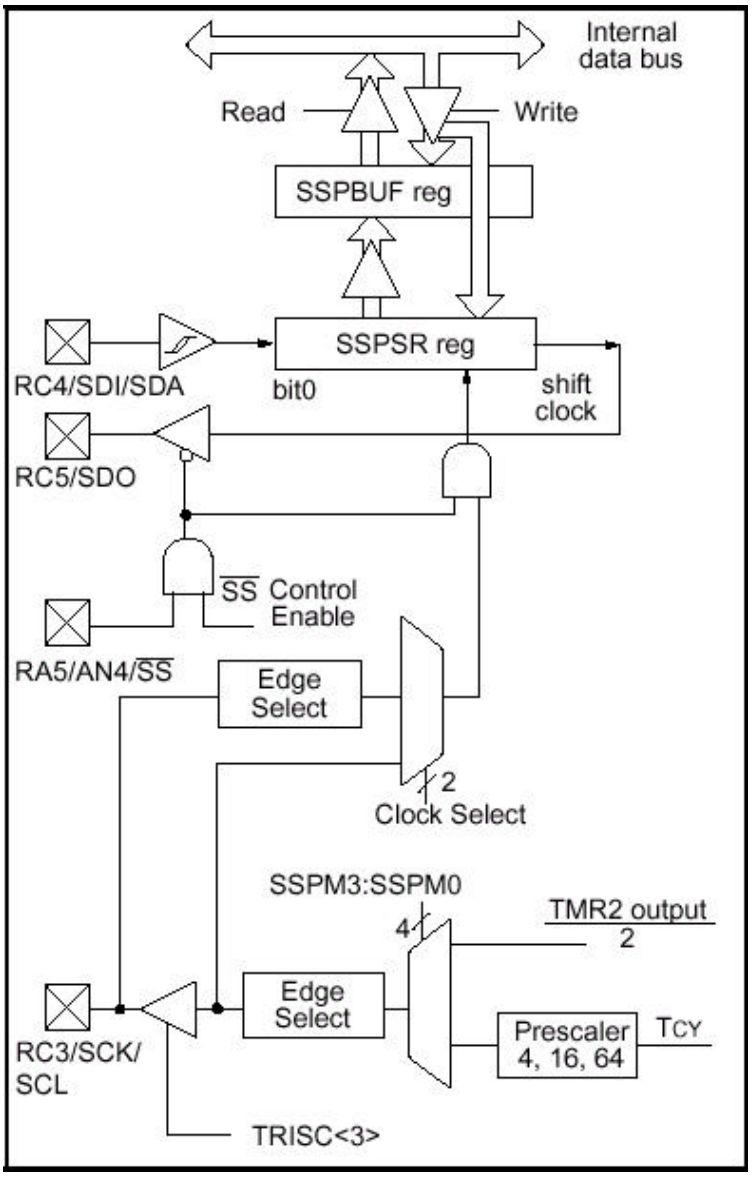


Figura 54. Diagrama de bloques del módulo SSP, modo SPI.

6.2 Modo I²C

Para comprender el funcionamiento del bus I²C consultar el apéndice I: **Especificaciones del bus I²C**.

El módulo SSP en el modo I²C implementa completamente todas las funciones en modo *slave*, y proporciona interrupciones en los bits de START y STOP en *hardware*. De esta manera también se podrá conseguir la implementación *firmware* de las funciones *master*. El módulo **SSP** implementa

las especificaciones del modo estándar así como direccionamiento de 7-bits ó 10-bits. Se utilizan dos pines para la transferencia de datos. Estos son: el pin RC3 que recibe o emite la señal de reloj y el RC4 que se conecta con la línea de datos. Estos pines deben ser configurados como entradas o salidas usando los bits del registro TRISC. Las funciones del módulo **SSP** son habilitadas activando el bit SSPEN (*Bit Enable SSP*).

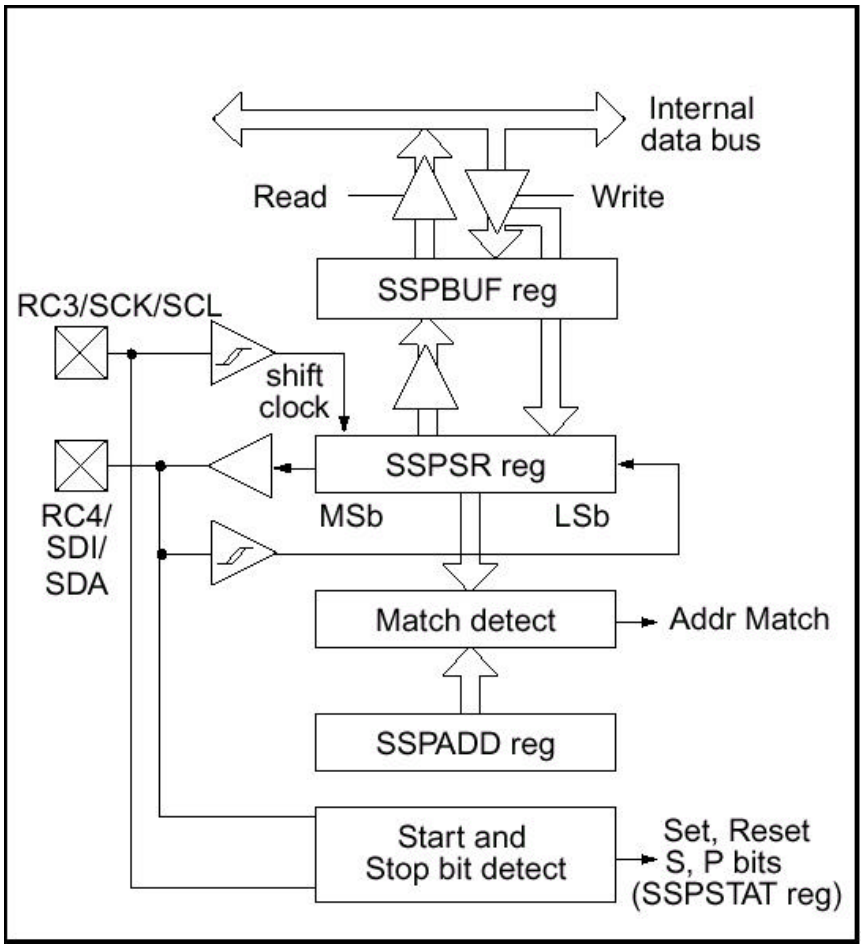


Figura 55. Diagrama de bloques del módulo SSP (modo I²C).

El módulo **SSP** tiene cinco registros para el modo I²C, éstos son:

- Registro de control **SSPCON**.
- Registro de estado **SSPSTAT**.
- *Buffer* de emisión/transmisión serie **SSPBUF**.

- Registro de desplazamiento, que no es direccionable externamente, **SSPSR**.
- Registro de direcciones **SSPADD**.

El registro **SSPCON** permite controlar el modo de funcionamiento en modo I²C. El modo de funcionamiento del bus I²C puede ser seleccionado mediante los cuatro bits de selección de modo de este registro.

- Modo *slave* I²C (dirección de 7-bits).
- Modo *slave* I²C (dirección de 10-bits).
- Modo *slave* I²C (dirección de 7-bits, con interrupciones en los bits de START y STOP).
- Modo *slave* I²C (dirección de 10-bits, con interrupciones en los bits de START y STOP).
- Modo *master* I²C controlado por *firmware*, el *slave* no se conecta.

Si se selecciona alguno de los modos I²C, poniendo a '1' el bit SSPEN, se forzará a que los pines SCL y SDA estén a drenador abierto, siempre que estos pines estén programados como entradas poniendo a '1' los bits pertinentes del registro TRISC.

El registro **SSPSTAT** muestra el estado de la transferencia de datos. Esta información incluye la detección de los bits START y STOP, especifica si el byte recibido es de dirección o de datos, si el siguiente byte es de conclusión de la dirección de 10-bits y si este byte es una de transferencia de datos de lectura o escritura. El registro **SSPSTAT** es de sólo lectura.

El registro **SSPBUF** es el registro en el cual se almacena el byte leído o el byte que va a ser enviado. El registro **SSPSR** desplaza el dato fuera o dentro del dispositivo. En operaciones de recepción, los registros **SSPSR** y **SSPBUF** suponen un almacenamiento doble. Esto permite que comience la recepción del siguiente byte antes de que finalice la lectura del último byte de datos. Cuando la

recepción del byte finaliza, éste es transferido al registro **SSPBUF** y se pone a '1' el bit de señalización SSPIF. Si se recibe otro byte completo antes de que se lea el registro SSPBUF ocurre un desbordamiento (*overflow*) y el bit SSPOV del registro SSPCON se pone a '1', perdiéndose el byte que está en el registro SSPSR.

El registro **SSPADD** mantiene la dirección del *slave*. En el modo de dirección de 10-bits, el usuario necesita escribir el byte alto de dirección (1 1 1 1 0 A9 A8 0). Tras la coincidencia del byte alto de dirección se debe cargar el byte bajo de la misma.

Realización de la librería de periféricos en Handel-C

Capítulo V. REALIZACIÓN DE LA LIBRERÍA DE PERIFÉRICOS EN HANDEL-C

1. Introducción

A continuación se va a explicar el código fuente Handel-C que constituye el ejemplo realizado en este proyecto (para ver el código completo consultar el apéndice B).

Mediante la realización de este ejemplo se pretendió utilizar, comprobar y analizar las distintas herramientas del entorno de programación, como el precompilador, el depurador, el enlazador y el compilador que se han descrito en el capítulo II. Para esto se decidió crear un proyecto complejo dividido en varios archivos (código fuente con función principal, códigos fuente de funciones, archivos librería, archivos cabecera y archivos de entrada/salida de datos) que son precompilados, depurados, enlazados y compilados en un mismo proyecto Handel-C.

La compilación del proyecto será triple. En una primera compilación (DEBUG) podremos acceder a la fase depuración que nos permitirá usar el depurador de código y simular, mediante canales, las distintas comunicaciones que se producen entre los periféricos, el Testbench (que será expuesto a continuación) y el exterior. En una segunda compilación (RELEASE) podemos obtener archivos HTML que nos indican la eficiencia del código en cuanto al consumo de puertas lógicas y retardo y que será una buena indicación del tamaño y velocidad que obtendremos en la fase de implementación del proyecto, ver apartado 12.2 del capítulo II de esta memoria. Además, si se desea, con esta compilación podremos obtener una versión compilada en la que se oculta el código de las librerías creadas por el programador para proteger la propiedad intelectual. Por último con la compilación EDIF podremos obtener los archivos *netlist* necesarios para la implementación real en la FPGA y que será necesaria para la correspondiente fase de pruebas final.

Por último, y para justificar la naturaleza del ejemplo, podemos decir que los periféricos emulados en una FPGA son una buena excusa que nos proporcionará la posibilidad de probar la eficacia de Handel-C en aspectos no estudiados con anterioridad a este proyecto relacionados con las comunicaciones externas y con la ejecución paralela masiva.

2. Explicación de los bloques de código

Como se ha explicado anteriormente, este proyecto consta de varios archivos fuente Handel-C, archivos cabecera, archivos librería y archivos que simularán las entradas y salidas por los pines de la FPGA.

2.1 Archivo top.h

Este archivo solamente tiene directivas *#define* y *#undef*. Con estas directivas podemos incluir los distintos periféricos y el proceso de escucha del reset en la compilación o excluirlos según convenga. Además podemos especificar con las palabras **Simul**, **Placa** y **Real** qué código es el pertinente para la simulación en el PC, cuál para la fase de pruebas en la placa y cuál sería el que generara el *hardware* final respectivamente, con las directivas *#ifdef*, *#ifndef*, *#else*, ... en los sitios correspondientes.

Este archivo, que debe ser incluido por el resto de archivos, puede ser evitado utilizando el entorno de programación para incluir o excluir el código que queramos con la declaración de palabras llave (equivalente a hacer *#define*).

2.2 Archivo cabecera.h

Este archivo cabecera contiene la declaración externa de todas las variables globales, constantes generales, constantes relacionadas con el direccionamiento, constantes relacionadas con las interrupciones, registros del

Testbench y canales del Testbench que se utilizan en los distintos archivos fuente.

2.3 Archivo milib.h

Este archivo cabecera contiene los prototipos de las macros, tanto expresiones como procedimientos, que se desarrollan en el archivo librería milib.c de creación propia.

2.4 Archivo stdlib.h

Este archivo cabecera contiene los prototipos de las macros creados por la empresa Celoxica Ltd. y cuya publicación está reservada.

2.5 Archivo milib.c

Este archivo contiene una librería de macros personal. Las macros incluidas son:

Expresiones:

- setbit(b,e): esta macro recursiva pone a '1' un bit determinado de una variable sin importar el ancho de ésta.
- clrbit(b,e): esta macro recursiva pone a '0' un bit determinado de una variable sin importar el ancho de ésta.

Procedimientos:

- escribe(REG,DATO): este procedimiento escribe un dato de 8 bits en el Testbench.
- lee (REG,w): este procedimiento hace justamente lo contrario, leer un dato del Testbench.

2.6 Archivo stdlib.c

En este archivo librería se incluyen todos las definiciones de las siguientes macros:

a) Relacionadas con la manipulación de bits: adjs, adju, copy, lmo, lmz, population, rmo, rmz y top.

b) Aritméticas: abs, addsat, decode, div, exp2, incwrap, log2ceil, log2floor, mod, sign y subsat.

Además se definen las constantes TRUE y FALSE.

2.7 Archivo Testbench.c

El bloque principal, llamado TESTBENCH, incluido en este archivo, contiene la única función **main**() que existe en todo el proyecto. Esto quiere decir que habrá un único dominio de reloj en todos los bloques y no será necesario recurrir a bloques de sincronización externos usando para ello librerías como **synchroniser.dll** suministrada por Handel-C.

Mediante este bloque se podrán configurar todos los periféricos implementados utilizando un bloque de registros realizados con variables globales y que pueden ser leídos y escritos desde el exterior. El bloque de registros está formado por los 17 de registros de 8 bits, entre los que están registros de configuración, registros de estado, registros accesibles desde el exterior, un registro de enmascaramiento de interrupciones y un registro de interrupciones de sólo lectura. A continuación enumero y describo cada uno de ellos:

1. **CONFIGH**: Registro superior de configuración. Los bits de este registro doble sirven para programar varias configuraciones de dispositivos. En este registro, 4 bits (lsb) forman parte de códigos de protección y 4 bits (msb) son de relleno ya que no están implementados en el PIC original. De esta manera habrá consistencia en el tamaño de los registros y los buses de entrada/salida de datos.

2. **CONFIGL**: Registro inferior de configuración. En este registro los 5 bits más significativos forman parte de códigos de protección (no

implementados en el proyecto). Tiene además dos bits de selección del tipo de oscilador y un bit de habilitación del **Watchdog Timer**.

3. **STATUS**: Es el registro de estado del microcontrolador. De este registro sólo se van a afectar dos bits. Las escrituras en el registro de estado son debidas a los resets provocados por el **Watchdog Timer**.

4. **OPTION**: Este registro es el registro de configuración tanto del **Watchdog Timer** como del **Timer0**. Mediante los bits de este registro se asignará el *scaler* (*prescaler* para el **Timer0** y *postscaler* para el **Watchdog Timer**), se programará la tasa del *scaler* y se seleccionará la entrada de reloj y el flanco activo para el **Timer0**.

5. **T1CON**: Registro de configuración del **Timer1**. Con los bits de este registro podremos configurar la tasa del *prescaler* de este periférico, se habilitará el oscilador de entrada, se seleccionará la fuente de reloj y se habilitará, o no, este contador/temporizador de 16 bits.

6. **T2CON**: Registro de configuración del **Timer2**. Cuatro bits de este registro son para seleccionar la escala del *prescaler*, mientras que dos bits son suficientes para seleccionar la del *postscaler*. Un bit más habilitará, en su caso, el propio periférico.

7. **PR2**: Registro de período del **Timer2**. En este registro se almacenará el número de cuenta límite de este temporizador.

8. **CCP1CON**: Registro de configuración del periférico **CCP (***Capture/ Compare/ PWM***)**. Con los cuatros bits menos significativos de este registro podemos programar el modo de funcionamiento del periférico **CCP**. Dos bits más sirven como extensión del registro **CCPR1L** en el modo de generador de pulsos.

9. **CCPR1L**: Registro bajo del periférico **CCP**. Este registro almacenará los 8 bits menos significativos del registro doble CCPR1. Su ubicación en el Testbench es debida al modo **PWM** en el que puede ser escrito desde el exterior (no ocurre los mismo con el registro par CCPR1H).

10. **TRISC**: Registro de control de direcciones de datos del puerto C. Mediante los bits de este registro podemos configurar la dirección de los pines que intervienen en las comunicaciones externas de los periféricos. En la versión original del PIC solamente se utilizan los 6 bits menos significativos, pero en la implementación del proyecto se han utilizado además los dos bits más significativos para controlar las puertas triestado de los pines *TOCKI,* del **Timer0**, y del pin $\overline{SS}$ del periférico **SSP** originalmente controlados por el registro TRISA, por motivos de optimización de los recursos.

11. **SSPCON**: Registro de configuración del periférico **SSP** (*Synchronous Serial Port*). Con los cuatro bits menos significativos se controla el modo de funcionamiento del periférico (modo SPI o modo I²C). Además tiene bits de configuración de la polaridad del reloj, de habilitación del periférico, indicador de sobrecarga de recepción (*overflow*) e indicador de colisiones.

12. **SSPSTAT**: Registro de estado del periférico **SSP**. Este registro indica el estado general del periférico en la transmisión/ recepción de datos seriales.

13. **SSPBUF**: Búfer del periférico **SSP**. En este registro se almacena un dato de 8 bits recibido, esperando a ser leído externamente, o bien, un dato de 8 bits dispuesto a ser transmitido por el puerto serie.

14. **SSPADDH**: Registro alto de dirección del periférico **SSP**. Este registro contiene los dos bits más significativos de la dirección de 10 bits del periférico **SSP** en el modo I²C. Además contiene una cadena fija de 5 bits para la sincronización del *slave* con el *master*.

15. **SSPADDL**: Registro bajo de dirección del periférico **SSP**. Contiene los 8 bits menos significativos de la dirección de 10 bits o la dirección de 7 bits completa del periférico **SSP** en el modo I²C.

16. **MASK**: Registro de máscara de interrupciones del sistema. Contiene 8 posiciones correspondientes a las 8 interrupciones posibles que se pueden dar en el conjunto TESTBENCH-PERIFÉRICOS. Este registro es original del proyecto y aglutina al correspondiente registros de máscara de interrupciones del PIC (registro PIE1) y hace enmascarables otras interrupciones.

17. **INTER**: Registro de interrupciones del sistema. Este registro es de sólo lectura y al igual que MASK es una adaptación del correspondiente registro de interrupciones del PIC (registro PIR1).

El conjunto de bits de acceso lectura/escritura que forman parte del bloque TESTBENCH y que configuran a los periféricos están expresados en la tabla.

Las líneas (declaradas como *interfaces*) que realizan las comunicaciones externas de este bloque son:

Un bus de entrada de datos de 8 bits.
Un bus de salida de datos de 8 bits.
Un bus de entrada de direcciones de 5 bits.
Una línea entrada de lectura/escritura.
Una línea de salida de interrupción.
Una línea de entrada de reset.

	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
CONFIGH	-	-	-	-	-	-	-	-
CONFIGL	-	-	-	-	CP	WDTE	FOSC1	FOSC0
STATUS	PA2	PA1	PA0	TO	PD	Z	DC	C
OPTION	-	-	TOCS	TOSE	PSA	PS2	PS1	PS0
T1CON	-	-	T1CKPS1	T1CKPS0	T1OSCEN	T1SYNC	TMR1CS	TMR1ON
T2CON	-	TOUTPS3	TOUTPS2	TOUTPS1	TOUTPS0	TMR2ON	T2CKPS1	T2CKPS0
PR2	PR2[7]	PR2[6]	PR2[5]	PR2[4]	PR2[3]	PR2[2]	PR2[1]	PR2[0]
CCP1CON	-	-	CCP1X	CCP1Y	CCP1M3	CCP1M2	CCP1M1	CCP1M0
CCPR1L	CCPR1L[7]	CCPR1L[6]	CCPR1L[5]	CCPR1L[4]	CCPR1L[3]	CCPR1L[2]	CCPR1L[1]	CCPR1L[0]
TRISC	TOCKI	SS	SDO	SDI	SCK	CCP1	RC1	RC0
SSPCON	WCOL	SSPOV	SSPEN	CKP	SSPM3	SSPM2	SSPM1	SSPM0
SSPSTAT	SMP	CKE	D/A	P	S	R/W	UA	BF
SSPBUF	SSPBUF[7]	SSPBUF[6]	SSPBUF[5]	SSPBUF[4]	SSPBUF[3]	SSPBUF[2]	SSPBUF[1]	SSPBUF[0]
SSPADDH	1	1	1	1	0	A9	A8	-
SSPADDL	A7	A6	A5	A4	A3	A2	A1	A0
MASK	BIT STOP	BIT START	BUFFER FULL	COMPARATION	CAPTURE	TIMER 2	TIMER 1	TIMER 0
INTER	BIT STOP	BIT START	BUFFER FULL	COMPARATION	CAPTURE	TIMER 2	TIMER 1	TIMER 0

Se pueden programar el Testbench de forma que cualquier periférico pueda funcionar en alguno de los modos de funcionamiento de los descritos por la guía del fabricante.

Este bloque permite resetear todos los registros que configuran los periféricos tras una señal externa de reset (este reset será síncrono por limitaciones del propio lenguaje).

Las configuraciones de los modos de funcionamiento del sistema que se van a utilizar en la fase de pruebas, las lee el bloque Testbench del archivo inicio.c. Estas configuraciones serán descritas más adelante.

El bloque Testbench permite la ejecución en paralelo de todos los procesos que implementan a periféricos (**Wacthdog Timer, Timer0, Timer1, Timer2, CCP** y **SSP**), junto con el procedimiento de escucha de reset, procedimiento de lectura/escritura en los registros y procedimiento de escucha de los canales de interrupción con posible generación de interrupción externa.

De entre estos procesos merece la pena mencionar el último, escucha de los canales de interrupción, ya que en él se utilizan procedimientos y objetos novedosos de la versión 3.0 Beta de Handel-C.

Para todas las interrupciones que se producen en los periféricos y que activan algún bit del registro inter del Testbench se ha utilizado un array de canales y variables signal que tienen la particularidad de estar activas durante un único ciclo de reloj y que pueden reproducirse dentro del mismo ciclo.

El trozo de código que hace la función de escucha de los canales de interrupción con posible generación de interrupción es el siguiente:

```
par
{
      ...
      while(TRUE)
        {
        /***        PROCESOS DE ESCUCHA DE CANALES DE INTERRUPCIÓN    ***/
            par
            {
                bus_int[INT_TMR0] ? cause[INT_TMR0];
                bus_int[INT_TMR1] ? cause[INT_TMR1];
                bus_int[INT_TMR2] ? cause[INT_TMR2];
                bus_int[INT_CCP1] ? cause[INT_CCP1];
                bus_int[INT_CCP2] ? cause[INT_CCP2];
```

```
                    bus_int[INT_SSP1] ? cause[INT_SSP1];
                    bus_int[INT_SSP2] ? cause[INT_SSP2];
                    bus_int[INT_SSP3] ? cause[INT_SSP3];
                }
            }
            while(TRUE)
            {
                par
                {
                    interrup =    cause[7]@cause[6]@cause[5]@cause[4]@
                                cause[3]@cause[2]@cause[1]@cause[0];
                    inter = inter | interrup;

                /* PROCESO DE ACTIVACIÓN DE LA LÍNEA DE INTERRUPCIÓN EXTERNA */

                #ifndef Real
                    if (inter&mask)
                        INTERRUP ! TRUE;
                    else
                        INTERRUP ! FALSE;
                #endif
                }
            }
           ...
}
```

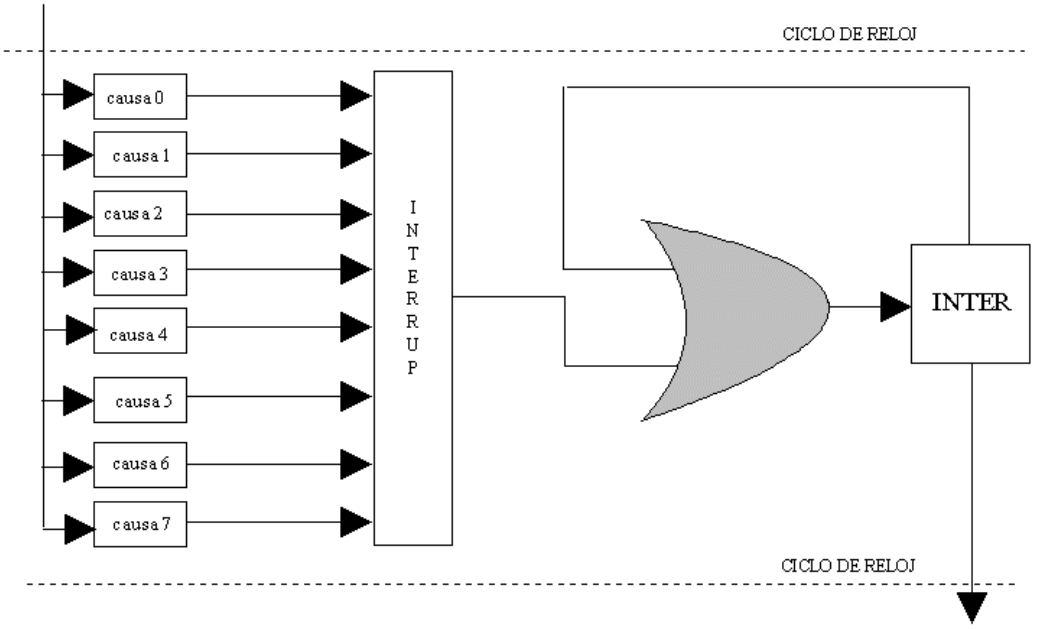


Figura 56. Diagrama de flujo del proceso de escucha de interrupciones.

cause [INT_XXX#] es un array de señales que leen continuamente un array de canales (canales de interrupción) y que en cada ciclo de reloj actualizan una variable auxiliar (interrup). Esta actualización se produce mediante la concatenación de las 8 señales (de anchura 1 bit) de recepción de interrupciones formando así un único valor de anchura 8 bits. Este valor (interrup) es a su vez una nueva señal de 8 bits que está directamente relacionada con el registro **inter**.

Si resulta que la interrupción generada no está enmascarada se generará la interrupción externa mediante escritura en un canal (en modo Simul) o en una interface (en los modos Placa o Real). Lo interesante de todo esto es que el proceso de lectura del canal, la actualización de la variable interrup, la actualización del registro de interrupción y la generación de la interrupción externa se producen en el mismo ciclo de reloj. Esto permite que no se pierda ninguna interrupción.

Con este ejemplo se muestra la potencia de las variables signal combinadas con los canales y los procesos en paralelo.

2.8 Archivo Inicio.c

Mediante este archivo podemos configurar el Testbench de manera que puedan ser probados los periféricos independientemente en el modo Placa. En este modo, se ha programada la FPGA mediante la compilación EDIF. Esta FPGA está en una placa cuyas salidas son observables desde un analizador lógico. Las configuraciones son programadas mediante cuatro interruptores, más uno independiente de validación. Las nueve configuraciones que están preprogramadas para probar un determinado periférico independientemente del resto. Estas son:

1. El Watchdog timer.
2. El Timer0.
3. El Timer1.
4. El Timer2.
5. El periférico CCP en modo Capture.
6. El periférico CCP en modo Compare.
7. El periférico CCP en modo PWM.
8. El periférico SSP en modo SPI.
9. El periférico SSP en modo I^2C.

2.9 Archivo Watchdog.c

Este archivo fuente contiene la función **WATCHDOG()** que implementa al periférico Watchdog timer. Cuando el watchdog se desborda generará un reset general del sistema implementado mediante la función **RESET**() residente el archivo Testbench.c.

En este archivo voy a resaltar cómo son los procedimientos que he considerado óptimos para incrementar la cuenta en un contador cada ciclo de reloj. Para ello hay que distinguir dos casos. El primero (tipo I), es cuando el contador llega al tope de su capacidad de cuenta de manera que en el siguiente ciclo de reloj hay un desbordamiento. El segundo caso (tipo II), es el de un contador que tiene un límite superior de cuenta por debajo de su capacidad de cuenta máxima. Un ejemplo de estos dos casos se produce en el archivo Watchdog.c, **wdt** sería una variable del tipo I y **postscaler** del tipo II.

El procedimiento a seguir para el tipo I es el de declarar un proceso **par** en el que por un lado siempre se está incrementando la variable (ya que si se produce desbordamiento el valor de la variable en binario es 000...0) y por otro una sentencia **if** que compruebe si se ha alcanzado el valor binario máximo (1111...1) en cuyo caso se ejecutaría una acción (en el ejemplo se actualizaría el valor de la variable postscaler) y en caso negativo se ejecutaría un retraso (**delay**).

Para el tipo II tendremos directamente una sentencia **if** que cuestiona si el valor de la variable ha llegado a su tope superior. En caso afirmativo, declaramos un proceso **par** en el que por un lado se pone a cero la variable y por otro se ejecuta una acción pertinente (en el ejemplo activamos una interrupción) y, en caso negativo, incrementamos la variable.

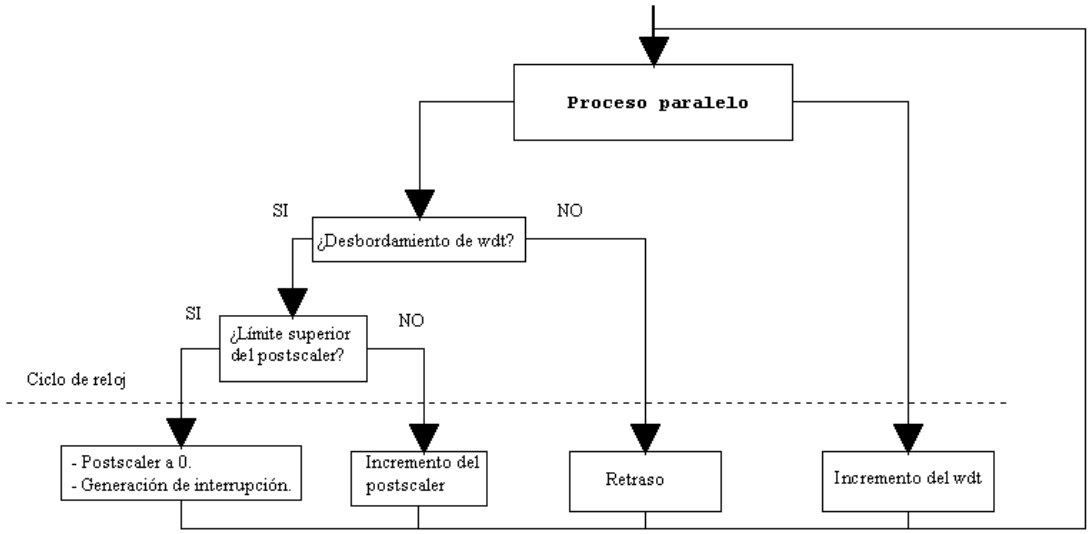


Figura 57. Diagrama de flujo de la cuenta en *Watchdog Timer*.

2.10 Archivo Timer0.c

Este archivo fuente contiene la función **TIMER0()** que implementa el periférico Timer0. En su interior está definida la macro **incr_tmr0()** que sirve para incrementar el *scaler* asignado a este periférico. Cuando el Timer0 se desborda provoca una interrupción señalada por el bit 0 del registro **inter** del Testbench.

2.11 Archivo Timer1.c

Este archivo fuente contiene la función **TIMER1()** que implementa el periférico Timer1. Dentro del código está definida la macro **incr_tmr1()** que se utiliza para incrementar el valor del prescaler y generar la interrupción asociada a este periférico. Cuando el Timer1 se desborda provoca una interrupción señalada por el bit 1 del registro **inter** del Testbench.

2.12 Archivo Timer2.c

Este archivo fuente contiene la función **TIMER2()** que implementa el periférico Timer2. Cuando el Timer2 se desborda provoca una interrupción señalada por el bit 2 del registro **inter** del Testbench.

2.13 Archivo Ccp.c

Este archivo fuente contiene la función **CCP()** que implementa el periférico *Capture/Compare/PWM*. En su interior están definidas las macros **lee_CCP1(),** que muestrea el pin CCP1 de entrada al periférico, y **incr_CCP1()** que incrementa el contenido de los registros **ccp1rl** y **ccp1rl**.

Cuando se produce una captura en modo *Capture* provocará una interrupción indicada por el bit 3 del registro **inter** del Testbench. Si estamos en modo *Compare*, cuando se produce la coincidencia de valores el **CCP** provocará una interrupción indicada por el bit 4 del registro **inter** ya citado.

2.14 Archivo Ssp.c

Este archivo fuente contiene la función **SSP()** que implementa el periférico *Synchronous Serial Port*. En el interior del código encontramos las siguientes macros:

RX_TX(): que realiza la recepción/transmisión de un byte tanto en el modo SPI como en el modo I²C.

ACK(): Esta macro genera un asentimiento en modo *slave* I²C.

RECIBE(): Que hace las tareas propias (generación de interrupciones, actualización de registros, reinicio de cuenta de contador de bits y copia de contenido del registro SSPSR en SSPBUF) tras la recepción de un byte en modo SPI y en modo I²C.

Master_I2C(): Que interactúa con el Testbench para comportarse como *master* en modo I²C.

Las interrupciones, todas ellas enmascarables, que se generan en este periférico son:

- Cuando el búfer está lleno, la número 5 del registro **inter**.
- La indicación del bit de start, la número 6 del registro **inter**.
- La indicación del bit de stop, la número 7 del registro **inter**.

CAPÍTULO VI:
FASE DE PRUEBAS

Capítulo VI. FASE DE PRUEBAS

Para la fase de pruebas contamos con una placa que contiene una FPGA Virtex de Xilinx y una serie de interruptores que nos permiten configurar los distintos periféricos dinámicamente. Para ello utilizamos el bloque inicio.c que cuenta con 9 posibles configuraciones que prueban respectivamente el Watchdog timer, Timer0, Timer1, Timer2, CCP en modo *Capture*, CCP en modo *Compare*, CCP en modo *PWM*, SSP en modo SPI y SSP en modo I²C. Para poder activar cada una de ellas en el bloque inicio.c se han definidos varias *interfaces* que permiten introducir datos desde los pines de la FPGA, como muestra el siguiente código, en el que la *interface* **interrup** se utiliza para programar el modo de funcionamiento y la *interface* **start** es para validar la programación:

```
...
#ifdef Placa
       interface port_in (unsigned 4 In) interrup() with {data={"P4","P5","P6","P7"}};
       interface port_in (unsigned 1 In) start() with {data= {"P14"}};
       unsigned 4 condicion;
       unsigned 1 ini;
#endif
...
```

Para observar las salidas que se producen, en el modo Placa se han definido previamente todas las señales de salida mediante *interfaces* que harán que podamos leer las señales de salida desde los pines de la FPGA.

Para poder programar la FPGA tenemos que hacer una compilación EDIF para que se genere el archivo *netlist* que servirá para este fin. Previamente hemos de facilitar en nº de pin por el que vamos a leer las correspondientes salidas. En esta fase encontramos numerosas dificultades ya que disponíamos por un lado de la versión DK1 de Celoxica que es una versión de evaluación y que, por tanto, no permite hacer compilaciones EDIF; por otro lado la versión beta de Handel-C 3.0, que sí dispone de este tipo de compilación, está en una fase de desarrollo un tanto precaria en lo que se refiere al tratamiento de los errores de compilación ya que en numerosas ocasiones no proporciona ninguna información de lo que hace

que la compilación sea errónea y, cuando sí lo hace, suele no corresponderse con el sitio indicado. Llegados a este punto entramos en una fase del proyecto en la que la depuración se hace completamente "a ojo de buen cubero" y entra en juego el factor suerte.

Una vez salvados todos los errores, conectamos los pines de salida con la entrada a un analizador lógico que dispusimos junto a la placa. En este analizador lógico pudimos comprobar la evolución de las salidas de los distintos periféricos pudiendo cambiar la configuración de éstos mediante los interruptores.

CAPÍTULO VII:
CONCLUSIONES

Capítulo VII. CONCLUSIONES

Con Handel-C en la versión 3.0 Beta de Embedded Solutions Limited y en la versión de evaluación DK1 Eval de Celoxica Limited, se nos muestra un entorno de programación de FPGAs completo y cómodo para el usuario en un lenguaje de programación que en mucho recuerda al C convencional.

Respecto a la versión anterior, v 2.1, además del entorno de programación con ventanas, se han incorporado y mejorado numerosas herramientas que facilitan y flexibilizan la programación. De entre las incorporaciones destaco el de las variables *signal*, que permiten leer un valor y escribirlo en un mismo ciclo de reloj; la posibilidad de introducir como señal de reloj una expresión interna, una expresión dividida por un determinado factor o una señal leída desde un puerto software; el uso de punteros, que hace más trasladable a Handel-C cualquier programa en ISO-C. Entre las mejoras destaca la posibilidad de crear puertos que conecten bloques Handel-C a otros escritos en otros lenguajes.

Si bien, a diferencia de la versión DK1 Eval, la versión 3.0 Beta permite la compilación EDIF del proyecto y, por consiguiente, su posterior prueba en una FPGA real, la DK1 resulta mucho más depurada en cuanto a la indicación de errores en la compilación, el seguimiento de hilos en paralelo en la fase de depuración de código (debug) y en diversos aspectos que hacen su utilización mucho más amable y eficaz. Además DK1 indica que se podrán hacer compilaciones a VHDL además de EDIF con lo que se podrán establecer comparaciones y se podrá conectar código escrito en ambos lenguajes sin ningún problema.

Si bien para la implementación *hardware* de algoritmos Handel-C puede resultar potentísimo, ya que basta sólo con trasladar código C a Handel-C para obtener una solución *hw*, en el debe de Handel-C encontramos que para

descripciones de *hardware* a nivel de bit este lenguaje tiene serias limitaciones (por ejemplo, no se puede direccionar un único bit de un registro para escritura, teniendo que actualizar todo el valor del mismo).

Como conclusión final he de decir que la experiencia programando Handel-C ha resultado positiva y personalmente me atrevo a decir que resulta tanto más conveniente frente a VHDL cuanto más complicado sea el algoritmo o ruta de datos que se vaya a implementar o, dicho de otro modo, cuanto más lejos de descripciones a nivel de bit se encuentre el diseño. En cuanto al entorno de programación esperemos que en futuras entregas del mismo se solventen los problemas anteriormente enunciados.

Capítulo VIII: Bibliografía.

Capítulo VIII. BIBLIOGRAFÍA

1. Documentos

- *Data sheet PIC16C9XX* (DS3044E), de Microchip Technology Inc.
- *Data sheet PIC16C5X* (DS30453C), de Microchip Technology Inc.
- *The I²C-bus specification* (version 2.1) January 2000, de Philips Semiconductors.
- *The I²C-bus and how to use it*, de Philips Semiconductors.
- Ayuda de Handel-C Versión 3.0 Beta, de Embedded Solutions Ltd.

2. Direcciones URL

- http://www.microchip.com/
- http://www.celoxica.com/
- http://www.embeddedsol.com/
- http://www.toolbox.xilinx.com/docsan/

Apéndice A: Especificaciones del bus I2C.

- APÉNDICE A -

ESPECIFICACIONES DEL BUS I²C

A.1. Introducción.

En algunas aplicaciones, especialmente las que emplean microcontroladores mono-chip, se deben establecer ciertos criterios:

Cierto tipo de microcontroladores no es ampliable desde el exterior por lo que la aplicación se tiene que ceñir al tipo y cantidad de elementos que contenga en su interior: memoria RAM de datos, EEPROM, líneas de E/S, temporizadores, contadores, etc. Si estos no son suficientes habrá que elegir otro modelo de microcontrolador con más prestaciones o, emplear un sistema ampliable externamente.

Normalmente un sistema completo además del microcontrolador, necesita de otros circuitos periféricos como memorias, ampliación de las líneas de E/S, convertidores A/D Y D/A, relojes en tiempo real, controladores de display, de LCD, etc.

El coste de conexión de todos estos periféricos con el microcontrolador debe reducirse al mínimo.

Normalmente la aplicación no requiere una gran velocidad de transferencia entre el microcontrolador y los distintos dispositivos.

La eficacia general del sistema depende de la versatilidad de los dispositivos conectados al mismo.

A.2. Concepto del bus I²C

Está formado por dos líneas, SDA (datos) y SCL (reloj), que transportan la información entre los diferentes dispositivos conectados al bus. Cada dispositivo se identifica por una única dirección y puede trasmitir o recibir dependiendo de la operación que se vaya a realizar. Un controlador de LCD, por ejemplo, sólo recibe mientras que una memoria de tipo RAM puede transmitir o recibir datos en función de que se vaya a leer o escribir.

Los dispositivos pueden clasificarse en MASTER (principal) o SLAVE (secundario). El MASTER es el que inicia la transferencia de datos y genera la señal de reloj, cualquiera de los dispositivos direccionados por un MASTER, se considera un SLAVE.

El I^2C es un bus MULTIMASTER. Puede haber más de un Master conectado y controlando el bus. Normalmente se tratan de microcontroladores o microcomputadores.

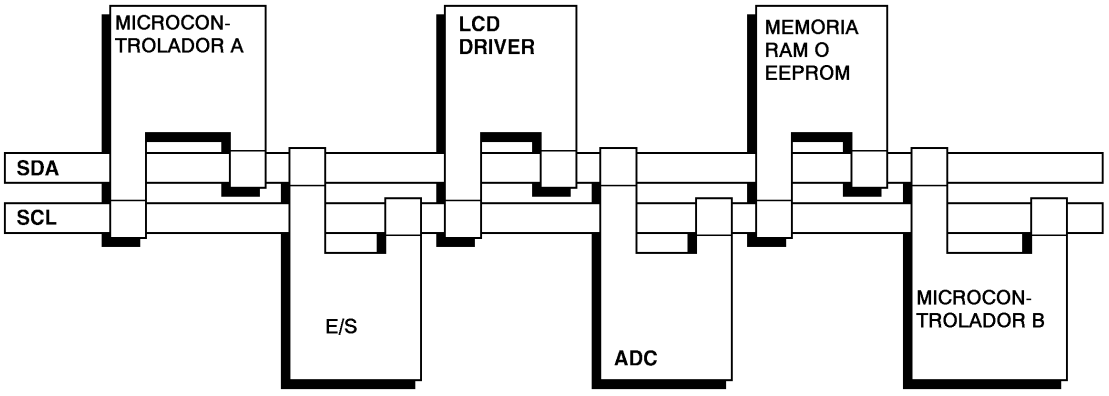


Figura 58. Conexiones al bus I²C.

Supongamos que en un momento dado el microcontrolador A actúa como MASTER:

1. Envía información al B
    a) A, emisor, direcciona a B
    b) A, emisor, envía el dato a B, receptor
    c) A termina la trasferencia

2. Recepción desde el B
    a) A, emisor, direcciona a B
    b) A, receptor, recibe el dato desde el B, transmisor
    c) A, termina la trasferencia

Esta configuración puede no ser permanente. En otro momento puede ser el B quien actúe de MASTER. Todo depende de la dirección del dato a transferir en un momento dado.

Dada la posibilidad de que existan varios MASTER, podría ocurrir que más de uno inicie la trasferencia en un mismo instante de tiempo. Para evitar el caos que puede producir esta situación, se ha desarrollado un sistema de arbitraje del bus. El procedimiento consiste en una conexión tipo AND entre todos los dispositivos conectados al bus.

Cuando uno o más MASTERs colocan información en la línea SDA, verifican si el bit que ellos sacan coincide con el nivel lógico de dicha línea. Si un MASTER saca un “1” pero la línea SDA está a “0” coincidiendo con un pulso de reloj, pierde la posesión del bus cancelando desde ese momento la transmisión. El nivel lógico “0” (bit dominante) presente en la línea SDA procederá de un MASTER. Ver figura.

Las señales de reloj durante el arbitraje del bus son una combinación entre las señales de clock de los distintos MASTERs conectados entre sí a la línea SCL mediante una conexión tipo AND.

A.2.1 Terminología del bus I^2C.

TRANSMISOR:	Dispositivo que coloca información en la línea SDA.
RECEPTOR:	Dispositivo que recibe información por la línea SDA.
MASTER:	Dispositivo que inicia la transferencia, genera la señal de reloj y finaliza la transferencia.
SLAVE:	Dispositivo seleccionado por el MASTER.
MULTI-MASTER:	Cuando más de un MASTER puede hacerse con el control del bus sin corromper la información.
ARBITRAJE:	Procedimiento que garantiza que, si más de un MASTER acceden simultáneamente al bus, únicamente uno de ellos se hace cargo del mismo con objeto de no alterar la información.
SINCRONIZACIÓN:	Procedimiento para sincronizar la señal de reloj en un sistema MULTIMASTER.

A.3. Características generales.

Tanto las líneas SDA como SCL son líneas bidireccionales que se conectan a +Vdd mediante resistencias de carga pull-up tal y como se muestra en

la figura. La sección de SCLKNx sólo es necesaria en los dispositivos que puedan funcionar cómo MASTER.

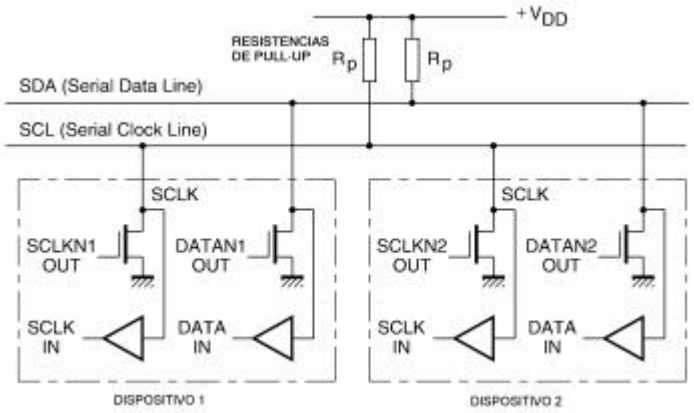


Figura 59. Conexión de SDA y SCL al bus.

Cuando el bus está libre ambas líneas están a nivel lógico “1”. Los transistores de salida conectados a las líneas del bus I^2C deben ser de colector abierto para que todos ellos se puedan conectar entre sí formando una conexión tipo AND .Los bits de datos sobre el bus pueden transferirse a una velocidad de 100 Kbits/s. La capacidad máxima en el bus es de 400 pF y el número de dispositivos conectados no debe superarla.

A.4. Transferencia del bit.

Debido a la variedad de tecnología empleada en los dispositivos diseñados para conectarse al bus I^2C (CMOS, NMOS, TTL, etc.) los niveles lógicos “0” y “1” de los bits transferidos, no tienen una tensión fija sino que dependen de la tensión Vdd de alimentación. Cada bit que se transfiere por la línea SDA debe ir acompañado de un pulso de reloj por la línea SCL.

A.4.1 Validez del bit.

El bit de datos trasferido por la línea SDA debe mantenerse estable durante el periodo en que la señal de reloj está a nivel "1". La línea de datos SDA sólo puede cambiar de estado durante el periodo en que la señal de reloj esté a "0", tal y como se muestra en la figura.

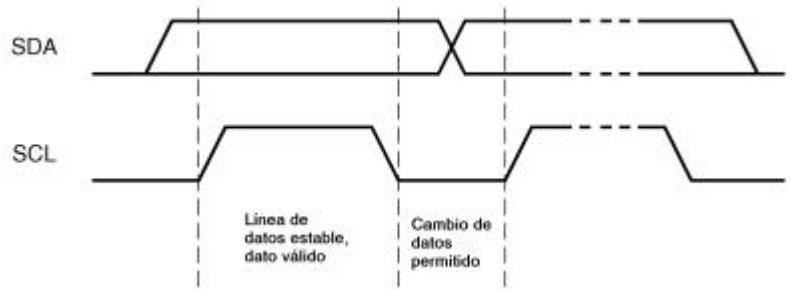


Figura 60. Validación del bit de datos

A.4.2 Condiciones de inicio (START) y parada (STOP).

Existen dos situaciones únicas que son definidas como las condiciones de inicio (START) y parada (STOP) que determinan el inicio y final de toda trasferencia de datos entre el MASTER y el SLAVE, dichas situaciones se caracterizan por cambios en la línea SDA permaneciendo a nivel alto la señal SCL para distinguirlas así de la transferencia normal de datos.

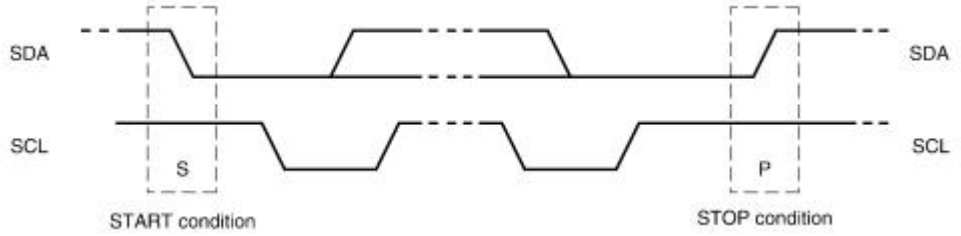


Figura 61. Condiciones de START y STOP

Una transición de “1” a “0” (flanco descendente) en la línea SDA al tiempo que SCL está a nivel lógico “1”, determina la condición de START. Una transición de “0” a “1 (flanco ascendente) sobre la línea SDA al tiempo que SCL permanece a nivel “1” determina la condición de STOP.

Ambas condiciones son siempre generadas por el MASTER que, en todo memento, inicia y finaliza toda transferencia. El bus se considera ocupado (BUSY) tras la condición de START y, se considera libre cierto tiempo después de la condición de STOP.

A.5. Transfiriendo datos.

A.5.1 Formato del byte

Todos los bytes colocados sobre la línea SDA deben constar de 8 bits. Cada byte va seguido de un bit de asentimiento, ACK, y comienza siempre por el bit de más peso (MSB) tal y como se muestra en la figura.

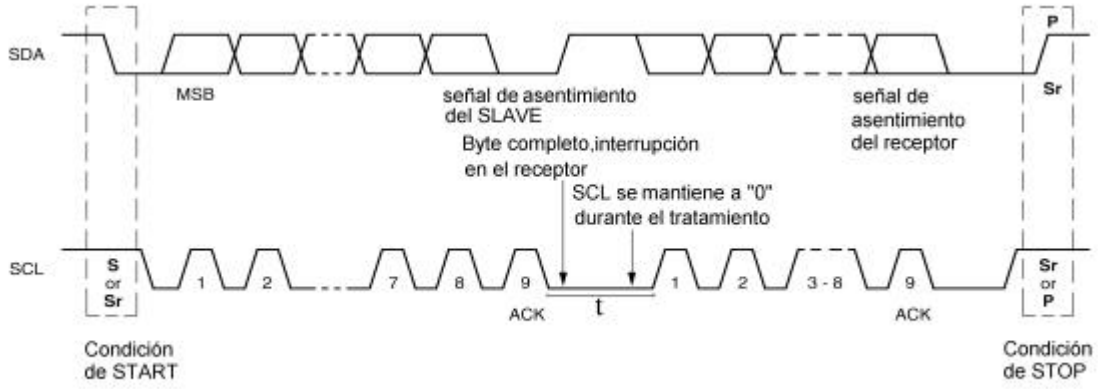


Figura 62. Formato de un byte.

Entre un byte y el siguiente puede haber un tiempo “t” de espera que se puede aprovechar para, por ejemplo, dar servicio a una interrupción. En este caso

el receptor mantiene la línea SCL a “0” para forzar al transmisor a un estado de espera.

Un mensaje cualquiera puede finalizar mediante la generación de la condición de STOP durante la transferencia de un byte. En este caso no se generará el bit de asentimiento ACK.

A.5.2 Asentimiento.

El bit de asentimiento ACK es obligatorio en la transferencia de cada byte. El pulso de reloj asociado a este bit (9º) lo genera el MASTER. El transmisor pone la línea SDA A “1” durante dicho pulso de reloj.

El receptor por su parte pone a “0” la línea SDA durante el pulso de reloj correspondiente al ACK y lo mantiene estable durante todo el periodo de dicho pulso.

En la línea SDA prevalece, por tanto, el nivel “0” como se aprecia en la figura.

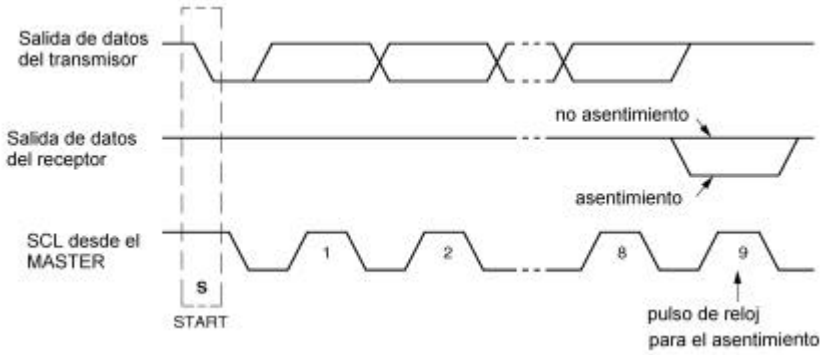


Figura 63. Generación de un asentimiento ACK.

Normalmente el receptor que ha sido direccionado está obligado a generar el bit ACK después de recibir cada byte. Cuando un receptor slave no

genera el bit ACK (NACK) al ser direccionado (por ejemplo cuando está realizando una función interna y no está dispuesto a recibir), debe mantener la línea SDA a nivel "1" durante el bit ACK (NACK). Esta situación es detectada por el MASTER que debe generar la condición de STOP y abortar así la trasferencia.

Igualmente si un SLAVE receptor reconoce ser direccionado pero no está dispuesto a recibir más bytes, el MASTER debe abortar la trasferencia. Esta situación se identifica porque el SLAVE tampoco genera el ACK en el siguiente byte que recibe. En su lugar pone la línea SDA a "1" (NACK) y el MASTER genera la condición de STOP.

Si un MASTER está recibiendo debe generar ACK tras cada byte enviado por el SLAVE excepto en el último, en el que el bit ACK lo mantiene a "1" (NACK) y genera la condición de STOP.

A.6. Arbitraje y sincronización del reloj.

Es la técnica mediante la cual, en un sistema donde existan varios MASTER, se evitan conflictos en el bus y pérdidas de información.

A.6.1 Sincronización.

Todos los MASTER generan una señal de reloj sobre la línea SCL para transferir mensajes sobre el bus I^2C. Los bits de información son válidos cuando dicha señal de reloj está a nivel lógico "1".

La sincronización del reloj se realiza mediante una conexión tipo AND de todos los MASTER existentes en el bus.

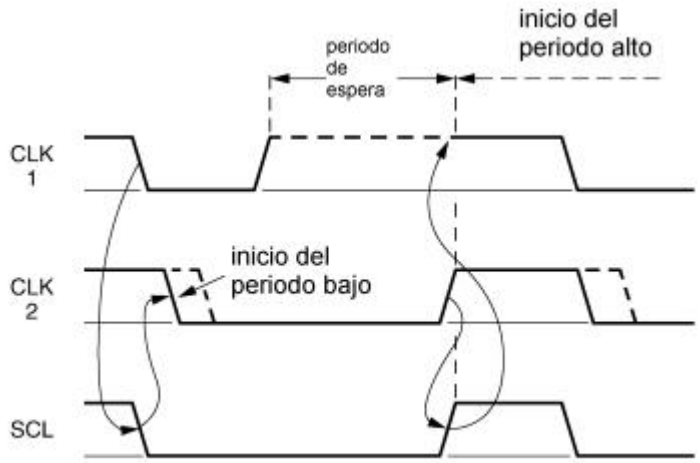


Figura 64. Sincronización del reloj.

Cuando un MASTER cualquiera, por ejemplo el 1, genera un flanco descendente en CLK1, la línea SCL pasa a “0”. Todos los dispositivos comienzan a temporizar sus respectivos periodos bajos de reloj.

Sin embargo un flanco ascendente en CLK1 no cambia el estado de SCL ya que en ese instante CLK2 está a “0”.

La línea SCL se mantiene a “0” durante el periodo de aquel dispositivo cuyo nivel “0” sea más largo. El resto de dispositivos entran en estado de espera.

Cuando todos los dispositivos finalizan su periodo bajo, SCL pasa a “1”. De esta forma todos inician el periodo alto al mismo tiempo. El primer dispositivo en finalizar este periodo alto, pone SCL a “0” y el proceso se repite.

La señal de reloj presente en la línea SCL queda sincronizada y...

El periodo bajo: Se determina por aquel cuyo periodo bajo sea el de mayor duración.

El periodo alto: Es determinado por aquel dispositivo cuyo periodo alto sea es de menor duración.

A.6.2 Arbitraje.

El arbitraje consiste en determinar qué MASTER se hace cargo del bus en un sistema MULTIMASTER. Cada MASTER debe comparar si el bit de datos que trasmite junto con su pulso de reloj en un momento dado, coincide con el nivel lógico presente en la línea de datos SDA. De no ser así, pierde el acceso al bus. Si un MASTER saca un nivel lógico "1" al tiempo en que otro saca un "0", en la línea SDA prevalece el segundo (bit "dominante"), por lo que el primero debe dejar el bus libre al menos hasta que se detecte la condición de STOP generada por el MASTER que se hizo cargo de dicho bus. Una situación de conflicto sólo puede producirse cuando el bus está libre y dos o más MASTERs tratan de transmitir, esta situación no se produce cuando el bus ya está ocupado.

La siguiente figura muestra el caso de dos MASTER. En el momento en que hay diferencia entre el nivel lógico interno del MASTER que genera el DATO1 y el nivel actual presente en la línea SDA, éste pierde el derecho al usar el bus y su nivel interno pasa a "1" liberando así la línea SDA.

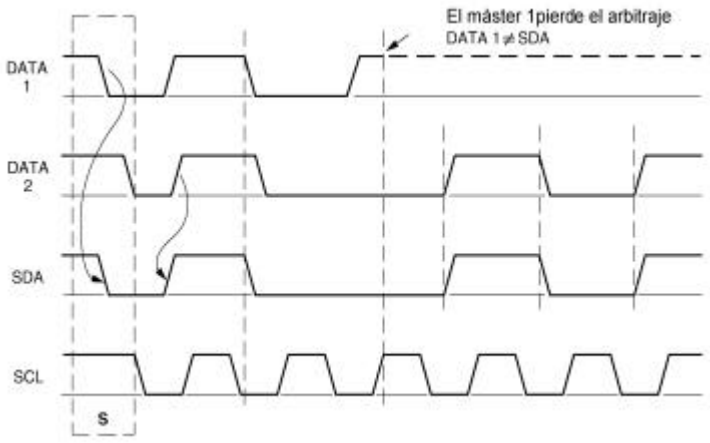


Figura 65. Arbitraje del bus.

La comparación para determinar el arbitraje puede continuar en varios bits hasta que haya una falta de coincidencia.

El MASTER que pierde el arbitraje puede seguir generando pulsos de reloj hasta que finalice el byte en que lo perdió.

Si un MASTER pierde el arbitraje en el momento en que está direccionando a un SLAVE, es posible que el MASTER "ganador" lo estuviera direccionando precisamente a él. Es por ello que el MASTER "perdedor" se debe poner inmediatamente en modo receptor y como SLAVE.

A.7. Formato

Los datos que se transfieren tienen el formato mostrado en la figura.

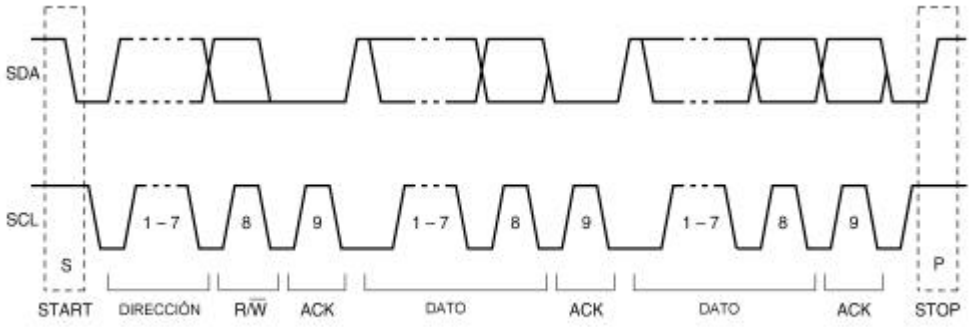


Figura 66. Formato de los datos transferidos.

Tras la condición de START, el MASTER envía la dirección del SLAVE al que se quiere dirigir. Esta dirección es de 7 bits más el bit R/W que indica lectura (R/W=0) del SLAVE. Toda transferencia de datos finaliza con la condición de STOP también generada por el MASTER. A pesar de todo, si un MASTER aún desea comunicar por el bus, puede generar otra condición de START y direccionar a otro SLAVE sin generar previamente la condición de STOP.

En las figuras se muestran posibles formatos de transferencia.

El MASTER transmite al SLAVE receptor. No cambia el byte de dirección.

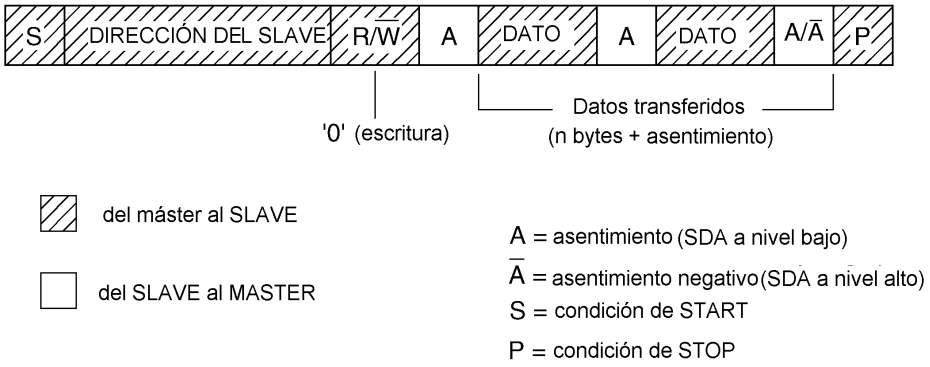


Figura 67. Master escribe en el receptor.

El MASTER recibe desde el SLAVE después de enviarle a éste la dirección.

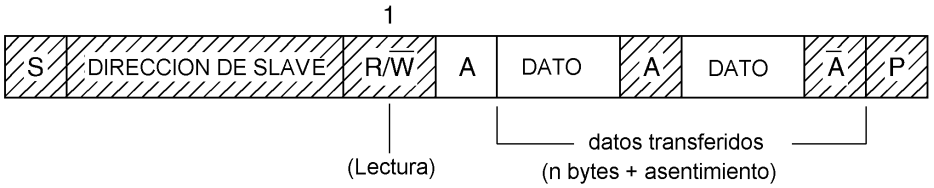


Figura 68. Master lee en el receptor.

Tras el primer ACK enviado por el SLAVE el MASTER transmisor se convierte en receptor y el SLAVE receptor en transmisor.

El MASTER cambia el byte de dirección y selecciona un SLAVE distinto.

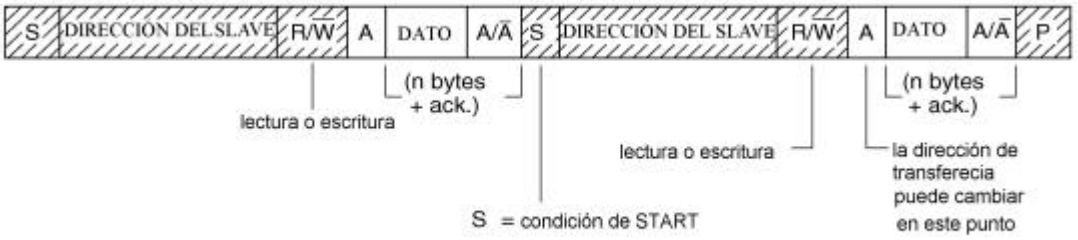


Figura 69. Cambio de SLAVE direccionado.

NOTAS:

- El formato combinado de la figura permite seleccionar un SLAVE para leerlo, por ejemplo, y luego para escribirlo. En el caso de un dispositivo I^2C de tipo memoria RAM puede ser útil.

- El tipo de datos o comandos que se envían pueden servir para realizar distintos tipos de operaciones según el dispositivo I^2C empleado. Habrá que consultar las características y posibilidades de cada cual.

A.8. Direccionamiento

El proceso de direccionamiento del bus I^2C consiste en que el primer byte que envía el MASTER tras la condición de START es un código que determina y selecciona a un determinado SLAVE. Existe una excepción en el código denominado “llamada general”. Cuando un MASTER realiza una “llamada general” (código 0000 000) todos los SLAVES existentes deben responder con el bit ACK. Puede darse el caso de dispositivos que ignoren esta llamada. El siguiente byte tras la misma indicará la operación a realizar.

A.8.1 Definición del primer byte

Los 7 bits de más peso del primer byte se emplean para direccionar a un determinado SLAVE. El de menos peso, el octavo, determina se realizará una

operación de lectura o de escritura (R/W) sobre el SLAVE direccionado, tal y como se muestra en la figura.

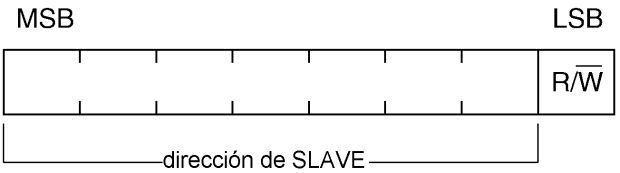


Figura 70. Direccionamiento de un SLAVE.

Cuando el MASTER envía una dirección por el bus, todos los SLAVE conectados la comparan con la suya propia interna. Aquel que coincida se considerará seleccionado por el MASTER.

De los 7 bits que forman una dirección hay una parte que son fijos y ya están definidos internamente por el propio dispositivo SLAVE y otra parte, que son programables. De esta forma es posible conectar al mismo bus dispositivos idénticos cuya parte fija de dirección es la misma pero la programable es diferente.

El número de dispositivos iguales que se pueden instalar a un mismo bus depende del número de bits programables disponibles. Por ejemplo, sin un determinado modelo de dispositivo recibe una dirección de 7 bits, 4 de los cuales son fijos y los 3 restantes programables, se podrán conectar al bus un máximo de 8 dispositivos de ese mismo modelo.

El comité I^2C creado al efecto coordina las direcciones asignadas a los diferentes dispositivos. La dirección 1111 111 está reservada como extensión de dirección en el que el proceso de direccionamiento de un dispositivo continua en los siguientes bytes transferidos por el MASTER. La dirección 1111 XXX está reservada también con propósitos de direccionamientos extendidos y la emplean ciertos dispositivos que tienen 10 bits de direccionado. La dirección 0000 XXX está reservada para un grupo especial de dispositivos:

DIRECCIÓN	R/W	FUNCIÓN
0000 000	0	Dirección de “llamada general”(1)
0000 000	1	Byte de inicio(2)
0000 001	X	Dirección CBUS(3)
0000 010	X	Reservada para un formato diferente(4)
0000 1XX	X	Hs-mode master code
1111 1XX		Reservada para futuros usos
0000 100	X	No definida
0000 101	X	No definida
0000 110	X	No definida
0000 111	X	No definida

NOTAS:

1. Para aquellos dispositivos que la admitan y/o necesiten

2. No está permitido el bit ACK tras la recepción del byte. Empleado por ciertos dispositivos.

3. La dirección CBUS está reservada para permitir la combinación de dispositivos I^2C y CBUS en un mismo sistema. Los dispositivos I^2C no responden a esta dirección.

4. Esta dirección está reservada para combinar el formato I^2C con otros protocolos. Únicamente los dispositivos I^2C que estén diseñados para trabajar con esos formatos y protocolos responden a esta dirección.

A.9. Especificaciones eléctricas

El bus I^2C permite la comunicación entre dispositivos fabricados con diferentes tecnologías y diferentes tensiones de alimentación. Para conexiones con niveles de entrada fijos y alimentación de +5V, se definen los siguientes valores:

V_{IL} = 1.5V (máxima tensión de entrada a “0”)
V_{IH} = 3V (mínima tensión de entrada a “1”)

Dispositivos capaces de trabajar con un rango variado de tensiones de alimentación (p.e. CMOS) se definen los siguientes niveles:

V_{IL} = 0.3 Vdd (máxima tensión de entrada a “0”)
V_{IH} = 0.7 Vdd (mínima tensión de entrada a “1”)

En ambos casos, la máxima tensión de salida a “0” es:

$V_{OL} = 0.4 V_{DD}$ (V)

La corriente máxima de entrada a “0” en las líneas SDA y SCL de un dispositivo compatible con el bus es de –10 μA. A nivel “1” la corriente de entrada es de 10 μA. La capacidad de dichas líneas es de 10 pF máximo.

Los dispositivos con niveles fijos de entrada pueden alimentarse desde diferentes fuentes de alimentación como se muestra en la figura.

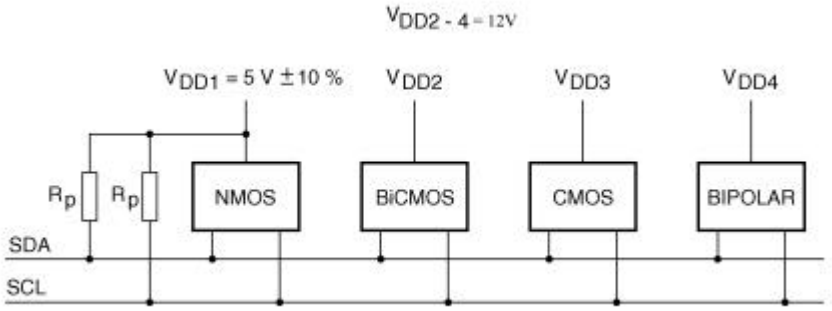


Figura 71. Conexión al bus I²C de distintos dispositivos.

Los dispositivos I^2C cuyos niveles lógicos de entrada están en relación a Vdd, deben alimentarse conjuntamente según figura.

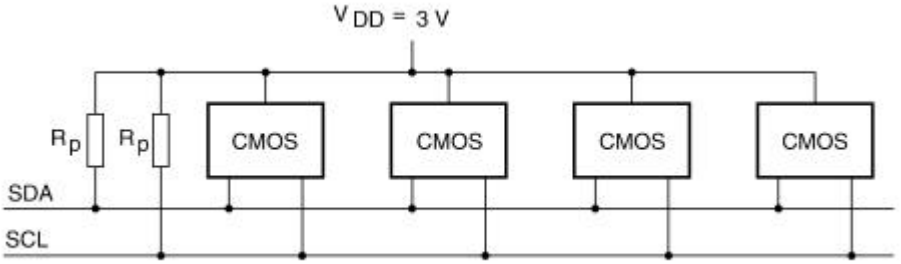


Figura 72. Conexión al bus I²C de dispositivos CMOS.

Cuando se combinan ambos tipos de dispositivos, los de niveles fijos de entrada (NMOS, TTL) y los dependientes de Vdd (CMOS), estos últimos deban conectarse a una alimentación común de +5V como se muestra en la figura.

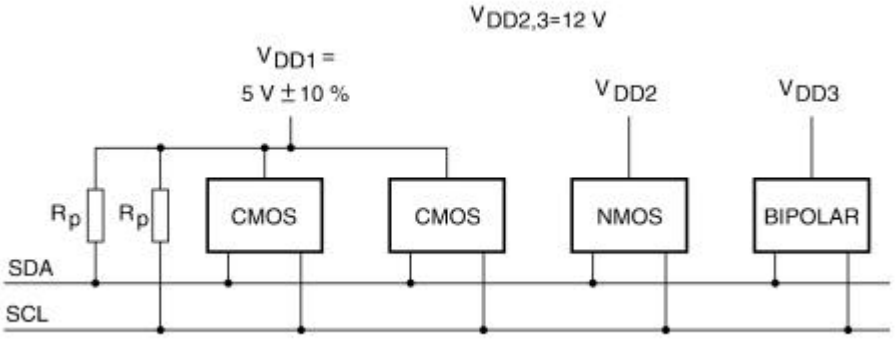


Figura 73. Conexión de dispositivos con distinta Vdd

La capacidad máxima del bus es de 400 pF que incluye la capacidad entre cables y la de los dispositivos conectados.

A.10. Tiempos

La señal de reloj en el bus I^2C tiene un periodo de tiempo a nivel "0" de como mínimo 4.7µS y a "1" de como mínimo 4µS. De esta forma el MASTER puede generar una frecuencia de hasta 100khz. La figura muestra un diagrama de tiempos típico.

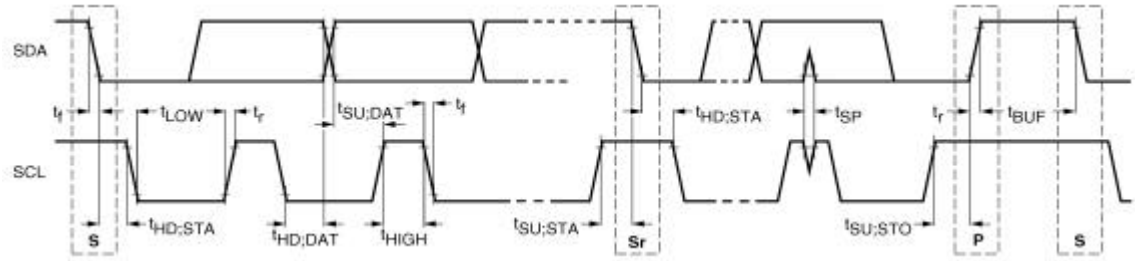


Figura 74. Diagrama de tiempos

SÍMBOLO	PARÁMETRO	MÍNIMO	MÁXIMO	UNIDAD
f_{SCL}	Frecuencia del reloj		100	kHz
t_{BUF}	Tiempo en que el bus debe estar libre antes de iniciar una nueva transmisión	4.7		µS
f_{HD};STA	Tiempo entre el START y primer pulso en SCL	4		µS
t_{LOW}	Duración del periodo bajo del reloj	4.7		µS
t_{HIGH}	Duración del periodo alto del reloj	4		µS
t_{SU}; STA	Tiempo antes de una nueva condición de START	4.7		µS
t_{HD};DAT	Tiempo del mantenimiento del dato	0		µS

t_{SU}; DAT	Tiempo de puesta de dato	250		nS
t_R	Tiempo del flanco ascendente para las líneas SDA y SCL		1	µS
t_F	Tiempo del flanco descendente para las líneas SDA y SCL		300	nS
T_{SU}; STO	Tiempo para la condición de STOP	4.7		µS

APÉNDICE B:
CÓDIGO FUENTE HANDEL-C

- APÉNDICE B -

CÓDIGO FUENTE HANDEL-C

B.1. Archivo cabecera: top.h

```
/*****************************************************************************
*                                                                            *
*                                  INCLUSIONES                               *
*                                                                            *
*****************************************************************************/

#define Simul
#undef Simul
#define Placa
#undef Placa
#define Real
#undef Real
#define Rst
#undef Rst
#define Watchdog
#undef Watchdog
#define Timer0
#undef Timer0
#define Timer1
#undef Timer1
#define Timer2
#undef Timer2
#define Ccp
#undef Ccp
#define Ssp
#undef Ssp
```

B.2. Archivo cabecera: cabecera.h

```
/*****************************************************************************
*                                                                            *
*                                   cabecera.h                               *
*                                                                            *
*****************************************************************************/
#include <stdlib.h>
#include "top.h"

/*****************************************************************************
*                                                                            *
*                     CONJUNTO DE CONSTANTES GENERALES                       *
*                                                                            *
*****************************************************************************/
#define TRUE 1
#define FALSE 0
#define W_REG 8
#define W_DIR 5
#define W_CORTO 4
#define W_BIT 1

/*****************************************************************************
*                                                                            *
*        CONJUNTO DE DEFINICIONES DE CONSTANTES PARA EL DIRECCIONAMIENTO     *
*                                                                            *
*           DE LOS REGISTROS EXTERNOS USADOS POR LOS PERIFÉRICOS             *
*                                                                            *
*****************************************************************************/

#define CONFIGH        0
#define CONFIGL        1
#define STATUS         2
#define OPTION         3
#define T1CON          4
#define T2CON          5
#define PR2            6
#define CCP1CON        7
#define CCPR1L         8
#define TRISC          9
#define SSPCON         10
#define SSPSTAT        11
#define SSPBUF         12
#define SSPADDH        13
#define SSPADDL        14
#define MASK           15
#define INTER          16

#define TAM_STACK      17

/*****************************************************************************
*                                                                            *
*        CONJUNTO DE DEFINICIONES DE CONSTANTES PARA EL DIRECCIONAMIENTO     *
*                                                                            *
*           DE LAS INTERRUPCIONES EN LA RUTINA DE INTERRUPCIÓN               *
*                                                                            *
*****************************************************************************/

#define INT_TMR0    0
#define INT_TMR1    1
#define INT_TMR2    2
#define INT_CCP1    3
#define INT_CCP2    4
#define INT_SSP1    5
#define INT_SSP2    6
#define INT_SSP3    7
```

```
/*****************************************************************************
*                                                                            *
*          CONJUNTO DE DEFINICIONES DE REGISTROS DEL TESTBENCH               *
*                                                                            *
*****************************************************************************/

extern unsigned W_REG configh;
extern unsigned W_REG configl;
extern unsigned W_REG status;
extern unsigned W_REG option;
extern unsigned W_REG t1con;
extern unsigned W_REG t2con;
extern unsigned W_REG pr2;
extern unsigned W_REG ccp1con;
extern unsigned W_REG ccpr1l;
extern unsigned W_REG trisc;
extern unsigned W_REG sspcon;
extern unsigned W_REG sspstat;
extern unsigned W_REG sspbuf;
extern unsigned W_REG sspaddh;
extern unsigned W_REG sspaddl;
extern unsigned W_REG mask;
extern unsigned W_REG inter;

/*****************************************************************************
*                                                                            *
*               CONJUNTO DE DEFINICIONES DE VARIABLES EXTERNAS               *
*                                                                            *
*****************************************************************************/

extern unsigned W_BIT r_w;                /*Línea de lectura-escritura del TESTBENCH*/
extern unsigned W_REG scaler_tmr0;       /*scaler_tmr0 referenciado por nombre*/
extern unsigned W_CORTO scaler_tmr1;
extern unsigned W_REG prescal_tmr2;
extern unsigned W_CORTO postscal_tmr2;
extern unsigned W_REG tmr2;            /* Será referenciado por nombre por CCP y SSP*/
extern unsigned W_BIT nuevo_dato;
extern unsigned W_REG sspsr with {base=2};
extern unsigned W_REG tmr1h;
extern unsigned W_REG tmr1l;
extern unsigned W_REG tmr2;

/*****************************************************************************
*                                                                            *
*               CONJUNTO DE DEFINICIONES DE CANALES DEL TESTBENCH            *
*                                                                            *
*****************************************************************************/

extern chan unsigned W_BIT bus_int[W_REG];    /*Canal para las interrupciones*/
#ifndef Real
extern chan unsigned W_REG busin;         /*Canal de entrada de datos del testbench*/
extern chan unsigned W_REG busout;           /*Canal de salida de datos del testbench*/
extern chan unsigned W_DIR addr;         /*Canal para direccionamiento del testbench*/
#endif
```

B.3. Archivo cabecera: milib.h

```
/*****************************************************************************
*                                                                            *
*                                    milib.h                                 *
*                                                                            *
*****************************************************************************/
#include "cabecera.h"
#include "top.h"

/*****************************************************************************
*                                                                            *
*                      PROTOTIPOS DE EXPRESIONES                             *
*                                                                            *
*****************************************************************************/

extern macro expr setbit(b,e);

extern macro expr clrbit(b,e);

#ifndef Real
/*****************************************************************************
*                                                                            *
*                      PROTOTIPOS DE PROCEDIMIENTOS                          *
*                                                                            *
*****************************************************************************/


extern macro proc escribe(REG,DATO);

extern macro proc lee(REG,w);

#endif
```

B.4. Archivo librería: milib.c

```
/*****************************************************************************
*                                                                            *
*                                  milib.c                                   *
*                                                                            *
*****************************************************************************/

#include "cabecera.h"
#include "top.h"

/*****************************************************************************
* Nombre de la expresión:       setbit(b,e)                                  *
*                                                                            *
* Parámetros :     b        Entero sin signo que representa la posición de un bit   *
*                   e        Variable o registro    de cualquier longitud           *
*                                                                            *
* Descripción:    Pone a 1 el bit b del registro e sin alterar el ancho de éste.    *
*                                                                            *
* Duración:        1 ciclo de reloj.                                          *
*                                                                            *
* Ejemplo:         cuenta=setbit(7,cuenta)                                    *
*                                                                            *
*                Pone a 1 el bit 7 de la variable cuenta.                    *
*                                                                            *
*****************************************************************************/

macro expr setbit(b,e)=select(width(e)==1,(unsigned 1)1,
    (((width(e)-1)==b)?1@((e)<-(width(e)-1)):
    e[width(e)-1]@setbit(b,((e)<-(width(e)-1)))));

/*****************************************************************************
* Nombre de la expresión:       clrbit(b,e)                                  *
*                                                                            *
* Parámetros :     b        Entero sin signo que representa la posición de un bit   *
*                  e        Variable o registro de cualquier longitud                *
*                                                                            *
* Descripción:     Pone a 0 el bit b del registro e sin alterar el ancho de éste.   *
*                                                                            *
* Duración:        1 ciclo de reloj.                                          *
*                                                                            *
* Ejemplo:         status=clrbit(4,status)                                    *
*                                                                            *
*                 Pone a 0 el bit 4 del registro status.                     *
*                                                                            *
*****************************************************************************/

macro expr clrbit(b,e)=select(width(e)==1,(unsigned 1)0,
    (((width(e)-1)==b)?0@((e)<-(width(e)-1)):
    e[width(e)-1]@clrbit(b,((e)<-(width(e)-1)))));

#ifndef Real
/*****************************************************************************
* Nombre del procedimiento:        escribe(REG,DATO)                         *
*                                                                            *
* Parámetros :    REG        Constante asociada a la dirección de un registro      *
*                dato    Constante de ancho W_REG.                           *
*                                                                            *
* Descripción:    Escribe el valor DATO en el registro asociado a la dirección     *
*                REG utilizando el canal busin.                              *
*                                                                            *
* Duración:       1 ciclo de reloj.                                          *
*                                                                            *
* Ejemplo:        escribe(OPTION,0x3A)                                       *
*                                                                            *
*                Escribe la cadena 0b00111010 en el registro option          *
*                                                                            *
*****************************************************************************/
```

```
macro proc escribe(REG,DATO)
{
    par
    {
        r_w     = FALSE;
        addr ! REG;
        busin ! DATO;
    }
}

/*****************************************************************************
* Nombre del procedimiento:      lee(REG,w)                                  *
*                                                                            *
* Parámetros :   REG        Constante asociada a la dirección de un registro *
*                w        Entero sin signo de ancho W_REG.                  *
*                                                                            *
* Descripción:   Lee el valor actual del registro asociado a la dirección REG *
*                utilizando el canal    busout y lo    almacena en la variable w.  *
*                                                                            *
* Duración:          1 ciclo de reloj.                                        *
*                                                                            *
* Ejemplo:           lee(OPTION,aux)                                         *
*                                                                            *
*                    Escribe el valor del registro option en la variable aux *
*                                                                            *
*****************************************************************************/

macro proc lee(REG,w)
{
    par
    {
        r_w  = TRUE;
        addr ! REG;
        busout ? w;
    }
}

#endif
```

B.5. Archivo fuente: Testbench.c

```
/*****************************************************************************
*                                                                            *
*                                  Testbench.c                               *
*                                                                            *
*****************************************************************************/

#include <stdlib.h>
#include "milib.h"
#include "top.h"

/*****************************************************************************
*                                                                            *
*                    CONJUNTO DE CONSTANTES GENERALES                        *
*                                                                            *
*****************************************************************************/

#define TRUE 1
#define FALSE 0
#define W_REG 8
#define W_DIR 5
#define W_CORTO 4
#define W_BIT 1

/*****************************************************************************
*                                                                            *
*        CONJUNTO DE DEFINICIONES DE CONSTANTES PARA EL DIRECCIONAMIENTO     *
*                                                                            *
*            DE LOS REGISTROS EXTERNOS USADOS POR LOS PERIFÉRICOS            *
*                                                                            *
*****************************************************************************/

#define CONFIGH            0
#define CONFIGL            1
#define STATUS             2
#define OPTION             3
#define T1CON              4
#define T2CON              5
#define PR2                6
#define CCP1CON            7
#define CCPR1L             8
#define TRISC              9
#define SSPCON             10
#define SSPSTAT            11
#define SSPBUF             12
#define SSPADDH            13
#define SSPADDL            14
#define MASK               15
#define INTER              16

#define TAM_STACK          17

/*****************************************************************************
*                                                                            *
*        CONSTANTES PARA EL DIRECCIONAMIENTO DE LAS INTERRUPCIONES           *
*                  EN LA RUTINA DE INTERRUPCIÓN                              *
*             Y BIT ASOCIADO DEL    REGISTRO DE INTERRUPCIÓN                 *
*                                                                            *
*****************************************************************************/

#define INT_TMR0    0
#define INT_TMR1    1
#define INT_TMR2    2
#define INT_CCP1    3
#define INT_CCP2    4
#define INT_SSP1    5
#define INT_SSP2    6
```

```
#define INT_SSP3    7

/*****************************************************************************
*                                                                            *
*            CONJUNTO DE DEFINICIONES DE REGISTROS EXTERNOS DEL TESTBENCH     *
*                                                                            *
*****************************************************************************/

unsigned W_REG configh with {base=2};
unsigned W_REG configl with {base=2};
unsigned W_REG status with {base=2};
unsigned W_REG option with {base=2};
unsigned W_REG t1con with {base=2};
unsigned W_REG t2con with {base=2};
unsigned W_REG pr2 with {base=10};
unsigned W_REG ccp1con with {base=2};
unsigned W_REG ccpr1l with {base=10};
unsigned W_REG trisc with {base=2};
unsigned W_REG sspcon with {base=2};
unsigned W_REG sspstat with {base=2};
unsigned W_REG sspbuf with {base=2};
unsigned W_REG sspaddh with {base=2};
unsigned W_REG sspaddl with {base=2};
unsigned W_REG mask with {base=2};
unsigned W_REG inter with {base=2};

/*****************************************************************************
*                                                                            *
*        CONJUNTO DE DEFINICIONES DE VARIABLES EXTERNAS DEL TESTBENCH        *
*                                                                            *
*****************************************************************************/

unsigned W_BIT r_w;                /*Línea de lectura-escritura del TESTBENCH*/
unsigned W_REG scaler_tmr0;
unsigned W_CORTO scaler_tmr1;
unsigned W_REG prescal_tmr2;
unsigned W_CORTO postscal_tmr2;
unsigned W_REG tmr2;
unsigned W_BIT nuevo_dato;
unsigned W_REG sspsr with {base=2};

/*****************************************************************************
*                                                                            *
*        CONJUNTO DE DEFINICIONES DE CANALES EXTERNOS DEL TESTBENCH          *
*                                                                            *
*****************************************************************************/

chan unsigned W_BIT bus_int[W_REG];    /*Canal para las interrupciones*/
extern chan unsigned W_BIT wdt_test;/*Canal de aviso de TIME OUT del WDT*/

#ifndef Real
    chan unsigned W_REG busin;        /*Canal de entrada de datos del testbench*/
    chan unsigned W_REG busout;        /*Canal de salida de datos del testbench*/
    chan unsigned W_DIR addr;        /*Canal para direccionamiento del testbench*/
#else
    unsigned W_REG Out;
    interface port_in (unsigned W_REG In) busin() ;
    interface port_out() busout (unsigned W_REG Output=Out) ;
    interface port_in (unsigned W_DIR In) addr();
    interface port_in (unsigned W_BIT In) R_W();
#endif

/*****************************************************************************
*                                                                            *
*                              PROTOTIPO DE FUNCIONES                        *
*                                                                            *
*****************************************************************************/


#ifdef Simul
CONFIGURACION();
#endif
```

```
#ifdef Placa
void CONFIG_PLACA();
#endif
#ifdef Rst
void RESET();
#endif
#ifdef WATCHDOG
void WATCHDOG();
#endif
#ifdef Timer0
void TIMER0();
#endif
#ifdef Timer1
void TIMER1();
#endif
#ifdef Timer2
void TIMER2();
#endif
#ifdef Cpp
void CCP();
#endif
#ifdef Ssp
void SSP();
#endif

#ifdef Rst
/*****************************************************************************
*                                                                            *
*                            RESET GENERAL DEL SISTEMA                       *
*                                                                            *
*****************************************************************************/
void RESET()
{
    /*Valores tomados del PIC*/
    par
    {
        configh=0x00;
        configl=0x00;
        status=0x18;
        option=0xff;
        t1con=0x00;
        t2con=0x00;
        pr2=0xff;
        ccp1con=0x00;
        ccpr1l=0x00;
        trisc=0xff;
        sspcon=0x00;
        sspstat=0x00;
        sspaddh=0xf0;
        sspaddl=0x00;
        mask=0x00;
        tmr2=0x00;
        inter=0x00;
    }
}
#endif

/*****************************************************************************
*                                                                            *
*                              ESCENARIO HARDWARE                            *
*                                                                            *
*****************************************************************************/
set clock = external "P92";

/*****************************************************************************
*                                                                            *
*                              FUNCIÓN PRINCIPAL                             *
*                                                                            *
*****************************************************************************/
void main(void)
{
/***************     DEFINICIONES    ******************/
```

```
       unsigned W_DIR dir;
       signal unsigned W_REG interrup;
       signal unsigned W_BIT cause[W_REG];
       #ifdef Rst
       signal unsigned W_BIT Reset;
       #endif
       #ifdef Watchdog
       signal unsigned W_BIT Timeout;
       #endif
       signal unsigned W_REG cpy;
       signal unsigned W_BIT Rw;
       signal unsigned W_DIR Addr;

#ifndef Real
       #ifdef Rst
             chanin unsigned W_BIT reset_chan with {infile ="RESET.dat"};
       #endif
#else
       #ifdef Rst
             interface bus_in (unsigned W_BIT In) reset_chan() with {data =
{"P129"}};
       #endif
       interface bus_out () INTERRUP (unsigned W_BIT OutPort=
                          ((inter&mask)==TRUE?TRUE:FALSE)) with {data = {"P128"}};
#endif

#ifdef Simul
       chanout unsigned W_BIT INTERRUP with {outfile="INTERRUP.dat"};
#endif

#ifdef Placa
        interface bus_out () INTERRUP (unsigned W_BIT OutPort=
                          ((inter&mask)==TRUE?TRUE:FALSE)) with {data = {"P128"}};
#endif

    par
    {
    /***************    PROCESO #A: TESTBENCH    *************/
        while (TRUE)
        {
        #ifdef Real /*CASO DE REALIZACIÓN CON PINES*/
            par
            {
                r_w=R_W.In;
                dir=addr.In;
                if (r_w==FALSE)
                {
                    switch(dir)
                        {
                        case CONFIGH:
                            configh=busin.In;
                            break;
                        case CONFIGL:
                            configl=busin.In;
                            break;
                        case STATUS:
                            status=busin.In;
                            break;
                        case OPTION:
                            par
                            {
                                cpy=busin.In;
                                option=cpy;
                                scaler_tmr0=decode(cpy[2:0]);
                            }
                            break;
                        case T1CON:
                            par
                            {
                                cpy=busin.In;
                                t1con=cpy;
```

```
                scaler_tmr1=decode(cpy[5:4]);
            }
            break;
        case T2CON:
            par
            {
                cpy=busin.In;
                t2con=cpy;
                postscal_tmr2=(unsigned W_CORTO)(cpy[6:3]);
                if (cpy[1]==TRUE)
                    prescal_tmr2=0x10;
                else if (cpy[0]==TRUE)
                    prescal_tmr2=0x04;
                else
                    prescal_tmr2=0x01;
            }
            break;
        case PR2:
            pr2=busin.In;
            break;
        case CCP1CON:
            ccp1con=busin.In;
            break;
        case CCPR1L:
            ccpr1l=busin.In;
            break;
        case TRISC:
            trisc=busin.In;
            break;

        case SSPCON:
            sspcon=busin.In;
            break;
        case SSPSTAT:
            sspstat=busin.In;
            break;
        case SSPBUF:
            {
                if (nuevo_dato==FALSE)
                {
                    par
                    {
                        cpy=busin.In;
                        sspbuf=cpy;
                        sspsr=cpy;
                        nuevo_dato=TRUE;
                        /*Se debe borrar la interrupcion SSP1*/
                    }
                }
                else
                    sspcon=setbit(7,sspcon);
            }
            break;
        case SSPADDH:
            sspaddh=busin.In;
            break;
        case SSPADDL:
            sspaddl=busin.In;
            break;
        case MASK:
            mask=busin.In;
            break;
        case INTER:
            inter=busin.In;
            break;
        default:
            delay;
            break;
        }
}
else                        /*CASO DE LECTURA DEL TESTBENCH */
{
```

```
                switch(dir)
                {
                    case CONFIGH:
                        Out=configh;
                        break;
                    case CONFIGL:
                        Out=configl;
                        break;
                    case STATUS:
                        Out=status;
                        break;
                    case OPTION:
                        Out=option;
                        break;
                    case T1CON:
                        Out=t1con;
                        break;
                    case T2CON:
                        Out=t2con;
                        break;
                    case PR2:
                        Out=pr2;
                        break;
                    case CCP1CON:
                        Out=ccp1con;
                        break;
                    case CCPR1L:
                        Out=ccpr1l;
                        break;
                    case TRISC:
                        Out=trisc;
                        break;
                    case SSPCON:
                        Out=sspcon;
                        break;
                    case SSPSTAT:
                        Out=sspstat;
                        break;
                    case SSPBUF:
                        par
                        {
                            if (sspcon[7]==TRUE)
                                sspcon=clrbit(7,sspcon);
                            else
                                delay;
                            Out=sspbuf;
                            nuevo_dato=FALSE;
                            sspstat=clrbit(0,sspstat);
                        }
                        break;
                    case SSPADDH:
                        Out=sspaddh;
                        break;
                    case SSPADDL:
                        Out=sspaddl;
                        break;
                    case MASK:
                        Out=mask;
                        break;
                    case INTER:
                        Out=inter;
                        break;
                    default:
                        delay;
                        break;
                } /*Fin del switch    ESCRITURA*/
            }/*Fin del else*/
        }/*Fin del par*/

    #else /*CASO SIMULACION CON CANALES*/
        prialt
        {
```

```
case addr ? dir:    /*CASO DE ESCRITURA EN EL TESTBENCH*/
{
    if (r_w==FALSE)
    {
        switch(dir)
            {
            case CONFIGH:
                busin ? configh;
                break;
            case CONFIGL:
                busin ? configl;
                break;
            case STATUS:
                busin ? status;
                break;
            case OPTION:
                par
                {
                    busin ? cpy;
                    option=cpy;
                    scaler_tmr0=decode(cpy[2:0]);
                }
                break;
            case T1CON:
                par
                {
                    busin ? cpy;
                    t1con=cpy;
                    scaler_tmr1=decode(cpy[5:4]);
                }
                break;
            case T2CON:
                par
                {
                    busin ? cpy;
                    t2con=cpy;
                    postscal_tmr2=(unsigned W_CORTO)(cpy[6:3]);
                    if (cpy[1]==TRUE)
                        prescal_tmr2=0x10;
                    else if (cpy[0]==TRUE)
                        prescal_tmr2=0x04;
                    else
                        prescal_tmr2=0x01;
                }
                break;
            case PR2:
                busin ? pr2;
                break;
            case CCP1CON:
                busin ? ccp1con;
                break;
            case CCPR1L:
                busin ? ccpr1l;
                break;
            case TRISC:
                busin ? trisc;
                break;
            case SSPCON:
                busin ? sspcon;
                break;
            case SSPSTAT:
                busin ? sspstat;
                break;
            case SSPBUF:
                {
                    if (nuevo_dato==FALSE)
                    {
                        par
                        {
                            busin ? cpy;
                            sspbuf=cpy;
                            sspsr=cpy;
```

```
                            nuevo_dato=TRUE;
                            /*Se debe borrar la interrupcion SSP1*/
                        }
                    }
                    else
                        sspcon=setbit(7,sspcon);
                }
                break;
            case SSPADDH:
                busin ? sspaddh;
                break;
            case SSPADDL:
                busin ? sspaddl;
                break;
            case MASK:
                busin ? mask;
                break;
            case INTER:
                busin ? inter;
                break;
            default:
                delay;
                break;
            }        /*Fin del switch*/
}                    /*Fin del if*/
else                          /*CASO DE LECTURA DEL TESTBENCH */
{
    switch(dir)
    {
        case CONFIGH:
            busout ! configh;
            break;
        case CONFIGL:
            busout ! configl;
            break;
        case STATUS:
            busout ! status;
            break;
        case OPTION:
            busout ! option;
            break;
        case T1CON:
            busout ! t1con;
            break;
        case T2CON:
            busout ! t2con;
            break;
        case PR2:
            busout ! pr2;
            break;
        case CCP1CON:
            busout ! ccp1con;
            break;
        case CCPR1L:
            busout ! ccpr1l;
            break;
        case TRISC:
            busout ! trisc;
            break;
        case SSPCON:
            busout ! sspcon;
            break;
        case SSPSTAT:
            busout ! sspstat;
            break;
        case SSPBUF:
            par
            {
                if (sspcon[7]==TRUE)
                    sspcon=clrbit(7,sspcon);
                else
                    delay;
```

```
                                busout ! sspbuf;
                                nuevo_dato=FALSE;
                                sspstat=clrbit(0,sspstat);
                            }
                            break;
                        case SSPADDH:
                            busout ! sspaddh;
                            break;
                        case SSPADDL:
                            busout ! sspaddl;
                            break;
                        case MASK:
                            busout ! mask;
                            break;
                        case INTER:
                            busout ! inter;
                            break;
                        default:
                            delay;
                            break;

                    }    /*Fin del switch    ESCRITURA*/
                }        /*Fin del else*/
            }
            break; /*Del case addr ? dir*/

            default:
                delay;
                break; /*Si no hay solicitud de lectura-escritura del testbench*/

        }/*Fin del prialt*/
    #endif
    }/*Fin del proceso A*/

/***************        PROCESO #B    *****************/
    seq
    {
         #ifdef Placa /*Simulación en placa*/
                CONFIG_PLACA();
         #endif
         #ifdef Simul /*Simulación software*/
   /*INICIALIZACION DE LOS REGISTROS DE CONFIGURACIÓN DE LOS PERIFÉRICOS*/
                CONFIGURACION();
         #else
                delay;
         #endif

        par
        {
        /**********    PROCESO #B0: CONFIGURACIÓN DEL TESTBENCH  ****/
        #ifdef Placa /*Simulación en placa*/
            CONFIG_PLACA();
        #endif
        /**********    PROCESO #B1: RESET SINCRONO    ***********/
        #ifdef Rst
            while (TRUE)
            {
                par
                {
                    #ifndef Real
                        reset_chan ? Reset;
                    #else
                        Reset=reset_chan.In;
                    #endif
                        if (Reset)
                            RESET();
                        else
                            delay;
                }
            }
        #endif
        /**********    PROCESO #B2: ESCUCHA DEL TIME OUT DEL WDT ****/
```

```
#ifdef Watchdog
/*Con este proceso actualizamos el valor del bit 4 del registro STATUS en
caso de producirse un "time-out" en el periférico watchdog timer garanti-
zando la no concurrencia con otras escrituras en el mismo registro*/

    while (TRUE)
    {
        par
        {
            wdt_test ? Timeout;
            /*Timeout es una señal :el próximo ciclo valdrá '0'*/
            if (Timeout==TRUE)
            {
            #ifndef Real
                escribe(STATUS,clrbit(4,status));
                /*En simulación (Simul) no puede haber conncurrencia en la
               escritura del registro status ya que los procesos quedan
                bloqueados hasta que se lee el canal que usan para
                escribir*/
            #else
                par
                {
                    Rw=R_W.In;
                    Addr=addr.In;
                    if (Rw==FALSE && Addr==STATUS)
                    /*Concurrencia de escrituras en status*/
                    {
                        delay;
                        status=clrbit(4,status);
                    }
                    else
                        status=clrbit(4,status);
                }
            #endif
            }
            else
                delay;
        }
    }
#endif

/**********    PROCESO #B3: RUTINA DE INTERRUPCION    *******/

/*Este proceso está diseñado para evitar que se pierdan interrupciones en
caso de simultaneidad de dos o más de ellas.
Si usamos sólamente "inter=setbit(#interrupcion, inter);" ubicado en cada
uno de los procesos que generan interrupción podría suceder la concurren-
cia de dos o más interrupciones en un mismo ciclo de reloj, perdiéndose
todas menos una. Con el sistema propuesto (array de canales escuchados en
paralelo y actualización de los correspondientes bits del registro inter en
cada ciclo de reloj) conseguimos paralelismo en la activación de los bits
del registro inter. De esta manera evitamos la pérdida de interrupciones.
El bit de inter activado por cada interrupcion debe ser reseteado por
software*/


    while(TRUE)
    {
    /***        PROCESOS DE ESCUCHA DE CANALES DE INTERRUPCION    ***/
        par
        {
            bus_int[INT_TMR0] ? cause[INT_TMR0];
            bus_int[INT_TMR1] ? cause[INT_TMR1];
            bus_int[INT_TMR2] ? cause[INT_TMR2];
            bus_int[INT_CCP1] ? cause[INT_CCP1];
            bus_int[INT_CCP2] ? cause[INT_CCP2];
            bus_int[INT_SSP1] ? cause[INT_SSP1];
            bus_int[INT_SSP2] ? cause[INT_SSP2];
            bus_int[INT_SSP3] ? cause[INT_SSP3];
        }
    }
```

```
                while(TRUE)
                {
                    par
                    {
                        interrup =    cause[7]@cause[6]@cause[5]@cause[4]@
                                    cause[3]@cause[2]@cause[1]@cause[0];
                        inter = inter | interrup;

                    /* PROCESO DE ACTIVACIÓN DE LA LÍNEA DE INTERRUPCIÓN EXTERNA */

                    #ifndef Real
                        if (inter&mask)
                            INTERRUP ! TRUE;
                        else
                            INTERRUP ! FALSE;
                    #endif
                    }
                }

            /************    PROCESO #B4: WATCHDOG TIMER    *********/

            #ifdef Watchdog
                WATCHDOG();
            #endif

            /************    PROCESO #B5: TIMER0    *************/

            #ifdef Timer0
                TIMER0();
            #endif

            /************    PROCESO #B6: TIMER1    *************/

            #ifdef Timer1
                TIMER1();
            #endif

            /************    PROCESO #B7: TIMER2    *************/

            #ifdef Timer2
                TIMER2();
            #endif

            /************    PROCESO #B8: CAPTURE/COMPARE/PWM    *****/

            #ifdef Ccp
                CCP();
            #endif

            /***********    PROCESO #B9: SYNCHRONOUS SERIAL PORT    *****/

            #ifdef Ssp
                SSP();
            #endif

            }/*Fin del par de procesos B0 a B9*/
        }/*Fin del proceso B*/
    }/*Fin del par de procesos A y B*/
}/*Fin del main()*/
```

B.6. Archivo fuente: Inicio.c

```
/*****************************************************************************
*                                                                            *
*                                CONFIGURACION.c                             *
*                                                                            *
*****************************************************************************/
#include "top.h"
#include "milib.h"
#include "cabecera.h"


#ifdef Placa
interface port_in (unsigned 4 In) interrup() with {data={"P4","P5","P6","P7"}};
interface port_in (unsigned 1 In) start() with {data= {"P14"}};
unsigned 4 condicion;
unsigned 1 ini;
#endif

#ifdef Simul
void CONFIGURACION()
{
    escribe (CONFIGH,0xff);
    escribe (CONFIGL,0xff);
    escribe (STATUS,0x18);
    escribe (OPTION,0x2a);
    escribe (T1CON,0x13);
    escribe (T2CON,0x1d);
    escribe (PR2,0xff);
    escribe (CCP1CON,0x3d);
    escribe (CCPR1L,0x00);
    escribe (TRISC,0x00);
    escribe (SSPCON,0x06);
    escribe (SSPSTAT,0x00);
    escribe (SSPBUF,0xff);
    escribe (SSPADDH,0xf0);
    escribe (SSPADDL,0x00);
    escribe (MASK,0xf0);
}

#endif
#ifdef Placa

void CONFIG_PLACA()
{
    while(1)
    {
        par
        {
            ini=start.In;
            condicion=interrup.In;
            if (ini==TRUE)
            {
                switch (condicion)
                {


                    case 1:    /*WATCHDOG*/
               /*Prescaler asignado a WDT y cuenta 128 cuentas hasta desbordamiento*/
                    {
                        escribe (CONFIGH,0xff);
                        escribe (CONFIGL,0xff);
                        escribe (STATUS,0x18);
                        escribe (OPTION,0x0f);
                        escribe (T1CON,0x00);
                        escribe (T2CON,0x00);
                        escribe (CCP1CON,0x00);
                        escribe (SSPCON,0x00);
```

```
            escribe (MASK,0x00);
        }
        break;
        case 2: /*Timer0*/
        {
/*Prescaler asignado a TMR0 y cuenta 256 cuentas hasta desbordamiento*/
            escribe (CONFIGH,0xff);
            escribe (CONFIGL,0xfb);
            escribe (STATUS,0x18);
            escribe (OPTION,0x2f);
            escribe (T1CON,0x00);
            escribe (T2CON,0x00);
            escribe (CCP1CON,0x00);
            escribe (SSPCON,0x00);
            escribe (TRISC,0x80);
            escribe (MASK,0x01);
        }
        break;
        case 3: /*Timer1*/
        {
        /*Entrada por pin T1CKP*/
            escribe (CONFIGH,0xff);
            escribe (CONFIGL,0xfb);
            escribe (T1CON,0x0b);
            escribe (T2CON,0x00);
            escribe (CCP1CON,0x00);
            escribe (SSPCON,0x00);
            escribe (TRISC,0x03);
            escribe (MASK,0x02);
        }
        break;
        case 4:/*Timer2*/
        {
        /*Prescaler y postscaler (1:1) Cuenta hasta 5*/
            escribe (CONFIGH,0xff);
            escribe (CONFIGL,0xfb);
            escribe (T1CON,0x00);
            escribe (T2CON,0x04);
            escribe (PR2,0x05);
            escribe (CCP1CON,0x00);
            escribe (SSPCON,0x00);
            escribe (TRISC,0x00);
            escribe (MASK,0x04);
        }
        break;
        case 5: /*CCP Capture*/
        {
        /*Prescaler(1:1) Rising edge*/
            escribe (CONFIGH,0xff);
            escribe (CONFIGL,0xfb);
            escribe (T1CON,0x01);
            escribe (T2CON,0x00);
            escribe (CCP1CON,0x05);
            escribe (CCPR1L,0x00);
            escribe (SSPCON,0x00);
            escribe (TRISC,0x03);
            escribe (MASK,0x08);
        }
        break;
        case 6: /*CCP Compare*/
        {
            escribe (CONFIGH,0xff);
            escribe (CONFIGL,0xfb);
            escribe (T1CON,0x13);
            escribe (T2CON,0x00);
            escribe (CCP1CON,0x0b);
            escribe (CCPR1L,0x00);
            escribe (SSPCON,0x00);
            escribe (TRISC,0x03);
            escribe (MASK,0x10);
        }
        break;
```

```
                    case 7: /*CCP PWM*/
                    {
                        escribe (CONFIGH,0xff);
                        escribe (CONFIGL,0xfb);
                        escribe (T1CON,0x00);
                        escribe (T2CON,0x04);
                        escribe (PR2,0x05);
                        escribe (CCP1CON,0x3d);
                        escribe (CCPR1L,0x00);
                        escribe (SSPCON,0x00);
                        escribe (TRISC,0x00);
                        escribe (MASK,0x04);
                    }
                    break;
                    case 8: /*SSP SPI slave*/
                    {
                        escribe (CONFIGH,0xff);
                        escribe (CONFIGL,0xfb);
                        escribe (T1CON,0x00);
                        escribe (T2CON,0x1d);
                        escribe (PR2,0xff);
                        escribe (CCP1CON,0x00);
                        escribe (TRISC,0x58);
                        escribe (SSPCON,0x06);
                        escribe (SSPSTAT,0x00);
                        escribe (SSPBUF,0xff);
                        escribe (MASK,0x20);
                    }
                    break;
                    case 9:/*SSP I2C slave*/
                    {
                        escribe (CONFIGH,0xff);
                        escribe (CONFIGL,0xfb);
                        escribe (T1CON,0x00);
                        escribe (T2CON,0x1d);
                        escribe (PR2,0xff);
                        escribe (CCP1CON,0x00);
                        escribe (TRISC,0x18);
                        escribe (SSPCON,0x06);
                        escribe (SSPSTAT,0x00);
                        escribe (SSPBUF,0xff);
                        escribe (SSPADDH,0xf0);
                        escribe (SSPADDL,0x00);
                        escribe (MASK,0xf0);
                    }
                    break;

                    default:
                        delay;
                }
                while (ini==TRUE)
                     delay;
            }
            else
                delay;
        }
    }
}
#endif
```

B.7. Archivo fuente: Watchdog.c

```
/*****************************************************************************
*                                                                            *
*                                   Watchdog.c                               *
*                                                                            *
*****************************************************************************/

#include <stdlib.h>
#include "milib.h"
#include "cabecera.h"
#include "top.h"

/***************  DEFINICIONES EXTERNAS    *****************/

unsigned W_REG wdt;
unsigned W_REG cuenta_wdt;
chan unsigned W_BIT wdt_test;            /*Canal de aviso de TIME OUT del WDT*/

/****************    WATCHDOG TIMER    *********************/
void WATCHDOG()
{
    while (TRUE)
    {    /*Este bucle hace que siempre esté en estado de
        alerta incluso cuando no está activado el wdt
        salvo reset*/
        while (configl[2]==TRUE && status[4]==TRUE)
        /*Mientras que el wdt esté activado y no haya condición de Time-out*/
        {
            par
            {
                wdt++;
                if (wdt==0xff)
                {
                    if (cuenta_wdt==(option[3]?scaler_tmr0:0x01))
                    {
                        par
                        {
                            cuenta_wdt=0x00;
                            wdt_test ! TRUE;
                        }
                    }
                    else
                        cuenta_wdt++;
                }
                else
                    delay;
            }
        } /* Fin del bucle while #2*/
    delay;
    }/*Fin del bucle while #1*/
} /*Fin de función WATCHDOG()*/
```

B.8. Archivo fuente: Timer0.c

```
/*****************************************************************************
*                                                                            *
*                                   Timer0.c                                 *
*                                                                            *
*****************************************************************************/

#include <stdlib.h>
#include "milib.h"
#include "cabecera.h"
#include "top.h"

/***********************  DEFINICIONES EXTERNAS    ***************************/

unsigned W_REG tmr0;
unsigned W_REG cuenta_tmr0;
unsigned W_CORTO div_tmr0;
unsigned W_BIT turn_tmr0;

void TIMER0()
{
/***********************     DEFINICIONES    ********************************/

    unsigned W_BIT ant_tmr0;
    unsigned W_BIT intr_tmr0;
    signal unsigned W_BIT pin;

#ifdef Simul
       chanin unsigned W_BIT TOCKIwith {infile="TOCKI_IN.dat"};
#else
       interface bus_ts (unsigned W_BIT PIN) TOCKI(unsigned W_BIT TOCKI_io,
       unsigned W_BIT TOCKI_ena=trisc[7]) with {data = {"P115"}};
/* Originalmente este bit no está controlado por el bit 7 de trisc pero por mejor
utilización de este registro lo he reubicado ahí*/
#endif

/***************************** PROCEDIMIENTOS INTERNOS ***************************/

    /*****************************************************************************
    * Nombre del procedimiento:      incr_tmr0()                                 *
    *                                                                            *
    * Descripción:    -    Incrementa tmr0 en función del valor de su escaler (en   *
    *                    caso de ser asignado al timer 0).                       *
    *                 -    Genera la interrupción asociada en caso de desborda-    *
    *                     miento del timer 0.                                    *
    *                                                                            *
    *****************************************************************************/
    macro proc incr_tmr0()
    {
        if (turn_tmr0==TRUE)
        {
            par
            {
                turn_tmr0=FALSE;
                if (cuenta_tmr0==(option[3]?0x01:scaler_tmr0))
                {
                    par
                    {
                        cuenta_tmr0=0x00;
                        tmr0++;
                        if (tmr0==0xff)
                            bus_int[INT_TMR0] ! TRUE;
                        else
                            delay;
                    }
                }
                else
```

```
                    cuenta_tmr0++;
            }
        }
        else
            turn_tmr0=TRUE;
    }

    /**********************    TIMER 0    ***************************/

    while (TRUE)
    {
        if(option[5]) /*MODO COUNTER*/
        {
            par
            {
                ant_tmr0=intr_tmr0;
           #ifdef Simul
               TOCKI ? pin;
           #else
              pin = TOCKI.PIN;
           #endif
              intr_tmr0=pin^option[4];/*Se selecciona flanco*/

                if(intr_tmr0==TRUE && ant_tmr0==FALSE)
                 /*CONDICION EN MODO COUNTER: si flanco de subida en TOCKI*/
                    incr_tmr0();
                else
                    delay;
            }
        }
        else    /*MODO TIMER*/
        {
            if (div_tmr0==0x3)
            {
                par
                {
                    div_tmr0=0x0;
                    incr_tmr0();
                }
            }
            else
                div_tmr0++;
        }
    } /*Fin del while(TRUE)*/
}/*Fin de función TMR0()*/
```

B.9. Archivo fuente: Timer1.c

```
/*****************************************************************************
*                                                                            *
*                                    Timer1.c                                *
*                                                                            *
*****************************************************************************/
#include <stdlib.h>
#include "milib.h"
#include "cabecera.h"
#include "top.h"

/*****************************************************************************
*                                                                            *
*                                  INCLUSIONES                               *
*                                                                            *
*****************************************************************************/

/*************************  DEFINICIONES EXTERNAS    *************************/

unsigned W_REG tmr1h;            /*Registro alto del timer1*/
unsigned W_REG tmr1l;            /*Registro bajo del timer1*/
unsigned W_CORTO cuenta_tmr1;
unsigned W_CORTO div_tmr1;

#ifdef Ccp
extern chan unsigned W_BIT ccp_tmr1;        /*Línea de trigger entre ccp y tmr1*/
#endif

void TIMER1()
{
/*************************     DEFINICIONES    *********************************/

    unsigned W_BIT ant_tmr1;
    unsigned W_BIT intr_tmr1;
    signal unsigned W_BIT trigger_ccp;
    signal unsigned W_BIT acum;

#ifdef Simul
       chanin unsigned W_BIT RC with {infile="RC.dat"};
#else
       interface bus_ts (unsigned W_BIT Pin) RC0(unsigned W_BIT RC0_io, unsigned W_BIT
RC0_ena=trisc[0]) with {data={"P116"}};
       interface bus_ts (unsigned W_BIT Pin) RC1(unsigned W_BIT RC1_io, unsigned W_BIT
RC0_ena=trisc[1]) with {data={"P114"}};
#endif

/***************************** PROCEDIMIENTOS INTERNOS ***************************/

    /*****************************************************************************
    * Nombre del procedimiento:        incr_tmr1()                               *
    *                                                                            *
    * Descripción:       -    Incrementa tmr1 en función del valor de su escaler.   *
    *                    -    Genera la interrupción asociada en caso de desborda-  *
    *                        miento del timer 1.                                 *
    *                                                                            *
    *****************************************************************************/
    macro proc incr_tmr1()
    {
        if (cuenta_tmr1==scaler_tmr1)
        {
            par
            {
                cuenta_tmr1=0x0;
                tmr1l++;
                if (tmr1l==0xff)
                {
                    par
                    {
```

```
                    tmr1h++;
                    if (tmr1h==0xff)
                        bus_int[INT_TMR1] ! TRUE;
                    else
                        delay;
                }
            }
            else
                delay;
        }
    }
    else
        cuenta_tmr1++;
}

/*****************************    TIMER 1    *******************************/

while (TRUE)
{
    while(t1con[0]==TRUE)
    {
        par
        {
        #ifdef Ccp

        /*PROCESO DE LECTURA DE LINEA DE TRIGGER CCP*/
            ccp_tmr1 ? trigger_ccp;
            if (trigger_ccp)
            {
                par
                {
                    tmr1h=0x00;        /*TRIGGER DEBIDO AL CCP */
                    tmr1l=0x00;
                    cuenta_tmr1=0x0;
                    div_tmr1=0x0;
                }
            }
            else
                delay;
        #endif

            if (t1con[1]==FALSE)/* MODO TIMER*/
            {
                if (div_tmr1==0x3)
                {
                    par
                    {
                        div_tmr1=0x0;
                        incr_tmr1();
                    }
                }
                else
                    div_tmr1++;

            }/* FIN MODO TIMER*/

            else /*MODO COUNTER*/
            {
                par
                {
                #ifdef Simul
                  RC ? acum;
                  intr_tmr1=acum^t1con[3];
              #else
                  if (t1con[3]==TRUE)
                       intr_tmr1=!(RC1.Pin);
                  else
                       intr_tmr1=RC0.Pin;
              #endif
                  ant_tmr1=intr_tmr1;
                  if(intr_tmr1==TRUE && ant_tmr1==FALSE)
                       incr_tmr1();
```

```
                    else
                         delay;
                      ant_tmr1=intr_tmr1;
                      if(intr_tmr1==TRUE && ant_tmr1==FALSE)
                          incr_tmr1();
                      else
                          delay;
                   }
               }/* Fin modo COUNTER */
               delay;
           }/*fin del par*/
        }/*Fin while(t1con[0]==TRUE)*/
        delay;
    } /*Fin de while (TRUE)*/
}/*Fin de funcion TMR1*/
```

B.10. Archivo fuente: Timer2.c

```
/*****************************************************************************
*                                                                            *
*                                    Timer2.c                                *
*                                                                            *
*****************************************************************************/
#include <stdlib.h>
#include "milib.h"
#include "cabecera.h"
#include "top.h"

/*************************  DEFINICIONES EXTERNAS    *************************/

unsigned W_REG cuenta1_tmr2;
unsigned W_CORTO cuenta2_tmr2;
unsigned W_CORTO div_tmr2;

/*********************************    TIMER 2    *********************************/

void TIMER2()
{
    while (TRUE)
    {
        while (t2con[2]==TRUE) /*Si TMR2ON está en "on" */
        {
            if (div_tmr2==0x3)
            {
                par
                {
                    div_tmr2=0x0;
                    if (cuenta1_tmr2==prescal_tmr2)
                    {
                        par
                        {
                            cuenta1_tmr2=0x00;
                            if (tmr2==pr2)
                            {
                                par
                                {
                                    tmr2=0x00;
                                    if (cuenta2_tmr2==postscal_tmr2)
                                    {
                                        par
                                        {
                                            cuenta2_tmr2=0x0;
                                            bus_int[INT_TMR2] ! TRUE;
                                        }
                                    }
                                    else
                                        cuenta2_tmr2++;
                                }
                            }
                            else
                                tmr2++;
                        }
                    }
                    else
                        cuenta1_tmr2++;
                }
            }
            else
                div_tmr2++;
        }/* Fin del while (t2con[2]==TRUE)*/
        delay;
    }/*Fin de while(TRUE)*/
} /* Fin de la función Timer2*/
```

B.11. Archivo fuente: Ccp.c

```
/*****************************************************************************
*                                                                            *
*                                    Ccp.c                                   *
*                                                                            *
*****************************************************************************/

#include <stdlib.h>
#include "milib.h"
#include "cabecera.h"
#include "top.h"

/*****************************  DEFINICIONES EXTERNAS    *************************/

unsigned W_REG ccpr1h;
unsigned W_CORTO cuenta_ccp;
chan unsigned W_BIT ccp_tmr1;                /*Línea de trigger entre ccp y tmr1*/

void CCP()
{
    unsigned W_REG ccpr1l_cpy;
    unsigned W_CORTO prescal_ccp;
    unsigned W_BIT ant_ccp;
    unsigned W_BIT intr_ccp;

#ifdef Simul
       chanin unsigned W_BIT CCP1_In with {infile="CCP1_IN.dat"};
       chanout unsigned W_BIT CCP1_Out with {outfile="CCP1_OUT.dat"};
#else
       unsigned W_BIT Output;
       interface bus_ts(unsigned W_BIT Pin) CCP1
       (unsigned W_BIT Out=Output, unsigned W_BIT ena=!trisc[2]) with {data={"P117"}};
#endif


/***************************** PROCEDIMIENTOS INTERNOS *************************/

    /*************************************************************************
    * Nombre del procedimiento:         lee_CCP1()                           *
    *                                                                        *
    * Descripción:    - Muestrea en cada ciclo de reloj la entrada por el pin CCP1   *
    *                 - Conserva el valor de la anterior muestra.            *
    *                 - Permite actuar con flanco de subida o de bajada.     *
    *                                                                        *
    *************************************************************************/
    macro proc lee_CCP1()
    {
       signal unsigned W_BIT capt;
       par
       {
              ant_ccp=intr_ccp;
              #ifdef Simul
                     CCP1_In ? capt;
              #else
                     capt=CCP1.Pin;
              #endif
                     intr_ccp=((ccp1con[1]==TRUE || ccp1con[0]==TRUE)?capt:!capt);

       /*Se selecciona flanco*/
       }
    }

    /*************************************************************************
    * Nombre del procedimiento:         incr_CCP1()                          *
    *                                                                        *
    * Descripción:     -Incrementa el valor de los registros ccp1rh y ccp1rl        *
    *                  coordinadamente.                                      *
    *                                                                        *
```

```
*****************************************************************************/

macro proc incr_CCP1()
{
    par
    {
        ccpr1l_cpy++;
        if (ccpr1l_cpy==0xff)
            ccpr1h++;
        else
            delay;
    }
}

while (TRUE)
{
    if (ccp1con[3]==TRUE || ccp1con[2]==TRUE)            /* CCP ON */
    {
        if (ccp1con[3]==FALSE && ccp1con[2]==TRUE)        /* MODO CAPTURE */
        {
            par
            {
                lee_CCP1();
                if(intr_ccp==TRUE && ant_ccp==FALSE)
                {
                    if (cuenta_ccp==prescal_ccp)
                    {
                        ccpr1h=tmr1h;
                        ccpr1l_cpy=tmr1l;
                        cuenta_ccp=0x0;
                        bus_int[INT_CCP1] ! TRUE;
                    }
                    else
                        cuenta_ccp++;
                }
                else
                    delay;
            }

        }
        else if (ccp1con[3]==TRUE && ccp1con[2]==FALSE)    /* MODO COMPARE */
        {
            par
            {
                incr_CCP1();
                if (ccpr1h==tmr1h && ccpr1l==tmr1l)
                {
                    if (ccp1con[1]==TRUE && ccp1con[0]==TRUE)
                    {
                        par
                        {
                            ccp_tmr1 ! TRUE;        /*TRIGGER PARA EL TMR1*/
                            bus_int[INT_CCP2] ! TRUE;
                        }
                    }
                    else if (ccp1con[1]==TRUE && ccp1con[0]==FALSE)
                        bus_int[INT_CCP2] ! TRUE;

                    else if (ccp1con[1]==FALSE && ccp1con[0]==TRUE)
                        #ifndef Real
                            CCP1_Out ! FALSE;
                        #else
                            Output=FALSE;
                        #endif
                    else
                        #ifndef Real
                            CCP1_Out ! TRUE;
                        #else
                            Output=TRUE;
                        #endif
                }
                else
```

```
                        delay;
                }
            }
            else                                    /* MODO PWM */
            {
                par
                {
                    ccpr1l_cpy=ccpr1l; /*Lectura del registro ccpr1l del Testbench*/
                    if (tmr2==0x00)
                        ccpr1h=ccpr1l_cpy;
                    else
                        delay;

                    if(tmr2@00<ccpr1h@ccp1con[5:4])
                    #ifdef Simul
                        CCP1_Out ! TRUE;
                    #else
                        Output=TRUE;
                    #endif
                    else
                    #ifdef Simul
                        CCP1_Out ! FALSE;
                    #else
                        Output=FALSE;
                    #endif

                }
            }
            else
                delay;
    }
}/* FIN FUNCION CCP*/
```

B.12. Archivo fuente: Ssp.c

```
/*****************************************************************************
*                                                                            *
*                                  Ssp.c                                     *
*                                                                            *
*****************************************************************************/

#include <stdlib.h>
#include "milib.h"
#include "cabecera.h"
#include "top.h"


void SSP()
{

/**************************     DEFINICIONES    *********************************/

    unsigned W_CORTO modo_ssp;        /*Modo de funcionamiento del SSP*/
    unsigned W_CORTO cuenta_sr;       /*Contador de bits rx/tx en sspssr*/
    unsigned W_REG semi_per_spi;      /*Contador  para generar reloj master SPI*/
    unsigned W_REG per_i2c;           /*Contador para generar reloj master I2C*/
    unsigned W_REG div;               /*Divisor de la frecuencia de reloj interno*/
    unsigned W_REG permanente;        /*Contador permanente*/
    unsigned W_BIT Out_spi;           /*Salida por el pin SDO en modo SPI*/
    unsigned W_BIT Out_i2c;           /*Salida por el pin IO en modo I2C*/
    unsigned W_BIT sdi_ant;           /*Valor anterior de la entrada por SDI*/
    unsigned W_BIT sdi_act;           /*Valor actual de la entrada por SDI*/
    unsigned W_BIT clk;               /*Valor actual de reloj*/
    unsigned W_BIT clk_ant;           /*Valor anterior de reloj*/
    unsigned W_BIT master_ena;        /*Habilitador del modo master*/
    unsigned W_BIT START7;            /*Flag de estado comienzo (dir 7 bits) en I2C*/
    unsigned W_BIT START10;           /*Flag de estado comienzo (dir 10 bits) en I2C*/
    unsigned W_BIT RCV;               /*Flag de estado recepcion en I2C*/
    unsigned W_BIT XMIT;              /*Flag de estado transmision en I2C*/
    unsigned W_BIT SPI;               /*Flag de modo SPI*/
    unsigned W_BIT STOP;              /*Flag de deteccion del bit de STOP*/
    unsigned W_BIT REC;               /*Flag de byte recibido*/
    unsigned W_BIT SEND;              /*Flag de byte transmitido*/
    unsigned W_BIT WAIT;              /*Flag de estado de espera*/
    unsigned W_BIT rw;                /*Variable de lectura=1 o escritura=0*/
    unsigned W_BIT MASTER;            /*Flag de modo master en I2C*/
    unsigned W_BIT MASTER_START;      /*Flag de generación de bit start (Master I2C)*/
    unsigned W_BIT MASTER_RxTx;       /*Flag de estado de Rx-Tx (Master I2C)*/
    unsigned W_BIT MASTER_STOP;       /*Flag de generación de bit stop (Master I2C)*/
    signal unsigned W_BIT slave_ena;     /*Señal de habilitación de modo esclavo*/
    signal unsigned W_BIT master_clk;    /*Señal de reloj master*/
    signal unsigned W_BIT slave_clk;     /*Señal de reloj slave*/
    signal unsigned W_BIT ena_DAT;       /*OR de master_ena y slave_ena*/

#ifdef Simul  /*MODO SIMULACION*/

    unsigned W_CORTO reloj;
    unsigned W_CORTO cont;
    unsigned W_REG dato;
    chanout unsigned W_REG salida with {outfile="salida_i2c.dat"};

    chanin unsigned W_BIT SCK_IN with {infile="SCK_IN.dat"};
    /*Canal de entrada de reloj (modo slave)*/
    chanout unsigned W_BIT SCK_OUT with {outfile="SCK_OUT.dat"};
    /*Canal de salida de reloj (modo master)*/
    chanin unsigned W_BIT SDI_PIN with {infile="SDI.dat"};
    /*Canal de entrada de bits (serie)*/
    chan unsigned W_BIT MASTER_I2C with {infile="SDI.dat"};
    /*Canal serie que comunica con un dispositivo MASTER I2C*/
    chan unsigned W_BIT MASTER_CLK with {infile="SCK_IN.dat"};
    /*Canal de reloj master*/
```

```
    chanout unsigned W_BIT SDO_PIN with {outfile="SDO.dat"};
    /*Canal de saldia de bits (serie)*/
    chanin unsigned W_BIT SS_PIN with {infile="SS.dat"};
    /*Canal de habilitamiento externo de señal de salida*/
    chanout unsigned W_REG SSPBUF_OUT with {outfile="SSPBUF.dat"};
    /*Simulación de escritura en el registro sspbuf*/

#else               /*REALIZACION HARDWARE*/

    interface bus_ts (unsigned W_BIT In) SCK_I2C (unsigned W_BIT SCK1_Out=clk,
        unsigned W_BIT ena_SCK=trisc[3]) with { pull=1};
    /*Línea de reloj modo I2C*/
    interface bus_ts (unsigned W_BIT In) IO_I2C (unsigned W_BIT IO_Out=Out_i2c,
        unsigned W_BIT ena_SDI=trisc[4]) with { pull=1};
    /*Línea de datos (serie) modo I2C*/
    interface bus_ts (unsigned W_BIT In) SCK (unsigned W_BIT SCK2_Out=master_clk,
        unsigned W_BIT ena_SCK=trisc[3]) with {data={"P118"}};
    /*Línea de reloj modo SPI*/
    interface bus_ts (unsigned W_BIT In) SDI (unsigned W_BIT SDI_Out=FALSE,
        unsigned W_BIT ena_SDI=trisc[4]) with {data={"P126"}};
    /*Línea de entrada de bits (serie) modo SPI*/
    interface bus_ts (unsigned W_BIT In) SDO(unsigned W_BIT SDO_Out=Out_spi,
        unsigned W_BIT ena_SDO=(!trisc[5]&&!(slave_ena|master_ena))) with
{data={"P125"}};
    /*Línea de salida de bits (serie) modo SPI*/
    interface bus_ts (unsigned W_BIT In) SS(unsigned W_BIT SS_Out,
        unsigned W_BIT ena_SS=trisc[6]) with {data={"P127"}};
    /*Línea de habilitación de salida de bits modo SPI*/
    /* Originalmente este pin está controlado por el bit trisa[5] pero ha sido
reubicado en
    trisc [6] para aprovechar mejor este registro*/

#endif

/***************************** PROCEDIMIENTOS INTERNOS *****************************/

    /*****************************************************************************
    * Nombre del procedimiento:         RX_TX()                                  *
    *                                                                            *
    * Descripción:    -   Proceso de recepción y/o transmision de un byte tanto en  *
    *                   modo SPI como en modo I2C.                               *

    *                                                                            *
    *****************************************************************************/
    macro proc RX_TX()
    {
        if (clk==TRUE && clk_ant==FALSE)
        {
            par
            {
                ena_DAT=slave_ena|master_ena;
                if (SPI==TRUE)
                {
                    if (ena_DAT==FALSE)
                    {
                    #ifdef Simul
                        SDO_PIN ! sspsr[W_REG-1];
                    #else
                        Out_spi=sspsr[W_REG-1];
                    #endif
                    }
                    else
                        delay;
                }
                else
                {
                #ifndef Real
                    SDO_PIN ! sspsr[W_REG-1];
                #else

                    Out_i2c=sspsr[W_REG-1];
                #endif
```

```
            }
            sspsr=sspsr[(W_REG-2):0]@sdi_act;
                    /*El nuevo bit se introduce en la parte menos
                    significativa del registro sspsr*/
            per_i2c=permanente;
            permanente=0x00;
            cuenta_sr++;
        }
    }
    else
        delay;
}


/*****************************************************************************
* Nombre del procedimiento:         ACK()                                    *
*                                                                            *
* Descripción:    - Proceso de generación de un asentimiento en modo slave I2C. *
*                                                                            *
*****************************************************************************/

macro proc ACK()
{
    unsigned W_REG i;

    while (nuevo_dato==TRUE && STOP==FALSE)
    {
        par
        {
            WAIT=TRUE;
            Out_i2c=TRUE;
        }
    }
    if (STOP==FALSE)
    {
        par
        {
            WAIT=FALSE;
            for (i=0;i<per_i2c;i++) /*Generación del ACK*/
                Out_i2c=FALSE;
        }
    }
    else
        delay;
    par
    {
        REC=FALSE;
        STOP=FALSE;
    }
}


/*****************************************************************************
* Nombre del procedimiento:         RECIBE()                                 *
*                                                                            *
* Descripción:    Cuando se ha recibido un byte se realizan en paralelo:     *
*                      -  Generación de interrupción por buffer lleno.       *
*                      -  Puesta a 1 del bit BF.                             *
*                      -  Reinicio de la variable que controla el nº de bits reci-*
*                       bidos en sspsr.                                      *
*                      -  Activación de bandera REC.                         *
*                      -  Copia de sspsr en sspbuf en caso de no haber overflow.  *
*                                                                            *
*****************************************************************************/

macro proc RECIBE()
{
    if (cuenta_sr==W_REG)
    {
        par
        {
            bus_int[INT_SSP1] ! TRUE;
```

```
            sspstat=setbit(0,sspstat);
            cuenta_sr=0x0;
            REC=TRUE;
        #ifndef Real
                SSPBUF_OUT ! sspsr;
        #else
            if (nuevo_dato==FALSE)
                sspbuf=sspsr;
            else
                sspcon=setbit(6,sspcon); /*OVERFLOW*/
        #endif
            }
    }
    else
        delay;
}

#ifdef Simul

/*****************************************************************************
* Nombre del procedimiento:         Master_I2C()                             *
*                                                                            *
* Descripción:   -   Simula un dispositivo que hace las veces de MASTER en el   *
*                  protocolo I2C.                                            *
*                -   Se comunica con el TESTBENCH para leer el nuevo dato manda-*
*                   do y poder así escribir en sspbuf un nuevo dato sin macha -*
*                  car el anterior.                                          *
*                                                                            *
*****************************************************************************/

macro proc Master_I2C()
{
    par
    {
        reloj++;
        MASTER_CLK ! (reloj\\3);
        while (TRUE)
        {
            while (reloj<0b1100)
                MASTER_I2C ! TRUE;
            MASTER_I2C ! FALSE;
            MASTER_I2C ! FALSE;   /*Generación del bit de start*/

            while (cont<0b1000)
            {
                while (reloj!=0b1000)
                    delay;
                MASTER_I2C ! FALSE;
            }
            lee (SSPBUF,dato);
            salida ! dato;
            while (cont<0b1000)
            {
                while (reloj!=0b1000)
                    delay;
                MASTER_I2C ! TRUE;
            }

            lee (SSPBUF,dato);
            salida ! dato;
            while (cont<0b1000)
            {
                while (reloj!=0b1000)
                    delay;
                MASTER_I2C ! FALSE;
            }
            lee (SSPBUF,dato);
            salida ! dato;
            while (cont<0b1000)
            {
                while (reloj!=0b1000)
                    delay;
```

```
                    MASTER_I2C ! TRUE;
                }
                lee (SSPBUF,dato);
                salida ! dato;
                while (cont<0b1000)
                {
                    while (reloj!=0b1000)
                        delay;
                    MASTER_I2C ! FALSE;
                }
                lee (SSPBUF,dato);
                salida ! dato;

                while (reloj<0b1100)
                    MASTER_I2C ! FALSE;
                MASTER_I2C ! TRUE;
                MASTER_I2C ! TRUE;   /*Generación del bit de stop*/
            }
        }
    }
    #endif

    /**************************    SSP    *********************************/


    par
    {
    #ifdef Simul
        Master_I2C();
    #endif
        while (TRUE)
        {
            if (sspcon[5]==TRUE)
            {
                par
                {
                    permanente++;

                   /**********************    MODO SPI   **************************/
                    if (SPI==TRUE)    /* ATENCION: trisc[3] debe estar puesto a '1'
                                         Para habilitar salida por SDO master_ena a
                                          '0'*/
                    {

                        par
                        {
                        #ifdef Simul
                            SDI_PIN ? sdi_act;
                        #else
                            sdi_act=SDI.In;
                        #endif
                            RX_TX();
                            RECIBE();
                        }
                       /****************   SPI    MODO MAESTRO    ******************/
                        if (sspcon[2]==FALSE)
                        /* En modo maestro podemos tener 4 posibilidades para el reloj:
                                                  reposo de reloj:  nivel alto o bajo
                                                selección de flanco:  subida o bajada*/
                        {
                            if (sspcon[1]==TRUE && sspcon[0]==TRUE)
                            {
                                par
                                {
                                    clk_ant=clk;
                                    master_clk=(master_ena & sspcon[4]) |
((!master_ena) & (sspcon[4]^sspstat[6]^tmr2[1]));
                                    clk=master_clk;
                                #ifdef Simul
                                    SCK_OUT ! master_clk;
                                #endif
                                }
```

```
                            }
                            else
                            {
                                if (div==semi_per_spi)
                                {
                                    par
                                    {
                                        div=0x00;
                                        clk_ant=clk;
                                        master_clk=(master_ena & sspcon[4]) |
((!master_ena) & (sspcon[4]^sspstat[6]^(!clk)));
                                        clk=master_clk;
                                    #ifdef Simul
                                        SCK_OUT ! master_clk;
                                    #endif
                                    }
                                }
                                else
                                    div++;
                            }
                        }
                       /****************  SPI    MODO ESCLAVO    *******************/

                        else          /* ATENCION: trisc[3] debe estar puesto a '0' */

                         /* En modo esclavo podemos tener 2 posibilidades:
                                                reposo de reloj: nivel bajo
                                                selección de flanco: subida
                                   o bien:      reposo de reloj: nivel alto
                                                selección de flanco: bajada*/
                        {
                            par
                            {
                            #ifdef Simul
                                SCK_IN ? slave_clk;
                                SS_PIN ? slave_ena;
                            #else
                                slave_clk=SCK.In;
                                slave_ena=SS.In;
                            #endif
                                clk_ant=clk;
                                clk=(slave_ena & sspcon[4]) | ((!slave_ena) &
(sspcon[4]^slave_clk));
                            }
                        }
                    }


                   /************************    MODO I2C    **********************/
                    else
                    {
                        if (MASTER==TRUE)
                        {
                            par
                            {
                                if (div==(per_i2c>>1))
                                {
                                    par
                                    {
                                        div=0x00;
                                        clk_ant=clk;
                                        clk=sspcon[4] & !clk;
                                    #ifdef Simul
                                        SCK_OUT ! clk;
                                    #endif
                                    }
                                }
                                else
                                    div++;
                                if (MASTER_START==FALSE &&  MASTER_RxTx==FALSE &&
MASTER_STOP==FALSE)
                                {
```

```
                                Out_i2c=TRUE;
                                sspcon=clrbit(4,sspcon);/*Esto detiene el reloj*/
                            }
                            else if (MASTER_START==TRUE)
                            {
                                if (clk==TRUE && div==0x14)
                                {
                                    par
                                    {
                                        Out_i2c=FALSE;
                                        MASTER_START=FALSE;
                                    }
                                }
                                else
                                    delay;
                            }
                            else if (MASTER_RxTx==TRUE)
                            {
                                par
                                {
                                    RX_TX();
                                    MASTER_RxTx=FALSE;
                                }
                            }
                            else
                            {
                                if (clk==TRUE && div==0x14)
                                {
                                    par
                                    {
                                        Out_i2c=TRUE;
                                        MASTER_STOP=FALSE;
                                    }
                                }
                                else
                                    delay;
                            }
                        }
                    }

                    else
                    {
                        par
                        {
                        #ifdef Simul
                            MASTER_CLK ? clk;
                            MASTER_I2C ? sdi_act;
                        #else
                            clk=SCK_I2C.In;
                            sdi_act=IO_I2C.In;
                        #endif
                            clk_ant=clk;
                            sdi_ant=sdi_act;
                           if (sdi_act==FALSE && sdi_ant==TRUE && clk==TRUE &&
clk_ant==TRUE)    /*Condición de START*/
                            {
                                par
                                {
                                    bus_int[INT_SSP2] ! TRUE;/*Inter. bit start*/
                                    sspstat=setbit(3,sspstat);/*Activamos el bit
S*/
                                    RCV=FALSE;
                                    XMIT=FALSE;
                                }
                            }
                            else
                                delay;

                           if (START7==FALSE && START10==FALSE && RCV==FALSE &&
XMIT==FALSE)    /*Situación de reposo*/
                            {
```

```
                                    if ((modo_ssp==0b0110 || modo_ssp==0b1110)&&
sspstat[3]) /* I²C SLAVE, 7 bits*/
                                    {
                                        par
                                        {
                                            permanente=0x00;
                                            cuenta_sr=0x0;
                                            START7=TRUE;
                                            sspstat=clrbit(3,sspstat);
                                        }
                                    }
                                   else if ((modo_ssp==0b0111 || modo_ssp==0b1111) &&
sspstat[3])    /* I²C SLAVE, 10 bits*/
                                    {
                                        par
                                        {
                                            permanente=0x00;
                                            cuenta_sr=0x0;
                                            START10=TRUE;
                                            sspstat=clrbit(3,sspstat);
                                        }
                                    }
                                    else
                                        delay;
                                }
                                else if (START7==TRUE)
                                {
                                    if (cuenta_sr==W_REG)
                                    {
                                        if (sspsr\\1==sspaddl<-7)
                                        {
                                            if (sspsr<-1==TRUE) /*Lectura*/
                                            {
                                                par
                                                {
                                                    START7=FALSE;
                                                    XMIT=TRUE;
                                                    sspstat=setbit(2,sspstat);
                                                    trisc=setbit(4,trisc);
                                                    nuevo_dato=TRUE;
                                                    ACK();
                                                }
                                            }
                                            else                /*Escritura*/
                                            {
                                                par
                                                {
                                                    START7=FALSE;
                                                    RCV=TRUE;
                                                    sspstat=clrbit(2,sspstat);
                                                    trisc=setbit(4,trisc);
   /*Habilitamos la salida por pin IO_I2C*/
                                                    nuevo_dato=TRUE;
                                                    ACK();
                                                }
                                                trisc=clrbit(4,trisc);
                                            }
                                        }
                                        else    /*Dirección no coincidente*/
                                            delay;
                                    }
                                    else
                                        RX_TX();
                                }
                                else if (START10==TRUE)
                                {
                                    if (cuenta_sr==W_REG)
                                    {
                                        par
                                        {
                                            cuenta_sr=0x0;
```

```
                                        if (sspsr\\1==sspaddh\\1 &&
sspstat[1]==FALSE)
                                        {
                                            par
                                            {
                                                rw=sspsr<-1;
                                                trisc=setbit(4,trisc);
                                                ACK();
                                                sspstat=setbit(1,sspstat);
                                            }
                                        }
                                        else if (sspsr==sspaddl &&
sspstat[1]==TRUE)
                                        {

                                            if (rw==TRUE) /*Lectura*/
                                            {
                                                par
                                                {
                                                    START10=FALSE;
                                                    sspstat=clrbit(1,sspstat);
                                                    XMIT=TRUE;
                                                    sspstat=setbit(2,sspstat);
                                                    trisc=setbit(4,trisc);
                                                    ACK();
                                                }
                                            }
                                            else                /*Escritura*/
                                            {
                                                par
                                                {
                                                    START10=FALSE;
                                                    sspstat=clrbit(1,sspstat);
                                                    RCV=TRUE;
                                                    sspstat=clrbit(2,sspstat);
                                                    trisc=setbit(4,trisc);
                                                    ACK();
                                                }
                                                trisc=clrbit(4,trisc);
                                            }
                                        }
                                        else
                                            delay;
                                    }

                                }
                                else
                                    RX_TX();

                            }
                            else if (RCV==TRUE)
                            {
                                par
                                {
                                   if (clk==TRUE && clk_ant==TRUE && sdi_act==TRUE
&& sdi_ant==FALSE && WAIT==TRUE)
                                    {
                                        par
                                        {
                                            STOP=TRUE;
                                            sspstat=setbit(4,sspstat);
                                           bus_int[INT_SSP3] ! TRUE;/*Int. stop*/
                                            RCV=FALSE;
                                        }
                                    }
                                    else if (clk==TRUE && clk_ant==FALSE &&
REC==FALSE)
                                    {
                                        par
                                        {
                                            sspsr=sspsr[(W_REG-2):0]@sdi_act;
                                            per_i2c=permanente;
```

```
                                        permanente=0x00;
                                        cuenta_sr++;
                                    }
                                }
                                else
                                    delay;
                                RECIBE();
                                if (REC==TRUE)
                                {
                                    par
                                    {
                                        trisc=setbit(4,trisc);
                                        ACK();
                                    }
                                    trisc=clrbit(4,trisc);
                                }
                                else
                                    delay;
                            }
                        }
                        else if (XMIT==TRUE)
                        {
                            if (cuenta_sr==W_REG)
                            {
                                par
                                {
                                    cuenta_sr=0x0;
                                    sspstat=clrbit(0,sspstat);
                                    trisc=clrbit(4,trisc);
                                    SEND=TRUE;
                                }
                            }
                            else
                            {
                                par
                                {
                                    sspstat=setbit(0,sspstat);
                                    if (clk==TRUE && clk_ant==FALSE)
                                    {
                                        par
                                        {
                                        #ifdef Simul
                                            SDO_PIN ! sspsr[W_REG-1];
                                        #else
                                            Out_i2c=sspsr[W_REG-1];
                                        #endif
                                            cuenta_sr++;
                                        }
                                    }
                                    else
                                        delay;
                                }
                            }
                            if (SEND==TRUE)
                            {
                               if (clk==TRUE && clk_ant==TRUE && sdi_act==TRUE
&& sdi_ant==FALSE && WAIT==TRUE)
                                {
                                    par
                                    {
                                        STOP=TRUE;
                                        sspstat=setbit(4,sspstat);
                                       bus_int[INT_SSP3] ! TRUE; /*Int stop*/
                                        SEND=FALSE;
                                        XMIT=FALSE;
                                    }

                                }
                                else if (clk==TRUE && clk_ant==FALSE &&
sdi_act==FALSE)
                                {
                                    par
```

```
                                        {
                                            SEND=FALSE;
                                            trisc=setbit(0,trisc);
                                            permanente=0x00;
                                            cuenta_sr++;
                                        }
                                    }
                                    else
                                        delay;

                                }
                                else
                                    delay;
                            }
                           else        /*Aún no se ha recibido el bit de START*/
                                delay;

                        }
                    }
                }
            }
        }/* fin if(sspcon[5]==TRUE)*/


/*****************************************************************************/
        else   /*PROCESO DE RECONFIGURACION DEL REGISTRO SSPCON*/
               /*Para configurar el puerto serie es necesario tener a '0' el bit
               SSPEN (sspcon[5])*/
        {
            modo_ssp=(unsigned W_CORTO) sspcon[3:0];
            par
            {
                if (modo_ssp<0b0110)    /*SPI*/
                {
                    par
                    {
                       SPI=TRUE;
                       semi_per_spi=(sspcon[1]==FALSE?(sspcon[0]==FALSE?0x02:0x0
                       8):0x14);
                              /*Frecuencias 2MHz,500kHz y 200kHz para SPI*/
                        trisc=0xdb;
                    }
                }
                else        /*I²C*/
                {
                    par
                    {
                        sspcon=setbit(4,sspcon);
                        SPI=FALSE;
                        per_i2c=0x50;            /*Frecuencia 100kHz*/
                        if (modo_ssp==0b1011)    /*I²C MASTER*/
                        {
                            par
                            {
                                trisc=0x18;
                                MASTER=TRUE;
                            }
                        }
                        else                    /*I²C SLAVE*/
                        {
                            par
                            {
                                MASTER=FALSE;
                                START7=FALSE;
                                START10=FALSE;
                                RCV=FALSE;


                                XMIT=FALSE;
                               trisc=0x00; /*Líneas de datos y reloj a nivel
alto*/
                            }
```

```
                            }
                        }
                    }
                    sspcon=setbit(5,sspcon);
                }
            }
        delay;
        }/*Fin del par MASTER-SSP*/
    } /*fin while(TRUE)*/
}/*FIN FUNCION SSP*/
```

ÍNDICE DE ILUSTRACIONES

ÍNDICE DE ILUSTRACIONES