
# UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

## FACULTAD DE ESTUDIOS SUPERIORES CUAUTITLÁN

# IMPLEMETACIÓN DE UN MICROPROCESADOR SINTETIZABLE T80 SOBRE UN FPGA XILINX SPARTAN 3

TESIS

QUE PARA OBTENER EL TÍTULO DE:

**INGENIERO MECÁNICO ELECTRICISTA**

PRESENTA:

**PÉREZ MOROYOQUI RENÉ**

ASESOR: **M. en C. VICENTE MAGAÑA GONZÁLEZ**

CUAUTITLAN IZCALLI , EDO DE MEX. 2011

Un sincero agradecimiento a la Universidad Nacional Autónoma de México por brindarme una formación profesional y humana que me ha dejado una riqueza de experiencias personales y de conocimientos, difíciles de cuantificar.

A los profesores, que a lo largo de este camino me guiarón, desde los mejores -la mayoria- hasta algunos que debieron de haber optado por otra carrera que no fuera la enseñanza -estos afortunadamente pocos- por que en esta huerta crece de todo.

A mi padre cuyas manos me apoyaron y dieron soporte mientras estuve en este largo proceso. Su ayuda incondicional, su calidez de persona, su generosidad su sentido del humor, su inventiva y su infinita paciencia siempre fueron un referente para mí, una persona en toda la extensión de la palabra, con algunos defectos como los tenemos todos, sinceramente, gracias .

A mi madre, de quien aprendi a no darme por vencido, a seguir adelante, a tomar decisiones y apegarme a ellas hasta el final.

A mis hermanos, que si bien su apoyo fue nebuloso, sé, que sinceramente me desean éxito en mi carrera y reconozco que cuento con su total respaldo, cariño, animos y voluntad para mis proyectos en la vida.

A todos aquellos que me apoyaron, me brindaron su amistad , experiencias, conocimientos y palabras de aliento.

Pumas 106

***Señor, ya que estas desgracias son la cosecha de la caballería, dígame vuestra merced si suceden muy a menudo, o si tienen sus dias limitados en que acaecen; por que me parece a mí que a dos cosechas quedaremos inútiles para la tercera, si Dios, por su infinita misericordia, no nos socorre.***

**Sancho Panza**

## Índice General

# 1 Introducción

Los FPGAs (*Field Programmable Gate Array)* son dispositivos con un alto numero de elementos lógicos programables. Su flexibilidad en el diseño los hace ideales para desarrollos rápidos y complejos. En estos dispositivos se pueden programar funciones muy diversas , desde una compuerta hasta elementos mas complejos como memorias o microprocesadores , teniendo un alto grado de flexibilidad.

Los microprocesadores sintetizables sobre FPGAs llevan tiempo siendo utilizados en entornos específicos de desarrollo. Se tiene la innegable ventaja de la rapidez en el diseño e implementación de sistemas empotrados. Disponibles en el mercado pueden encontrarse microprocesadores sintetizables de 8, 16 ,34 y 64 bits, generalmente de paga.

Sin embargo existen alternativas libres que pueden encontrarse , y cuyo único inconveniente es la falta de soporte técnico para su implementación y puesta en funcionamiento. Ejemplo de ello es el proyecto OpenSparc [1], un microprocesador sintetizable programado en Verilog, su bus de datos es de 64 bits con 32 hilos de ejecución cuyas funcionalidades son idénticas al microprocesador UltraSparc T1 de Sun Microsystems, el código fuente puede obtenerse gratuitamente y esta bajo licencia GPLv2. Siguiendo con la misma línea, se tiene el procesador Leon que es una CPU RISC de 32 bits , desarrollado por la compañía Gaisler Research [2] , un echo destacable de este CPU es que lo utiliza la Agencia Espacial Europea (ESA) para sus programas espaciales, se encuentra disponible bajo una licencia dual LGPL/ FLOSSS. Estos microprocesadores demandan un sistema de desarrollo bastante robusto y por lo tanto oneroso, ya que para implementar un microprocesador OpenSparc / Leon es necesario contar con una tecnologia en FPGA capaz de soportarlo y sus prestaciones generalmente son sobradas para la mayoria de los proyectos.

Existen sin embargo microprocesadores mas modestos, aptos para desarrollos de sistemas que no requieran un gran poder de computo, entre estos se puede mencionar el T80 cuyas funciones son idénticas a la del clásico Z80 de Zilog, el desarrollo gl85 cuya estructura asemeja al 8085 de Intel. Estos son ideales para la puesta en marcha de sistemas embebidos y pequeños dispositivos de control, además de que acercan a cualquier alumno de electrónica al desarrollo inicial y básico, obteniendo practica y conocimiento, si es que quiere adentrarse en sistemas mas complejos.

1 Página oficial del proyecto http://www.opensparc.net/

2 Página electrónica de la compañía donde se puede encontrar información acerca del microprocesador http://www.gaisler.com/cms/

## 1.1 El problema

Los sistemas FPGA tienen una gran flexibilidad para la implementación de sistemas digitales, ya que pueden ser reprogramados, los hay en el marcado con múltiples características entre las que se pueden mencionar la velocidad, interfaces Ethernet , memoria RAM , display LCD etc. . Pueden contener sistemas digitales diseñados en forma esquemática, programados en lenguaje VHDL, Verilog, C. El punto que se aborda en el presente trabajo es la implementación de un microprocesador T80 sobre la plataforma especifica del fabricante Xilinx, este sin embargo puede ser aplicado a otros FPGAs en el mercado.

En sistemas embebidos el microprocesador Z80 continua teniendo una gran presencia, se lanzo al mercado en 1979 y desde entonces se ha estado fabricando, es un microprocesador de 8 bits con múltiples herramientas de software libre que pueden ayudar a programarlo. El inconveniente para desarrollar sistemas basados en este microprocesador es que el mismo necesita circuitos extras para funcionar, por ejemplo, memoria ROM , memoria RAM , circuitos codificadores de direcciones, circuitos de reloj, interfaces de bus , etc, además de circuitería extra para la aplicación en especifico. La ventaja de implementarlo en un FPGA es que en este puede ser programada la memoria RAM , ROM y la circuitería faltante para su funcionamiento, sin tener que pasar por el largo proceso de re-alambrado de los circuitos , adquisición de nuevos componentes, seguimiento de errores y depuración del código ( borrar-programar la memoria ROM ) constantemente.

En el mercado existen compañías que venden el IP Core ( bloque de propiedad intelectual) del Z80, es decir una funcionalidad que puede ser incluida en un FPGA, este componente que es vendido puede ser sintetizado y ejecuta las mismas instrucciones que el micro Z80 dentro del FPGA. Los costos de este bloque IP dependen de la compañía, el numero de implementaciones si es para uso comercial o no comercial , si es para uso académico y esta sujeto a un contrato que indica restricciones en la implementación Para evitar estos contratiempos se puede recurrir a un núcleo IP libre denominado T80 desarrollado por Daniel Wallner (jesus@opencores.org) en 2002 y que se encuentra disponible de forma gratuita en www.opencores.org y esta bajo licencia GPL [3] .

El problema con el núcleo T80 es que solo se encuentra el código fuente escrito en VHDL, no hay en Internet, libro, manual o tutorial alguno que indique la forma de implementarlo , configurarlo , sintetizarlo ni tampoco la forma para adaptar software como compiladores y ensambladores. Este trabajo buscara la forma de implementarlo en un sistema FPGA , modificarlo si es el caso y así mismo mostrar la forma en que se pueden usar herramientas de software libre para crear programas compatibles con este micro.

Debido a que estas funcionalidades son como componentes electrónicos, pero sin su parte física, se los suele llamar componentes virtuales.

3 GNU General Public License. Licencia Pública General de GNU, licencia creada por GNU Public License y se orienta a proteger el libre uso, distribución, y modificación de software , para protegerlo de intentos de apropiación que restrinjan esas libertades a los usuarios

## 1.2 El objetivo

Se planteó como objetivo la implementación del núcleo T80 en un sistema FPGA Xilinx Spartan 3E así como la integración de un ensamblador para generar programas que puedan ejecutarse sobre el mismo.

## 1.3 Resultados Esperados

Se espera que el sistema sea implementado de forma exitosa en un FPGA Xilinx, y que pueda ejecutar programas elaborados para el Z80.

# 2 Hardware

Diversos fabricantes ofrecen soluciones FPGA entre ellos MathStar Inc., Achronix Semiconductor , Atmel , QuickLogic, Actel, Lattice Semiconductor, Altera ,Xilinx. De estos se eligió a Xilinx por ser pionero en la implementación de esta tecnología , además cuenta con una amplia gama de productos y precios.

Se eligió la plataforma Spartan 3 por ser una familia de gama intermedia , capaz de soportar la síntesis de un microprocesador de 8 Bits. Su costo no es es excesivo, eligiendo la tarjeta de desarrollo de la empresa Digilent Starter Kit con el dispositivo XC3S200 cuyo precio gira entorno a los $ 2000.

Las características del FPGA son las siguientes:

- Dispositivo de 200 000 compuertas lógicas
- 4320 Celdas Lógicas
- 12 multiplicadores de 18 x 18 bits integrados en hardware
- 12 bloques de memoria de 18Kbit , con un total de (216Kbits)
- Cuatro manejadores de reloj digital
- Arriba de 173 pines para entrada/salida definidas por el usuario

En cuanto a las características de la tarjeta de desarrollo, se tiene:

- Memoria Flash serial de 2Mbit para la programación ROM del FPGA
- 1 Mbyte de memoria Asíncrona SRAM
  - Dos 256x16 ISSI IS61LV25616AL-10T 10 ns SRAM
  - Arquitectura de Memoria Configurable
    - Arreglo sencillo de 256K x 32 SRAM
    - Dos arreglos independientes de 256K x16 SRAM
    - CS individual para cada dispositivo
- Puerto VGA estándar
- Puerto serial RS232 .
  - Conector hembra DB9 (DCE conector)

    - RS-232 acoplador de voltaje.
- Conector de entrada PS/2 de teclado o  ratón..
- 4 display de 7 segmentos color rojo.
- 8 salidas de led individuales.
- 8 interruptores deslizables para entrada
- 4 interruptores push buttons
- 1 fuente con un cristal oscilador de 50 MHz
- 1  zócalo para un cristal auxiliar.
- 1 jumper para elegir el modo de configuración del FPGA
- 1 push button para forzar el reset del FPGA
- 1 Led indicador  de que el FPGA  se configuro correctamente.
- 3 Conectores de 40 pines para expansión
- 1 Puerto JTAG  para programación del FPGA
- 1 Eliminador de corriente para alimentación eléctrica
- 1 Led indicador para encendido.
- Reguladores en tarjeta de 3.3 V , 2.5 V , y 1.2 V


Figura 2.1[4]

4 Imagen descargada  de la página principal de Xilinx  www.xilinx.com. Corresponde a la placa de desarrollo FPGA **Xilinx Starter Kit Spartan 3**

# 3 Software

La mayoría del software utilizado en este trabajo consisten en programas con licencia GPL/GNU .Es decir que la descarga , distribución, y modificación se hacen sin violentar leyes de derecho. La única excepciona esto es el software de desarrollo Xilinx ISE, este puede obtenerse de varias formas. En primer lugar una copia de este software se suministra con la tarjeta de desarrollo y con ello una licencia para su uso. La segunda forma de obtener el software es descargándolo directamente de la página de Xilinx, previo registro, en este caso se puede optar por el paquete WebPack de uso gratuito o la paquetería completa y en este caso poder utilizarlo por un periodo de pruebas con funcionamiento de 30 días. En caso de que requiera adquirir este software directamente de la compañía pagando la licencia , el costo puede ser de aproximadamente $ 2495 dolares.

Este trabajo se elaboró tomando en cuenta la compatibilidad de plataformas y arquitecturas, todos los programas pueden trabajar en arquitecturas x86 sobre los sistemas operativos, Windows XP, Windows Vista, Windows 7, Linux Red Had Enterprise y SUSE Linux Enterprice . Los requerimientos que para el sistemas de software mas robusto que es Xilinx ISE se encuentran en torno a los 8Gb de espacio en disco y 1 MB de memoria RAM con un procesador Pentium III a 800 Mhz o equivalente como mínimo.

## 3.1 Lenguaje Ensamblador.

Debido a que el T80 duplica las características del microprocesador , se procederá como si el sistemas fuera hecho para el micro Z80.

Se elegirá un ensamblador que pueda general archivos hexadecimales utilizables por este micro,. Y este código se colocará para su ejecución en el prototipo de prueba esperando que las instrucciones en el T80 realicen las mismas funciones y comprobándolas de acuerdo a resultados esperados.

## 3.2 Programa ensamblador Pasmo.

Dentro de las múltiples herramientas disponibles libremente se eligió el ensamblador PASMO, el mismo es multiplataforma, fácil de compilar y usar. Puede generar código para muchas arquitecturas z80 y emuladores. Su autor es Julián Albo y puede ser usado y distribuido bajo licencia GPL [5].

La función de este software es traducir el lenguaje ensamblador de un programa a un archivo conteniendo los opcodes con el cual se pueda programar la memoria ROM del microprocesador.
El lenguaje ensamblador es el mismo que el usado para el microprocesador Z80. Puede encontrarse el manual de referencia del lenguaje específico de este micro en la página del fabricante www.zilog.com [6]

5 Su página en Internet es http://www.arrakis.es/~ninsesabe/pasmo/

6 En esta dirección se puede descargar el archivo en formato pdf que contiene el manual del lenguaje ensamblador http://www.zilog.com/docs/z80/um0080.pdf

### 3.2.1 Instalación de Pasmo dentro de GNU/Linux.

Se descarga desde la página del autor, en este caso se ejecuta desde la consola de comandos de Linux.

[Tet@localhost ~]$ wget http://pasmo.euler.es/bin/pasmo-0.5.3.tgz

Una vez descargado el archivo es descomprimido con el siguiente comando.

[Tet@localhost programas]$ tar xvf pasmo-0.5.3.tgz

Para compilar el código fuente tenemos los siguientes comandos .

[Tet@localhost pasmo-0.5.3]$ ./configure

[Tet@localhost pasmo-0.5.3]$ make

Para instalar los archivos binarios ya compilados tenemos que cambiar al usuario root (administrador de sistema ) y dentro del directorio en donde se ha compilado el programa ejecutamos

[Tet@localhost pasmo-0.5.3]$ make install

Una vez que el comando termina su tareas , los binarios ya se encuentran instalados.

### 3.2.2 Comprobación de la instalación del programa pasmo

Para Probar que el sistemas ya contiene el comando de ejecución, invocamos al binario

[Tet@localhost pasmo-0.5.3]$ pasmo

lo cual nos da la siguiente salida de consola

Pasmo v. 0.5.3 (C) 2004-2005 Julian Albo

Usage:

pasmo [options] source object [symbol]

Las opciones que pueden pasarse al programa al momento de llamar al ejecutable son los siguientes:

```
Opciones:
                -d          -->  Mostrar información de depuración
                                 durante el ensamblado.

                -1          -->  Mostrar información de depuración
                                 durante el ensamblado, también en
                                 el primer paso.
```

```
-v          -->  Verboso. Muestra información de
                 progreso del ensamblado.

-I          -->  Añadir directorio a la lista de
                 directorios en los que se buscarán
                 ficheros para INCLUDE e INCBIN.

--bin       -->  Generar el fichero objeto en binario
                 puro sin cabecera.

--hex       -->  Generar el fichero objeto en formato
                 Intel HEX.

--prl       -->  Generar el fichero objeto en formato
                 PRL. Adecuado para RSX de CP/M Plus.

--cmd       -->  Generar el fichero objeto en formato
                 CMD de CP/M 86.

--plus3dos  -->  Generar el fichero objeto con cabecera
                 PLUS3DOS (Spectrum disco).

--tap       -->  Generar un fichero .tap para emuladores
                 de Spectrum (imagen de cinta).

--tzx       -->  Generar un fichero .tzx para emuladores
                 de Spectrum (imagen de cinta).

--cdt       -->  Generar un fichero .cdt para emuladores
                 de Amstrad CPC (imagen de cinta).

--tapbas    -->  Igual que que la opción --tap pero
                 añadiendo un cargador Basic.

--tzxbas    -->  Igual que que la opción --tzx pero
                 añadiendo un cargador Basic.

--cdtbas    -->  Igual que que la opción --cdt pero
                 añadiendo un cargador Basic.

--amsdos    -->  Generar el fichero objeto con cabecera
                 Amsdos (Amstrad CPC disco).

--msx       -->  Generar el fichero objeto con cabecera
                 para usarse con BLOAD en MSX Basic.

--public    -->  El listado de símbolos incluirá sólo los
                 declarados PUBLIC.

--name      -->  Nombre para la cabecera en los formatos
                 que lo usan (si no se especifica se usa
                 el nombre del fichero objeto).

--err       -->  Dirige los mensajes de error a la salida
                 estándar en vez de a la salida de error
                 (excepto los errores en las opciones).
```

```
--nocase   -->  Hace que los identificadores no distingan
                mayúsculas de minúsculas.

--alocal   -->  Modo autolocal: las etiquetas que comienzan
                por un '_' son locales y su ámbito termina
                en la siguiente etiqueta no local o en la
                siguiente directiva PROC, LOCAL o MACRO.

-B
--bracket  -->  Modo sólo corchetes: los paréntesis quedan
                reservados para expresiones.

-E
--equ      -->  Predefine una etiqueta.

-8
--w8080    -->  Mostrar warning cuando se usan instrucciones
                del z80 que no existen en el 8080.

--86       -->  Generar código 8086.

-          -->  Fin de opciones, todo lo que siga se
                consideran nombres de fichero aunque
                comience por -.
```

### 3.2.3 Prueba del ensamblado

El siguiente ejemplo mostrará la forma de generar el archivo Intel Hex a partir del archivo en lenguaje ensamblador.

### 3.2.4 Descripción del programa de prueba

*Carga en el acumulador el valor hexadecimal 0xb8 , este valor se guarda en la localidad de memoria 0xfe63, después carga el valor 0xac en el acumulador , se suma con el contenido de la localidad de memoria 0xfe63 y el resultado exponerlo en el puerto 0x01*

Código fuente en ensamblador.

```
org     0x0000
ld      a,0xb8
ld      hl,0xfe63
ld      (hl),a
ld      a,0xac
add     a,(hl)
out     (0x01),a
end
```

El archivo que contiene este código se ha designado como 000.asm. A continuación desde la línea de comando ejecutamos.

[Tet@localhost pasmo-0.5.3]$ pasmo --hex 000.asm 001.hex

El archivo 000.hex resultante es un archivo Intel Hex. Para volcar el contenido de este archivo en pantalla, ejecutamos

[Tet@localhost pasmo]$ cat 001.hex

*:0B0000003EB82163FE773EAC86D301C2*
*:00000001FF*

Este es un archivo IntelHex estándar el cual puede ser programado en una memoria ROM para poder ejecutarse en un sistema Z80, en nuestro caso este contenido sera convertido a un archivo VHDL para poder ser insertado dentro del modulo de memoria ROM del esquemático en el programa Xilinx ISE.

Con esto hemos instalado y comprobado el funcionamiento del compilador pasmo.

### 3.2.5 Instalación del programa Pasmo dentro de Windows

La instalación sobre la plataforma Windows no reviste mayores complicaciones. Se entra a la página de Internet de PASMO y se descarga el ejecutable Pasmo.exe al momento de llemarlo para la ejecución se hace sobre MS-DOS, en la línea de comando en el mismo directorio en el que se ha guardado el archivo.

## 3.3 Instalación de WINE

El programa WINE ( Wine is Not a Emulator ) es un implementación de la API de WIN16 y Win32 para sistemas operativos Unix. Permite la ejecución de programas escritos originalmente para Ms-DOS , Windows 3.11 , 95 , 98, ME , NT , 200 y XP. Se instala para permitir la ejecución del programa hex2rom que convierte el contenido de un archivo IntelHex a una implementación en VHDL que puede ser sintetizada como memoria ROM dentro de el sistemas de desarrollo Xilinx ISE .

[root@localhost pasmo]# yum install wine

## 3.4 Programa hex2rom

El archivo .hex procedente del ensamblado por medio del programa PASMO o mediante el compilador SDCC es un archivo en formato Intel HEX , para poder incluirlo en la síntesis dentro del programa Xilinx ISE se necesita colocarlo en formato VHDL para este propósito se recurre al programa hex2rom escrito por Daniel Wallner, este programa creado para ms-dos al ser ejecutado convierte el archivo con formato .hex a un archivo . vhdl conteniendo la misma información de las locaciones de memoria. Este archivo es el que formara parte de la síntesis en el programa Xilinx ISE.

Para ejecutar este programa dentro de el sistema operativo GNU Linux se recurrirá al programa nativo wine. Los programas fuentes escritos en C++ pueden descargarse desde Internet[7] para poder compilarse y poder ejecutarlo en varias plataformas. La Licencia bajo la cual el software es usado distribuido y modificado es GPL.

### 3.4.1 Parámetros de Operación.

Los siguientes son los parámetros de operación de hex2rom

-b esta opción esta disponible si el archivo leído esta en formato binario.

La cadena de formato que se pasa como argumento al programa es de la forma AEDOS en donde:

A=Bits de Direcciones . En este caso el formato debe de ser de 16 bits ya que es este la longitud de el registro de direcciones.

E = Ordenación de datos Big-endian o Little-endian, l or b

Designa la forma en que se almacenan los datos de mas de un byte. Little-endian (l) o Big-endian (b) , El CPU Z80 el Little-endian , eso significa que cuando se coloca un valor en el registro de 16 bits hl por ejemplo , el byte mas significativo se almacena en el registro H y el menos significativo en el registro L.

D = Data bits

Bits de datos, en ancho de bits del bus del Z80 es 8 Bits

O = ROM t

z para colocar la memoria en alta impedancia.

a para una arreglo ROM

s para una ROM síncrona.

u para XST ucf

l para formato Leonardo UCF

7 Sitio en donde puede descargarse el código fuente http://code.google.com/p/processoronanfpga/source/browse/trunk/tools/?r=4

### 3.4.2 Ejemplo de aplicación hex2rom

Descripción del programa.

- Carga el valor 0xbc en el acumulador y después se coloca en el puerto 0x00.

Archivo ensamblador

```
            org     0x0000
inicio:
      ld            a,0xbc
      out           (0x00),a
      end
```

Contenido del archivo Intel Hex

```
:040000003EBCD3002F
:00000001FF
```

Ahora ejecutamos win2hex con ayuda del programa wine dentro de Linux. Desde la línea de comandos se introduce lo siguiente.

```
[Tet@localhost pasmo]$  wine hex2rom.exe 100.hex Bloque_Memoria  16l8 > 001.vhdl
```

Volcando el contenido del archivo 001.vhdl tenemos en contenido de dicho archivo que se agregara al proyecto mas adelante.

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity Bloque_Memoria is
      port(
            A      : in std_logic_vector(15 downto 0);
            D      : out std_logic_vector(7 downto 0)
      );
end Bloque_Memoria;
architecture rtl of Bloque_Memoria is
begin
      process (A)
      begin
            case to_integer(unsigned(A)) is
            when 000000 => D <= "00111110"; -- 0x0000
            when 000001 => D <= "10111100"; -- 0x0001
            when 000002 => D <= "11010011"; -- 0x0002
            when 000003 => D <= "00000000"; -- 0x0003
            when others => D <= "--------";
            end case;
      end process;
end;
```

## 3.5 OpenCores[8]

En este sitio se encuentran disponibles, núcleos, herramientas y los distintos lenguajes y plataformas para la implementación de microprocesadores, memorias , SoC (system on chip ), coprocesadores matemáticos , etc . La mayoría de la información y archivos dentro de este repositorio se distribuye con licencia GPL así que se pueden bajar alterar y usar siempre y cuando el código se de a conocer a la comunidad en general. De esta fuente se descarga los archivos escritos en VHDL que se usaran para la síntesis del microprocesador T80.


Figura 3.5.1[9]

El acceso y la navegación dentro del sitio en los distintos proyectos puede hacerse de forma libre, sin embargo para descargar información, como archivos fuentes, esquemáticos o datos en general, se necesita contar con un registro en el sitio. La forma para darse de alta es rápida y sencilla , solo se debe proporcionar, nombre, correo electrónico y contraseña. Una vez realizada esta operación se puede registrar al entrar a la página, mientras se navegue en ella se podrá acceder área de descargas de todos los proyectos sin restricción alguna. El proyecto que se ubica para este trabajo esta bajo el apartado **Poyects -> Processor -> T80_cpu** .

8 El sitio en Internet es http://opencores.org/

9 Captura de pantalla de la página principal http://opencores.org/

## 3.6 Xilinx ISE Design Suite 11.1

El paquete de desarrollo Xilinx ISE Design Suite 11.1 generalmente se suministra con los productos de la compañía que se hayan adquirido recientemente. La licencia es para un usuario,en caso de que requiera adquirir el programa de forma distinta, hay soluciones accesibles. El paquete WebPack es un paquete que puede descargarse desde Internet, no contiene todas las características del paquete completo, pero contiene las suficientes para este trabajo. Se puede descargar la versión completa y con una licencia de 30 días de uso gratuito, previo registro.

### 3.6.1 Instalación de SDK Xilinx ISE.

Una ves que se introduce el DVD de instalación, la pantalla de bienvenida se despliega en pantalla.

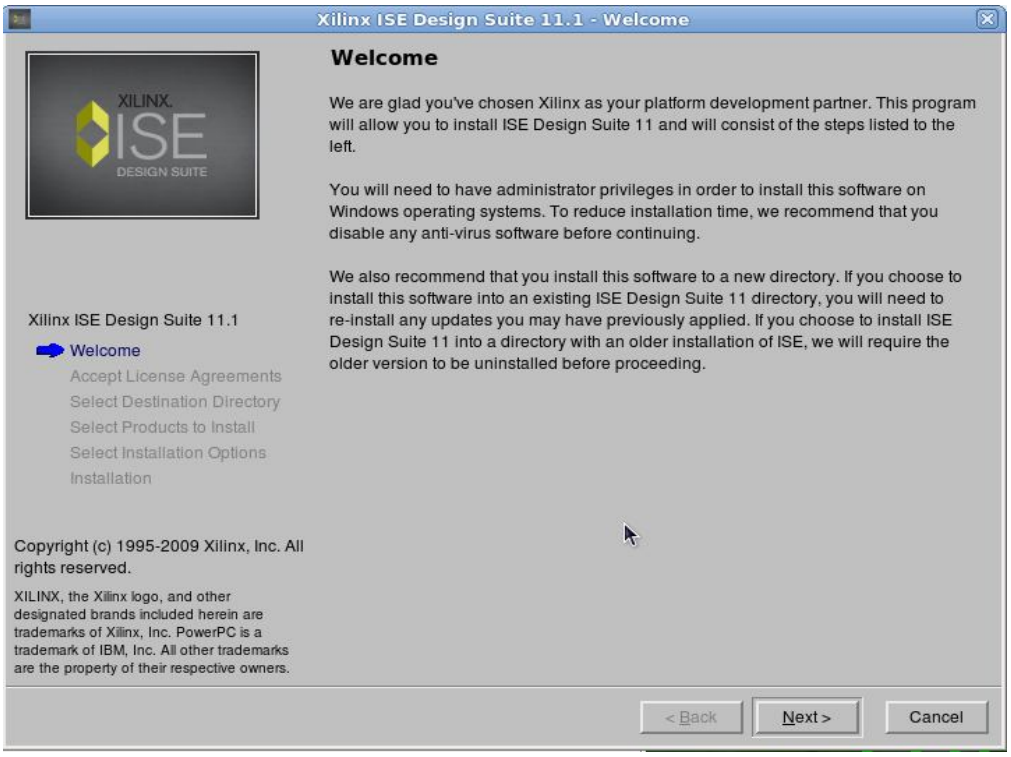



Figura 3.6.1.1[10]

10 Inicio del programa de instalación del software Xilinx ISE Design Suite 11.1

Las dos pantallas siguientes se pide la aceptación de los términos de licencia del programa.


Figura 3.6.1.2[11]

La segunda parte de la aceptación de la licencia se muestra aquí.

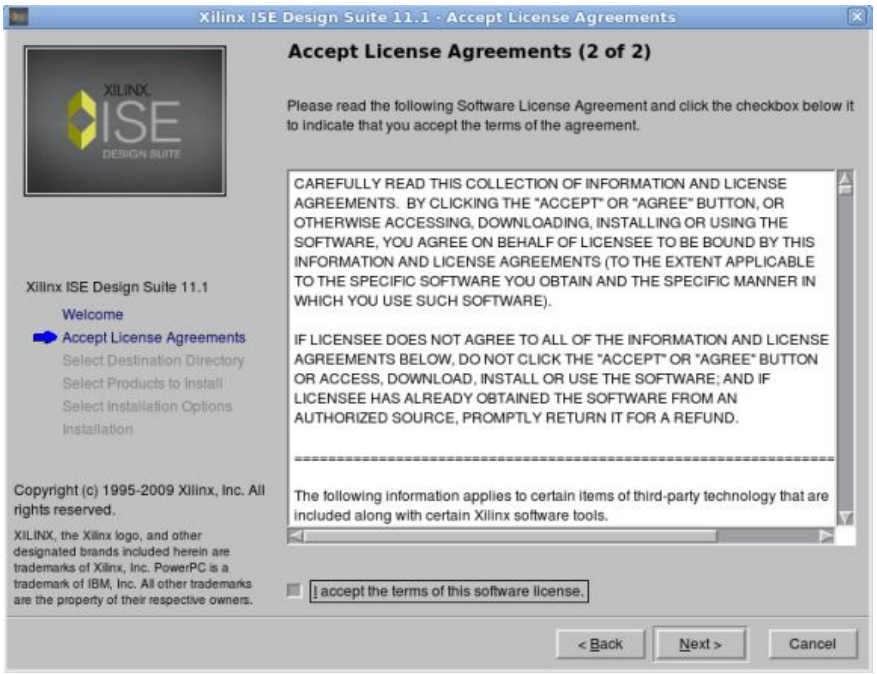



Figura 3.6.1.3[12]

11 Primera parte de los acuerdos de la licencia.
12 Segunda parte de los acuerdo de licencia.

Una vez aceptadas las políticas de uso de software la siguiente pantalla muestra la ruta en donde se instalará Xilinx ISE Design Suite 11.1, en este caso se instalará en GNU/Linux sin embargo puede también instalarse en MS-Windows en la ruta al disco duro que se crea mas conveniente.

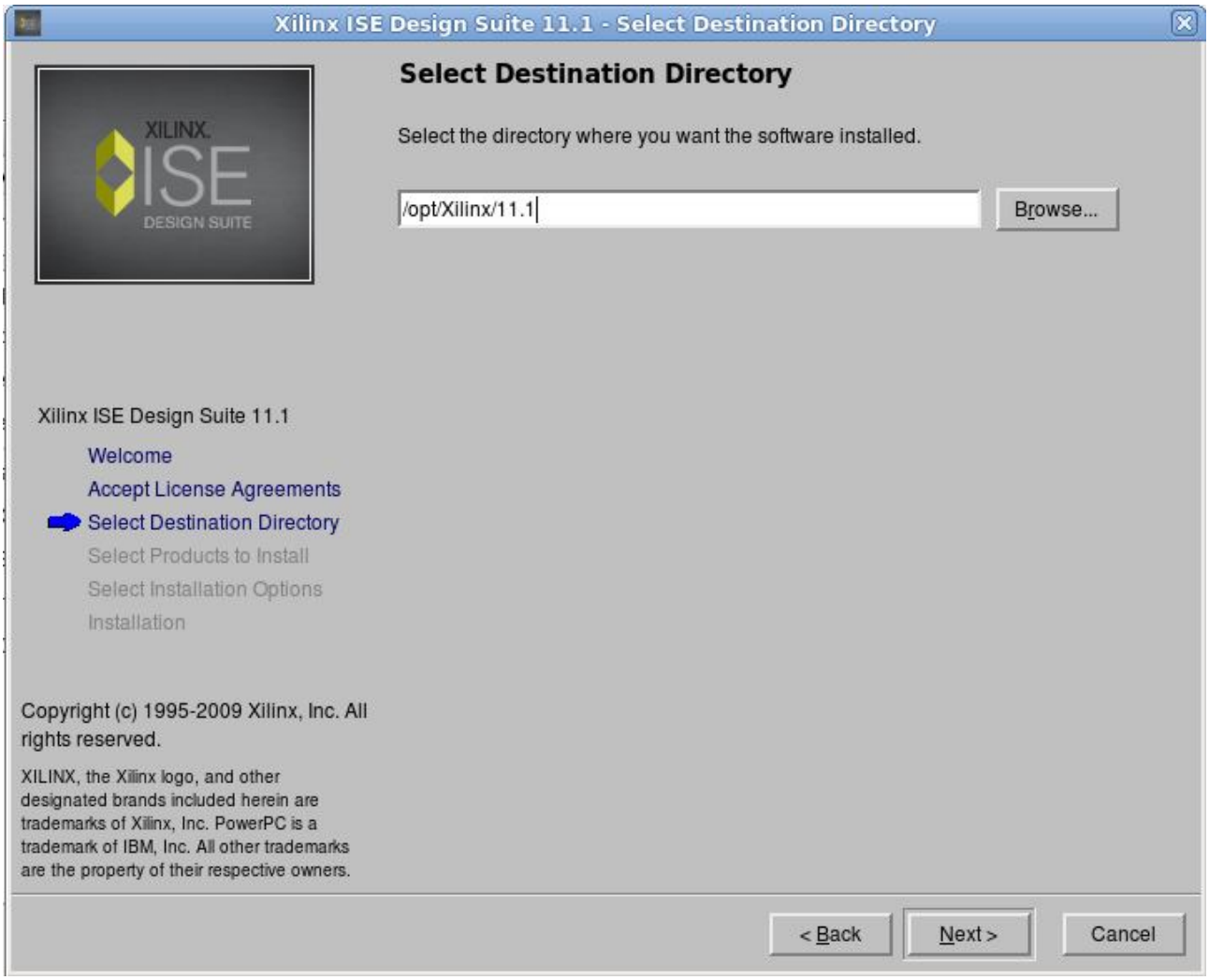



Figura 3.6.1.4[13]

El software Xilinx ISE Design Suite 11.1 puede ser instalado dentro de los sistemas operativos MS-Windows 2000, XP, Vista y 7 y en los sistemas *nix , Linux Red Had Enterprice Edition, SUSE Linux y Fedora LINUX bajo plataformas x86 y x86_64. Dependiendo del sistema operativo y de la plataforma de hardware sera necesario diversos requerimientos para la instalación y arranque de los programas. La mayoría de las computadoras que pueden adquirirse al día de hoy superan con creces los requerimientos mínimos necesarios para poder ejecutar las aplicaciones que se instalan. Para el presente trabajo se instaló el software en dos sistemas operativos distintos. Uno fue el MS- Windows Xp y el otro GNU/GPL Linux Fedora 13. Esto se llevo a cabo así para probar la generación de programas tanto en una plataforma u otra y verificar que independientemente de ello la síntesis de los programas funciona de forma correcta.

13 Cuadro de diálogo en el que se pide introducir una ruta para la instalación de toda la paquetería de software, en este caso en Linux, se instala dentro del directorio /opt/Xilinx/11.1.

El siguiente paso consiste en seleccionar las herramientas que se instalarán, en este caso se selecciona **ISE Design Suite Product.**

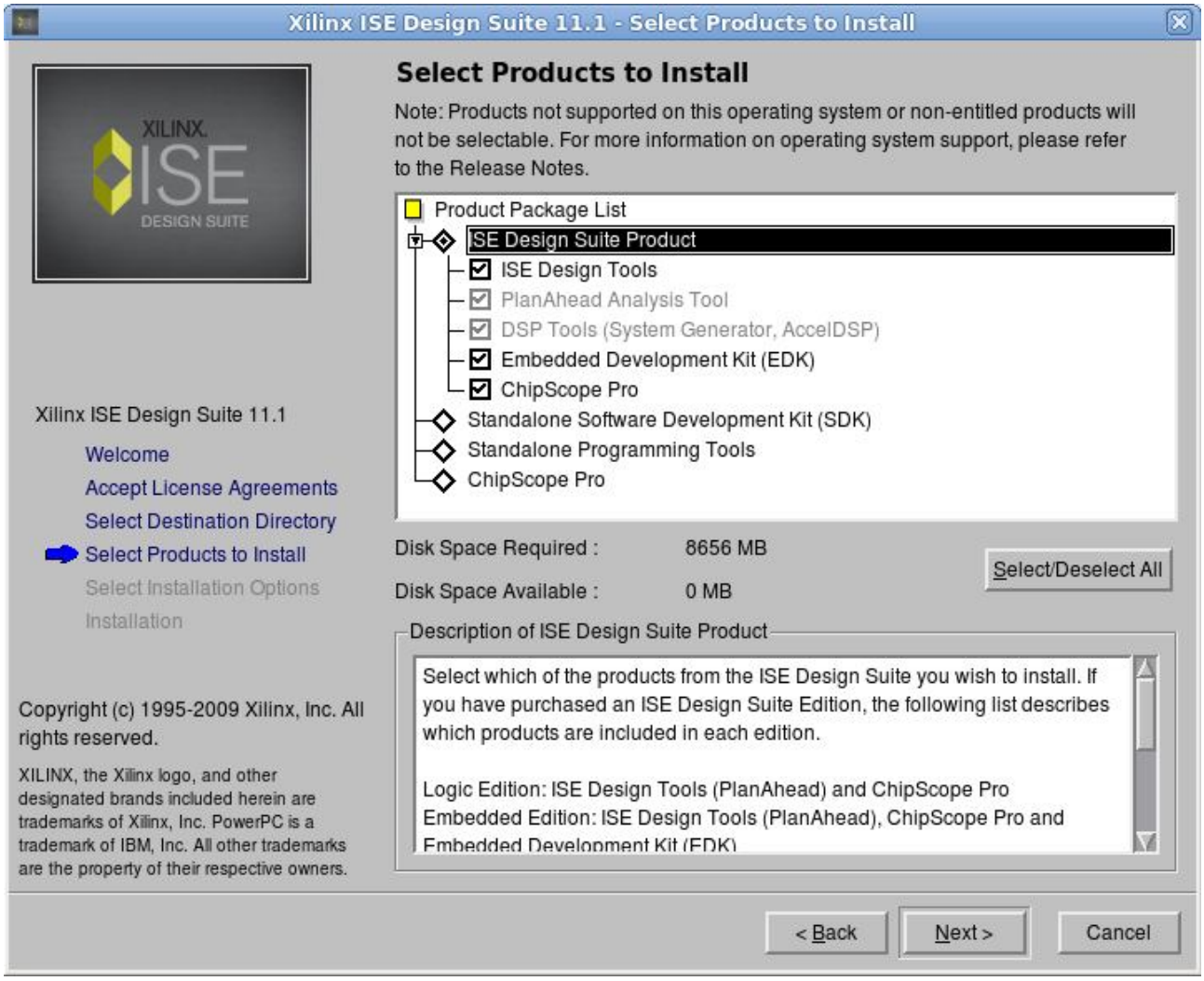



Figura 3.6.1.5[14]

Seleccionando el apartado completo **ISE Design Suite Product** se tiene el software con el cual se trabajar que es **ISE Design Tools**. Mención aparte merece la aplicación **ChipScope pro**, que permite la verificación en tiempo real de los FPGAs. Le tamaño de la instalación dependerá obviamente de la cantidad de software seleccionado, en este caso se pide aproximadamente 9GB de espacio en disco libre.

14 Cuadro de selección de software a instalar.

A continuación del programa elegido se puede seleccionar las características individuales que todo el paquete.

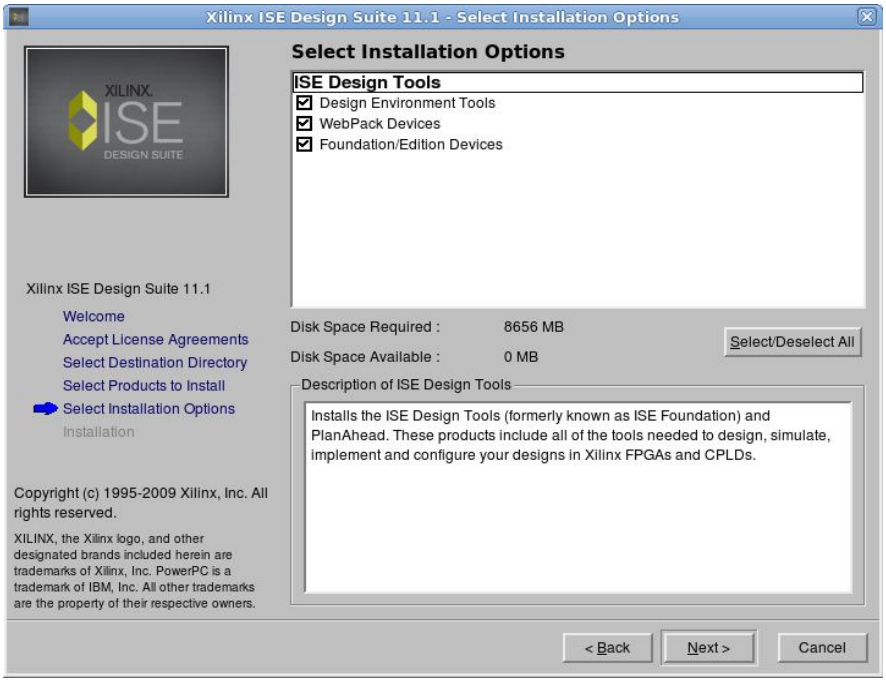



Figura 3.6.1.5[15]

Después se observa la pantalla en la que se nos informa de las rutas en donde se instalarán los programas y contenidos de las diferentes herramientas que se han seleccionado.

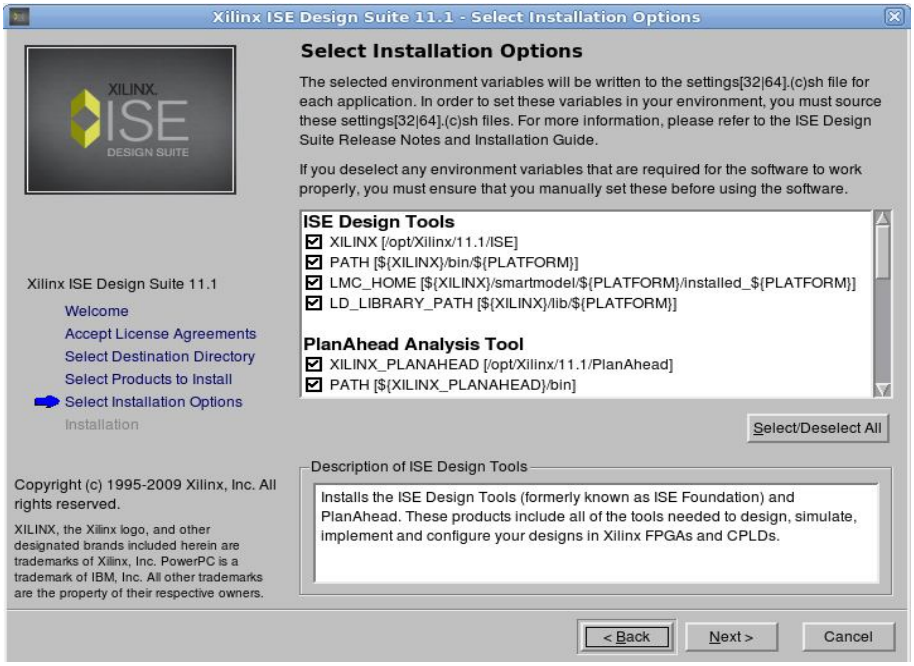



Figura 3.6.1.6[16]

15 Cuadro de selección de características extras.
16 Informe de rutas en nuestro sistemas.

La pantalla siguiente muestra un resumen de los programas a instalar , rutas ,, e información de carácter general.

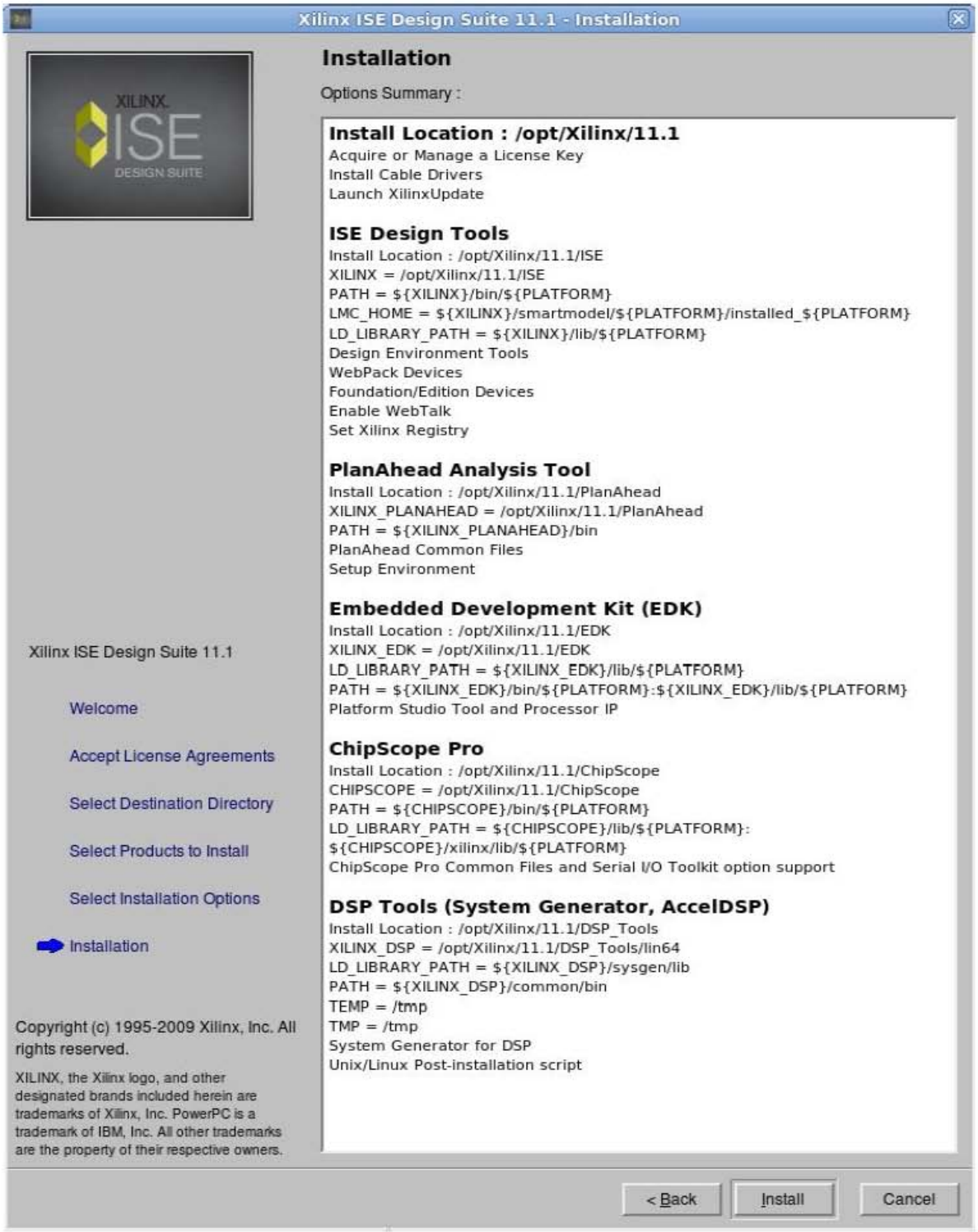



Figura 3.6.1.7[17]

17 Cuadro de diálogo final con información general de la instalación.

Una ves que se da la orden de instalar el programa tarda aproximadamente 15 minutos en instalar toda la paquetería, durante este tiempo es visible el porcentaje de instalación e indica el proceso que se esta llevando a cabo por si es que surge algún problema.

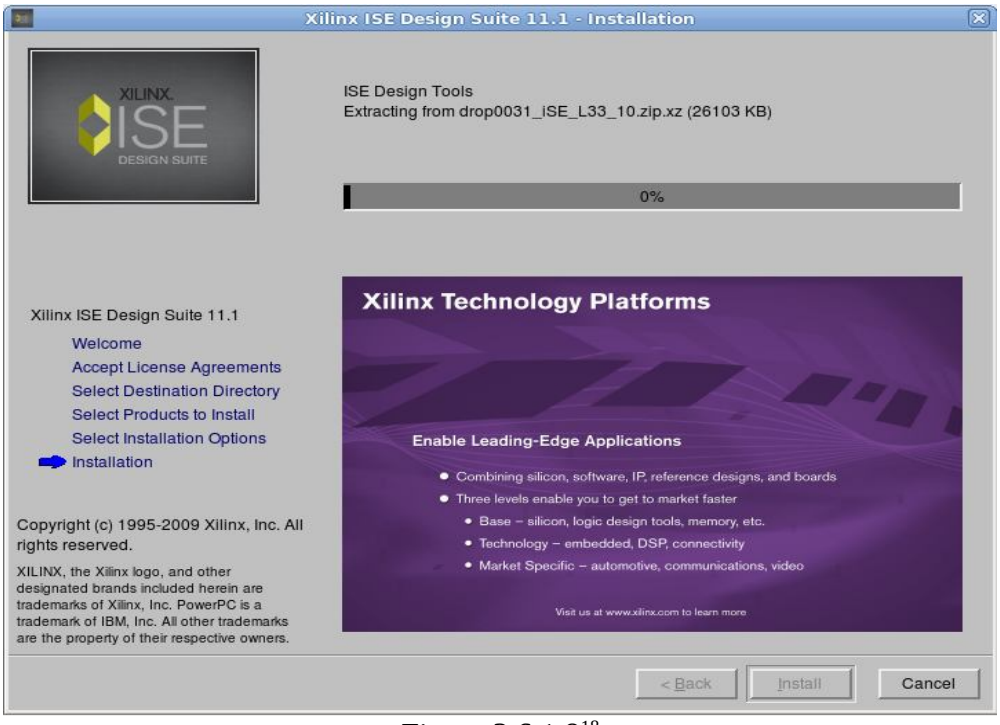



Figura 3.6.1.8[18]

Finalizada la instalación se muestra una pantalla en la cual se nos dice que tipo de licencia de uso disponemos. En este caso se selecciona el uso de evaluación de 30 días, el cual es tiempo suficiente para poder llevar a cabo las tareas que se tienen planeadas .


Figura 3.6.1.9[19]

18 Cuadro de progreso.
19 Recuadro de diálogo, solicitando la información acerca de la licencia.

# 4 Implementación

Una ves que se obtuvo y se instalaron las herramientas necesarias, se procede a conseguir los archivos fuente del núcleo T80. Descargar los mismo y demostrar la forma en que estos archivos se configuran, agregan y sintetizan. En este apartado también se abordarán los temas necesarios para que el microprocesador ya sintetizado pueda funcionar. Esto es al núcleo IP agregarle memoria RAM , la memoria ROM conteniendo los programas así como la circuitería esquemática extra para su puesta en marcha. Se asignarán las patillas del FPGA para que este pueda tener comunicación con el exterior y una ves que todo ello se haya llevado a cabo de manera satisfactoria, se harán pruebas programando el micro. Todo ello usando las herramientas de software del apartado anterior .

## 4.1 Mapa de Memoria de T80

El núcleo T80 dispone de 16 líneas de dirección, por lo tanto tiene la capacidad de direccionar 64 Kbytes de mapa de memoria, comenzando en la dirección hexadecimal 0x0000 y terminando en la dirección 0xffff. Se configurará la memoria de la forma siguiente

| | Dirección de inicio y final del bloque (hexadecimal) | Dirección de inicio y final del bloque (decimal) |
|---|---|---|
| Memoria ROM | 0x0000<br>0x07ff | 0000<br>2047 |
| Rengo de Memoria Disponible | 0x0800<br>0xf7ff | 2048<br>63487 |
| Memoria RAM | 0xf800<br>0xffff | 63488<br>65535 |

## 4.2 Búsqueda y Descarga de archivos del IP Core T80

Se procede a buscar , descargar y descomprimir el IP Core del Microprocesador T80. Para ello se puede acceder a la página electrónica, ya sea solicitando una búsqueda en un motor www.google.com o www.yahoo.com o en su caso acceder directamente a la página.

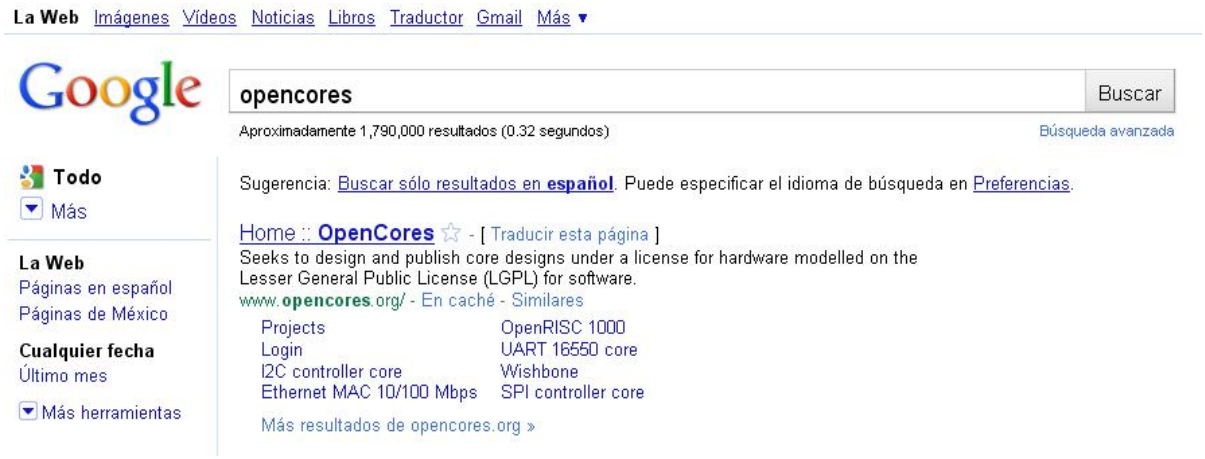



Figura 4.2.1[20]

Accediendo a la página electrónica en www.opencores.org se tiene la siguiente pantalla de la página.

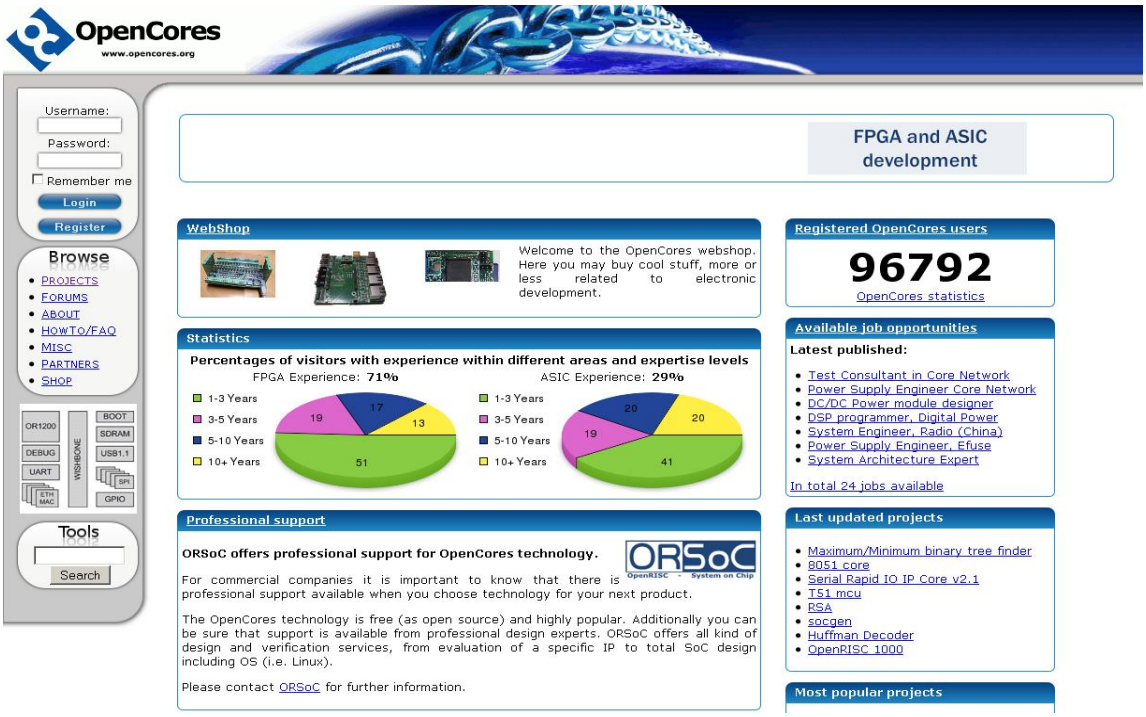



Figura 4.2.2[21]

20 Buscador Google después de realizar la búsqueda con el nombre opencores
21 página electrónica principal de www.opencores.org

### 4.2.1 Registro y acceso al sitio web

Dentro del sitio se puede navegar sin ningún tipo de restricción, sin embargo para acceder al área de descarga de todos los proyectos es necesario registrarse. Hacerlo toma poco menos de 5 minutos y una ves que se tiene el registro , basta el nombre de usuario y la contraseña que asigna el sistemas.


Figura 4.2.1.1[22]

Una ves que el registro se ha completado con éxito y la página nos identifica por nuestro nombre de usuario podemos acceder a la página en donde están listados todos los proyectos.

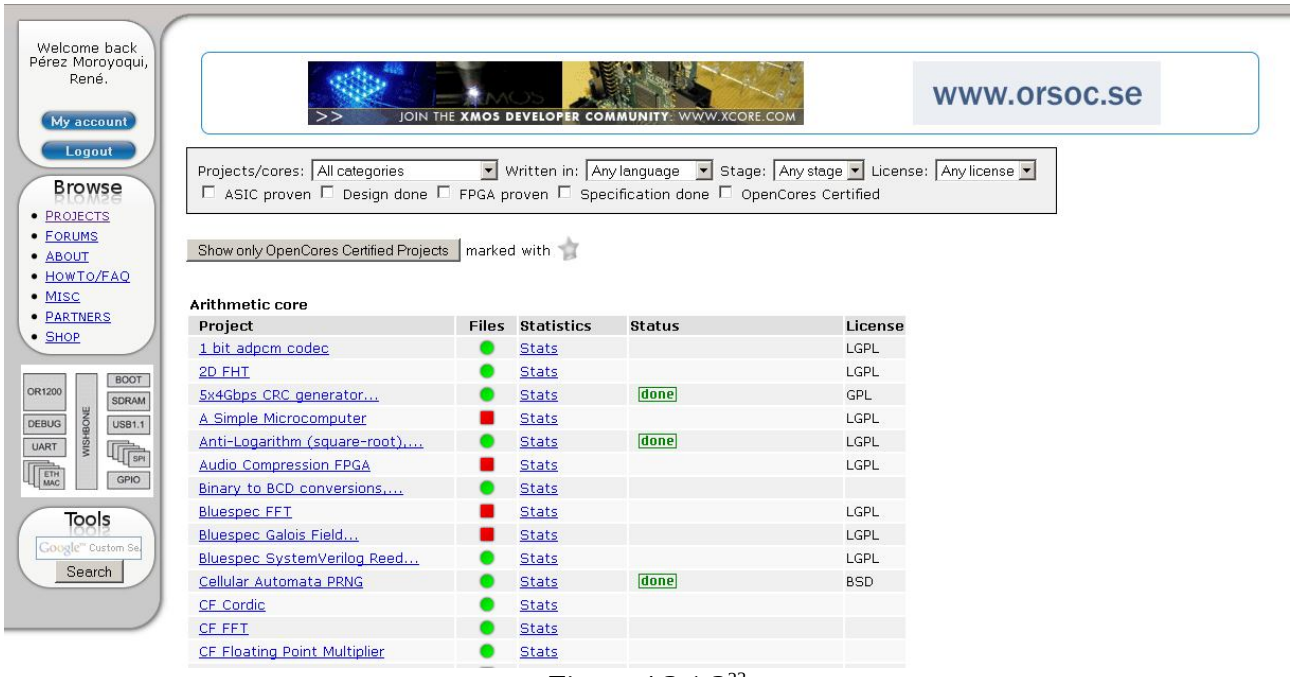
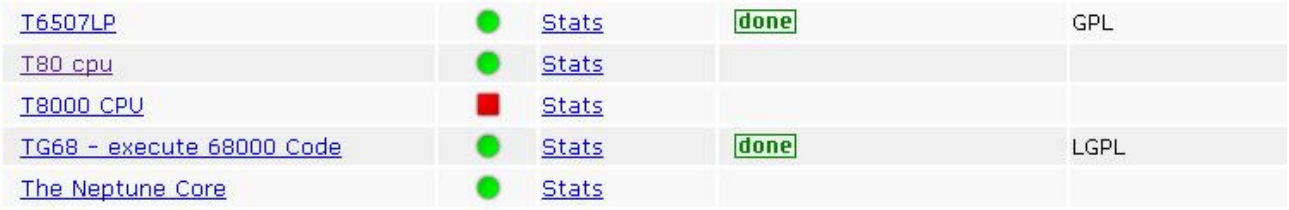



Figura 4.2.1.2[23]

22 Imágenes describiendo los pasos para acceder registrado al sitio.
23 Listado de proyectos dentro de la página.

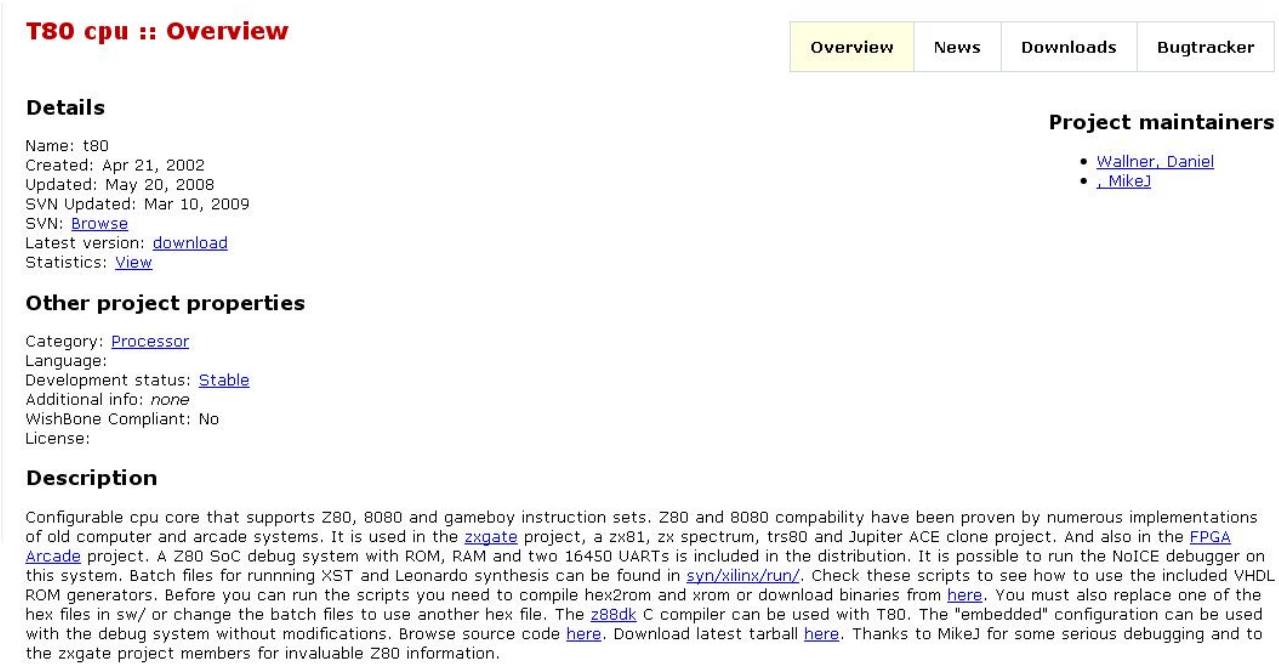
Se localiza el proyecto de nombre T80 CPU y se da click cobre el



| | | | | |
|---|---|---|---|---|
| T6507LP | | Stats | done | GPL |
| T80 cpu | | Stats | | |
| T8000 CPU | | Stats | | |
| TG68 - execute 68000 Code | | Stats | done | LGPL |
| The Neptune Core | | Stats | | |

Figura 4.2.1.3[24]

Esta es la página particular del proyecto T80 conteniendo una pequeña descripción , las personas que lo mantienen , el nombre del proyecto, la última fecha de actualización y el formato de archivos en los que esta programado ya sea VHDL o Verilog. Entre las secciones que se aprecian se encuentra la sección **Downloads** que es en donde encontraremos los archivos de este proyecto.



**T80 cpu :: Overview**

Overview | News | Downloads | Bugtracker

**Details**

Name: t80
Created: Apr 21, 2002
Updated: May 20, 2008
SVN Updated: Mar 10, 2009
SVN: Browse
Latest version: download
Statistics: View

**Other project properties**

Category: Processor
Language:
Development status: Stable
Additional info: *none*
WishBone Compliant: No
License:

**Project maintainers**

- Wallner, Daniel
- , MikeJ

**Description**

Configurable cpu core that supports Z80, 8080 and gameboy instruction sets. Z80 and 8080 compability have been proven by numerous implementations of old computer and arcade systems. It is used in the zxgate project, a zx81, zx spectrum, trs80 and Jupiter ACE clone project. And also in the FPGA Arcade project. A Z80 SoC debug system with ROM, RAM and two 16450 UARTs is included in the distribution. It is possible to run the NoICE debugger on this system. Batch files for runnning XST and Leonardo synthesis can be found in syn/xilinx/run/. Check these scripts to see how to use the included VHDL ROM generators. Before you can run the scripts you need to compile hex2rom and xrom or download binaries from here. You must also replace one of the hex files in sw/ or change the batch files to use another hex file. The z88dk C compiler can be used with T80. The "embedded" configuration can be used with the debug system without modifications. Browse source code here. Download latest tarball here. Thanks to MikeJ for some serious debugging and to the zxgate project members for invaluable Z80 information.

Figura 4.2.1.4[25]

24 Localización del proyecto T80 dentro del listado.
25 Área de entrada de la página del proyecto T80. En donde se describe el proyecto, actualizaciones, autor, etc.

### 4.2.2 Descarga del archivo

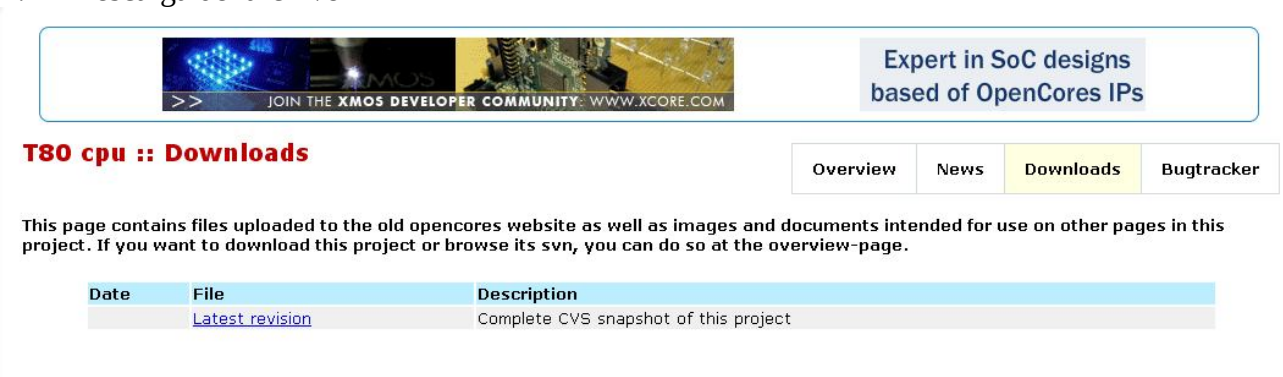



Figura 4.2.2.1[26]

Dando click sobre el link del repositorio se puede guardar el archivo.

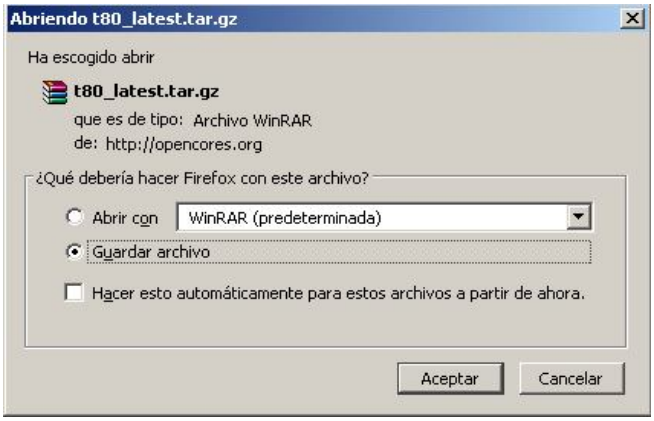



Figura 4.2.2.2[27]

La zona de descarga de este microprocesador en particular solo consta de un archivo comprimido en formato RAR el cual contiene los archivos que se necesitan para implementarlo. Descargado el archivo se descomprime en una carpeta en la que se tenga planeado el desarrollo de todo el sistema. Con esto la descarga y la descompresión de los archivos ha concluido.

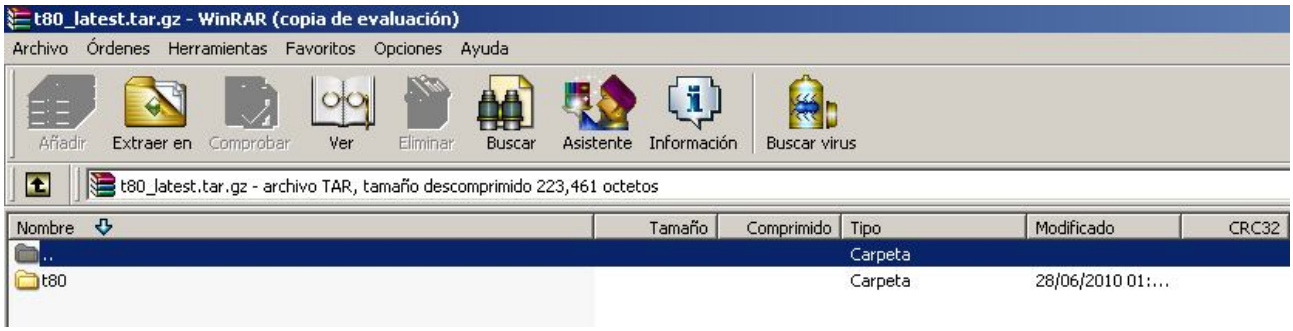



Figura 4.2.2.3[28]

26 Área de descargas del proyecto T80
27 Cuadro de selección del área donde se descargara el archivo.
28 Archivo t80_lastest.tar.gz abierto con el des-compresor Winrar.

### 4.2.3 Árbol de directorios del contenido del archivo comprimido.

```
--t80
   |-- branches
   |-- tags
   |-- trunk
   |  |-- bench
   |  |  `-- vhdl
   |  |     |-- AsyncLog.vhd
   |  |     |-- AsyncStim.vhd
   |  |     |-- DebugSystem_TB.vhd
   |  |     |-- SRAM.vhd
   |  |     |-- StimLog.vhd
   |  |     `-- TestBench.vhd
   |  |-- rtl
   |  |  `-- vhdl
   |  |     |-- DebugSystem.vhd
   |  |     |-- DebugSystemXR.vhd
   |  |     |-- SSRAM2.vhd
   |  |     |-- SSRAM.vhd
   |  |     |-- SSRAMX.vhd
   |  |     |-- T16450.vhd
   |  |     |-- T8080se.vhd
   |  |     |-- T80_ALU.vhd
   |  |     |-- T80a.vhd
   |  |     |-- T80_MCode.vhd
   |  |     |-- T80_Pack.vhd
   |  |     |-- T80_Reg.vhd
   |  |     |-- T80_RegX.vhd
   |  |     |-- T80se.vhd
   |  |     |-- T80s.vhd
   |  |     `-- T80.vhd
   |  |-- sim
   |  |  `-- rtl_sim
   |  |     `-- bin
   |  |        |-- compile.do
   |  |        `-- RX_Cmd1.txt
   |  |-- sw
   |  |  |-- hex2rom.cpp
   |  |  |-- sine.bin
   |  |  `-- xrom.cpp
   |  `-- syn
   |     `-- xilinx
   |        |-- bin
   |        |  |-- t80debug.pin
   |        |  |-- t80debug.prj
   |        |  |-- t80debug.scr
   |        |  |-- t80debug.tcl
   |        |  |-- t80debugxr_leo.pin
   |        |  |-- t80debugxr.pin
   |        |  |-- t80debugxr.prj
   |        |  |-- t80debugxr.scr
   |        |  |-- t80debugxr.tcl
   |        |  |-- t80_leo.pin
   |        |  |-- t80.pin
   |        |  |-- t80.prj
   |        |  |-- t80.scr
   |        |  `-- t80.tcl
   |        |-- log
   |        |-- out
   |        |-- run
   |        |  |-- t80.bat
   |        |  |-- t80debug.bat
   |        |  |-- t80debug_leo.bat
   |        |  |-- t80debugxr.bat
   |        |  |-- t80debugxr_leo.bat
   |        |  `-- t80_leo.bat
   |        `-- src
   `-- web_uploads
```

## 4.3 Arranque del Programa de Desarrollo Xilinx ISE

Se ejecuta el **Project Navigator** de **Xilinx ISE Design Suite 11.1**

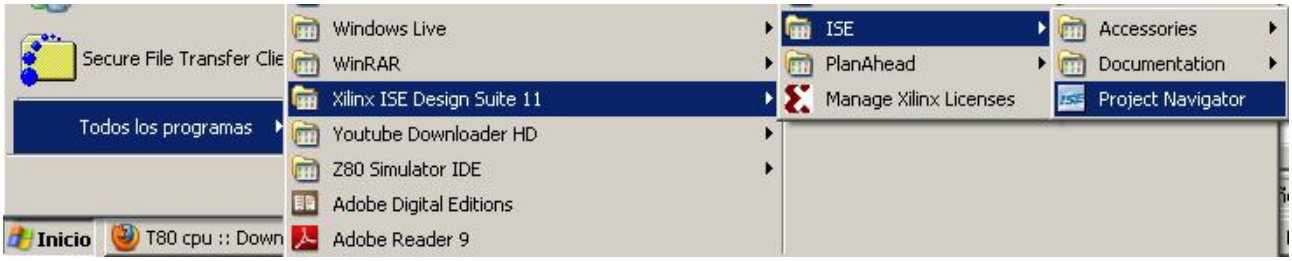



Figura 4.3.0.1[29]

Esta es la pantalla de inicio del programa

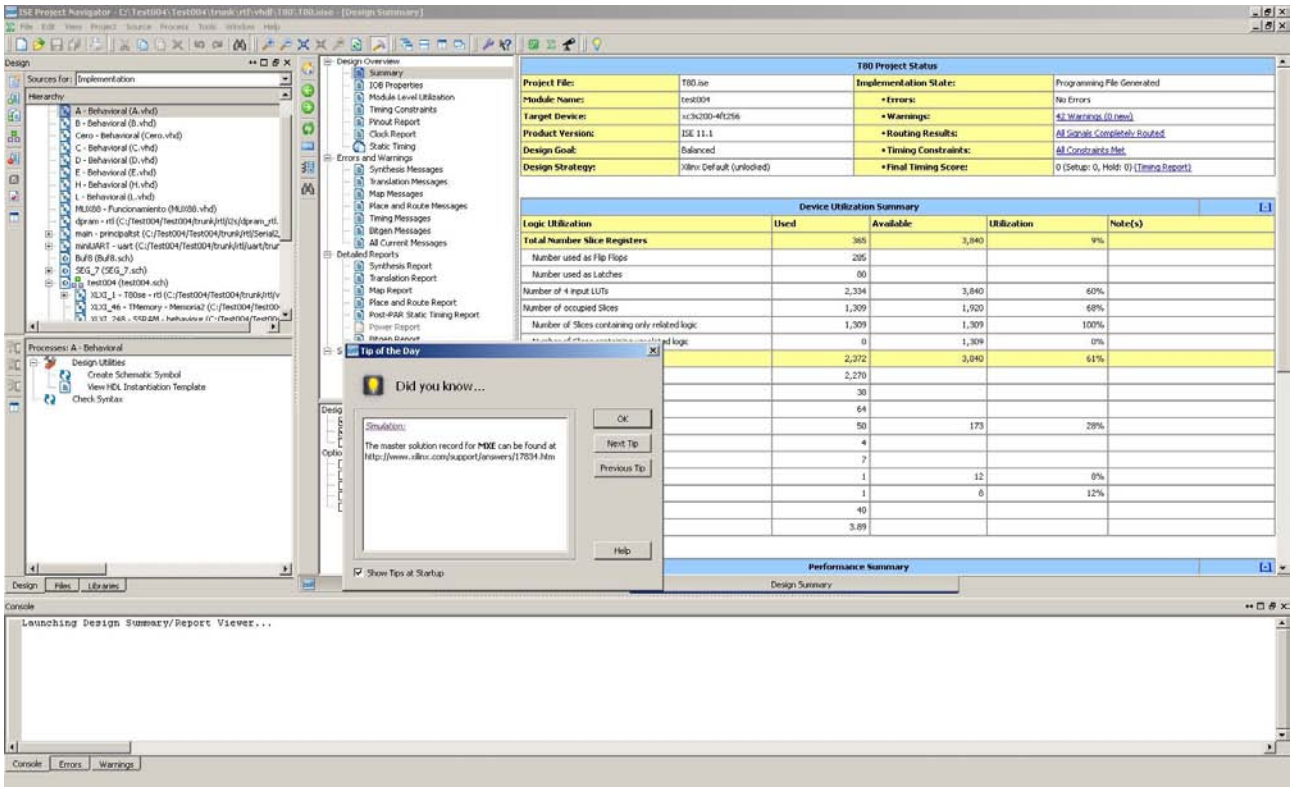



Figura 4.3.0.2[30]

29 Localización dentro de la barra de menú de Windows de la Xilinx ISE Design Suite 11.1
30 Pantalla de bienvenida.

### 4.3.1 Crear un nuevo proyecto

Se inicia con un nuevo proyecto a partir de cero, se ejecuta **File->New Proyect**

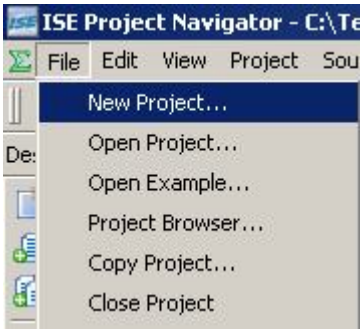



Figura 4.3.1.1[31]

En el siguiente paso se elije el nombre del proyecto, su ubicación y si es posible se escribe una pequeña descripción del proyecto. Se elije además el lenguaje preferente en el que programará, en este caso VHDL.

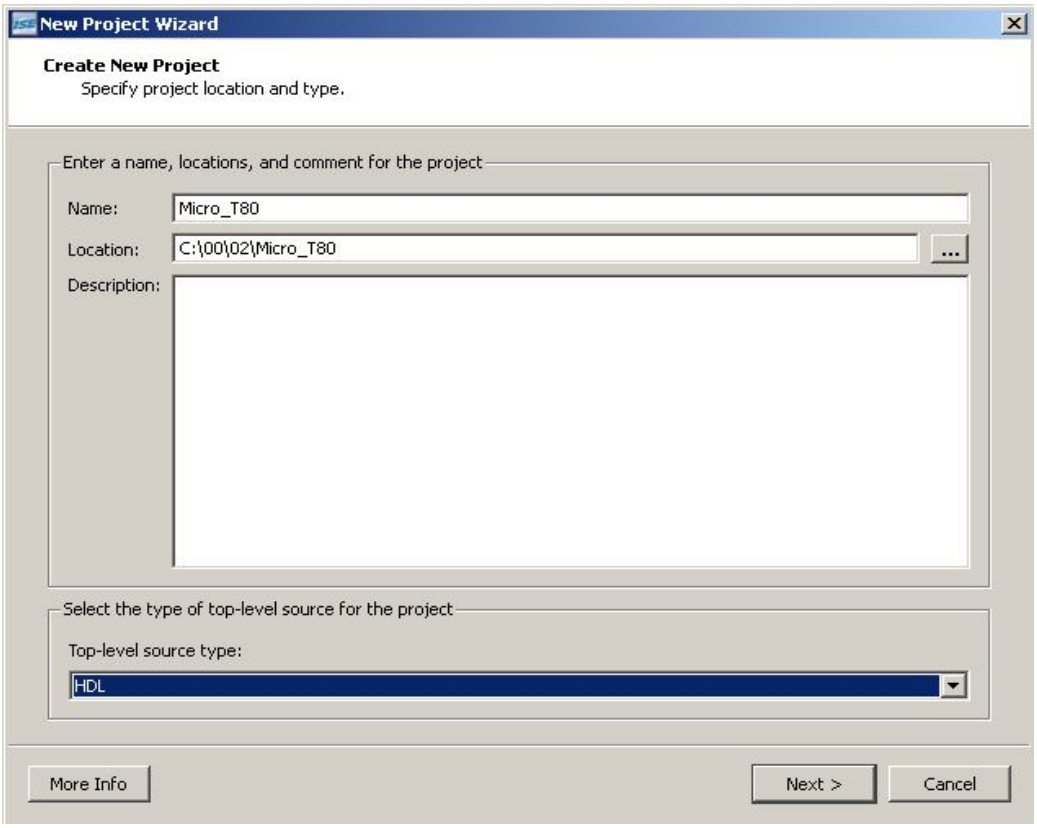



Figura 4.3.1.1.2[32]

31 Forma de crear un nuevo proyecto.
32 Cuadro de diálogo, se nombra el proyecto, su ubicación y puede darse una descripción general del mismo.

A Continuación se pregunta por el tipo de producto en el cual se sintetizará el proyecto, en este caso es un FPGA Xilinx Spartan 3 XA3S200, mismo que se encuentra en la tarjeta de desarrollo.

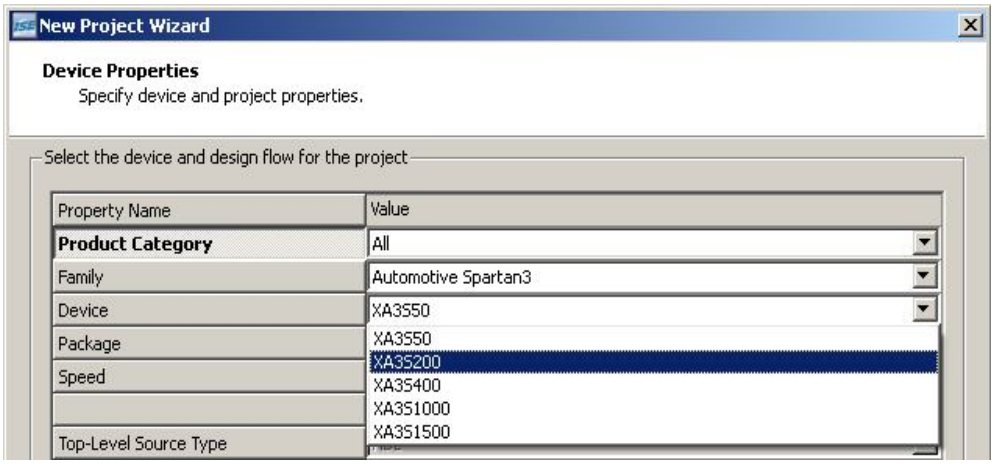



Figura 4.3.1.1.3[33]

Se elige además la forma del empaquetado del circuito dado que un mismo dispositivo puede tener distintas configuraciones de patillaje.

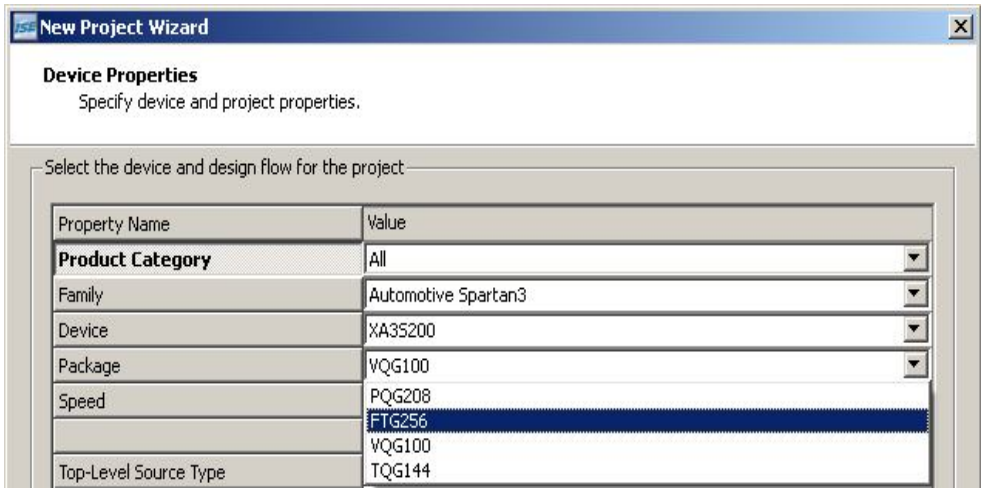



Figura 4.3.1.1.4[34]

33 Cuadro de diálogo para elegir la familia se FPGA en la cual se sintetizara el proyecto.
34 Cuadro de diálogo para elegir el formato de empaquetado del dispositivo.

Finalizado los pasos anteriores se muestra un resumen del proyecto e información general.

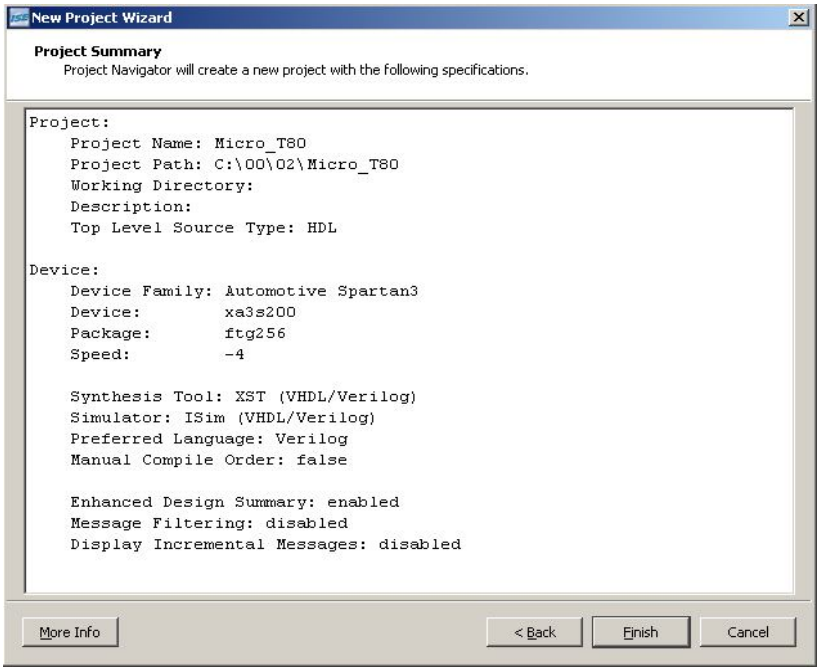



Figura 4.3.1.1.5[35]

El espacio de trabajo se muestra a continuación, en la parte superior izquierda solo se muestra el nombre del proyecto y el dispositivo sobre el cual se trabajará.

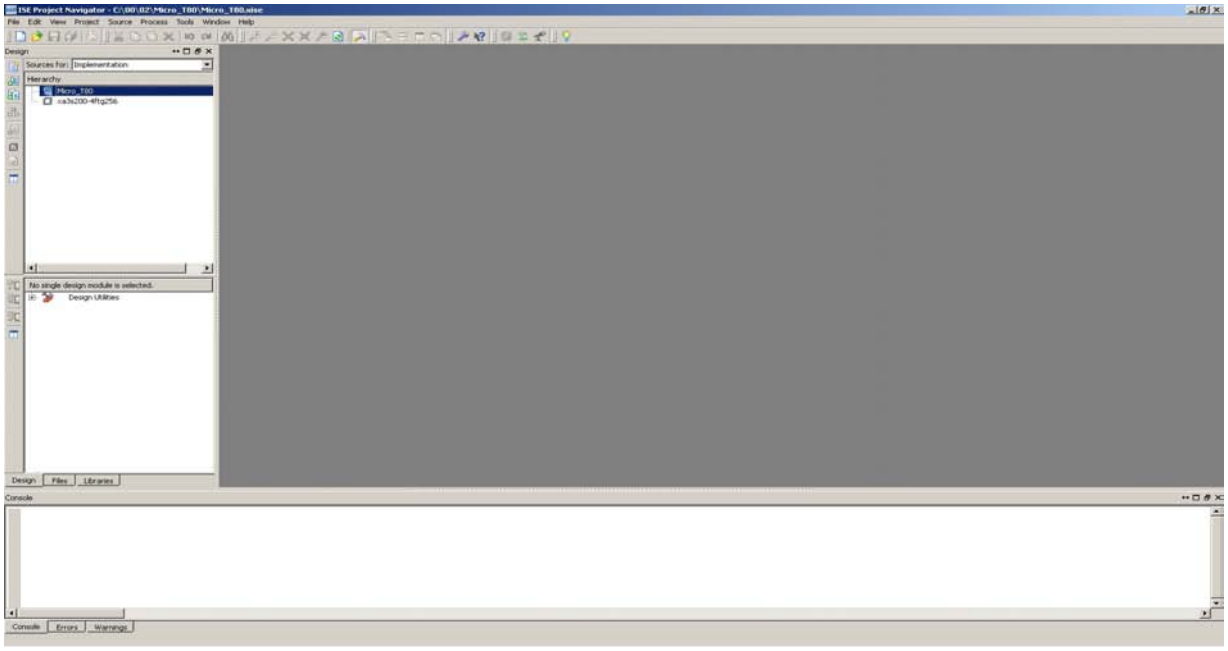


Figura 4.3.1.1.6[36]

35 Ventana de información general del proyecto una vez que se ha concluido la creación del mismo.
36 Pantalla de inicio del área de trabajo

### 4.3.2 Agregar archivos del IP Core T80 al Proyecto

Se agregarán los archivos principales del núcleo T80, se coloca el puntero sobre el dispositivo y con el botón derecho se despliega el menú de selección y se elige **Add Source.** Puede encontrarse en el apéndice de este trabajo, el listado de todos los archivo que se agregan al proyecto.

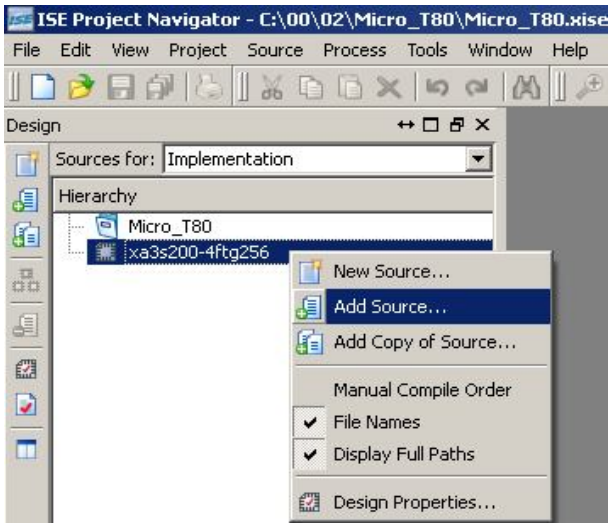



Figura 4.3.2.1[37]

La ventana de diálogo se abre para dar una ruta al archivo que se agregará. Se elige el primer término el archivo T80.vhd[38] que se encuentra dentro de la carpeta en la cual se descomprimió el archivo rar.

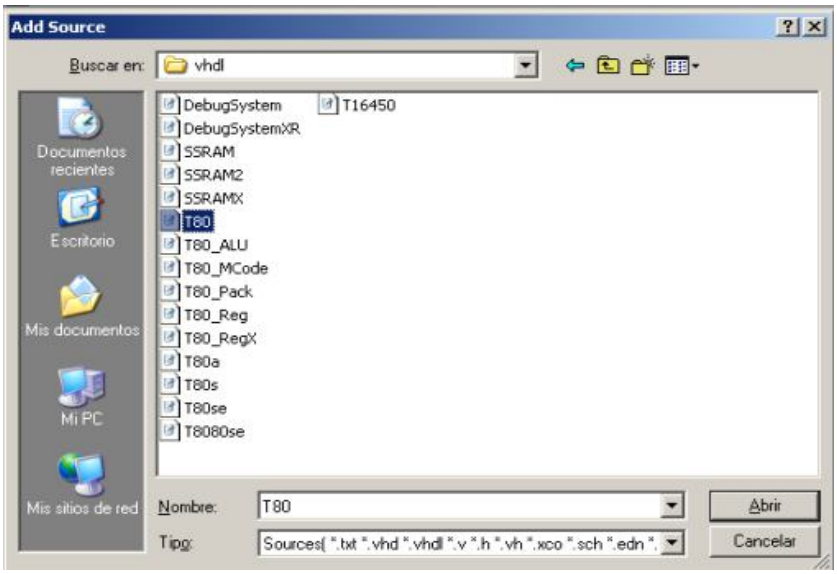



Figura 4.3.2.2[39]

37 Forma de agregar archivos al proyecto
38 El contenido del archivo T80.vhd puede consultarse en el apéndice, desde la página 90 hasta la 105.
39 Cuadro de navegación para encontrar el archivo T80.vhd

Una vez seleccionado y aceptado el programa verifica la sintaxis del archivo y notifica si se ha encontrado problema alguno, si no es así entonces marcara el archivo como valido.

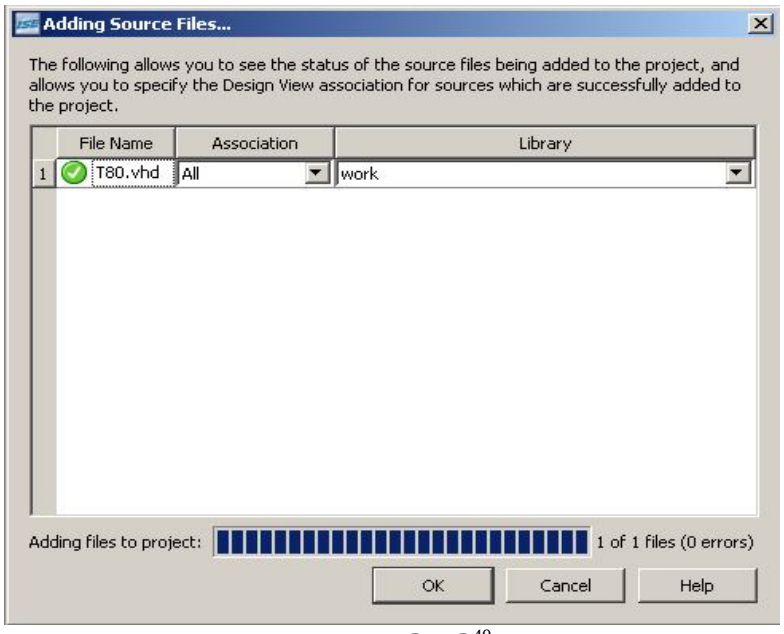



Figura 4.3.2.3[40]

En la ventana de archivo agregados se puede ver ahora la etiqueta T80 y bajo esta varias etiquetas de archivo que se deben de agregar ya que estos son invocados dentro del archivo T80.vhd.

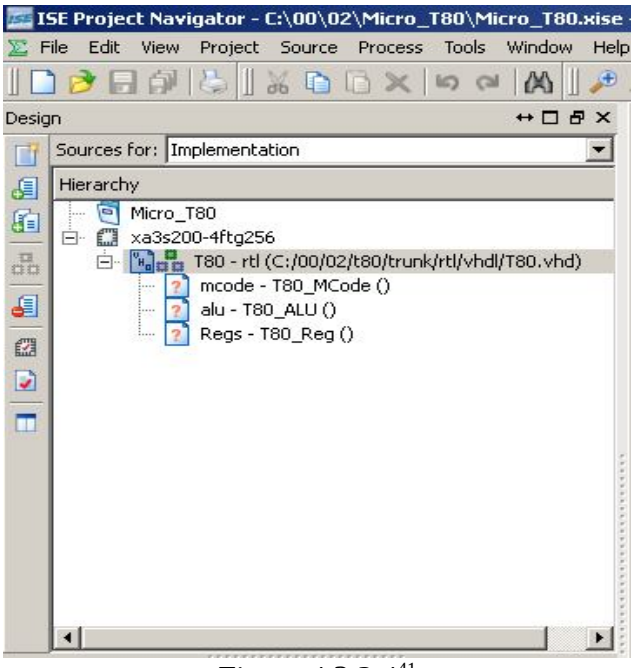



Figura 4.3.2.4[41]

40 Cuadro de información acerca del archivo agregado. En este caso no se detectaron problemas.
41 Área de trabajo. Se observa que el apartado T80 cambia de icono.

Se seguirá el mismo procedimiento anterior para agregar un archivo, se coloca el puntero sobre la etiqueta del mismo, se acciona el botón derecho y se ejecuta **Add Source**.

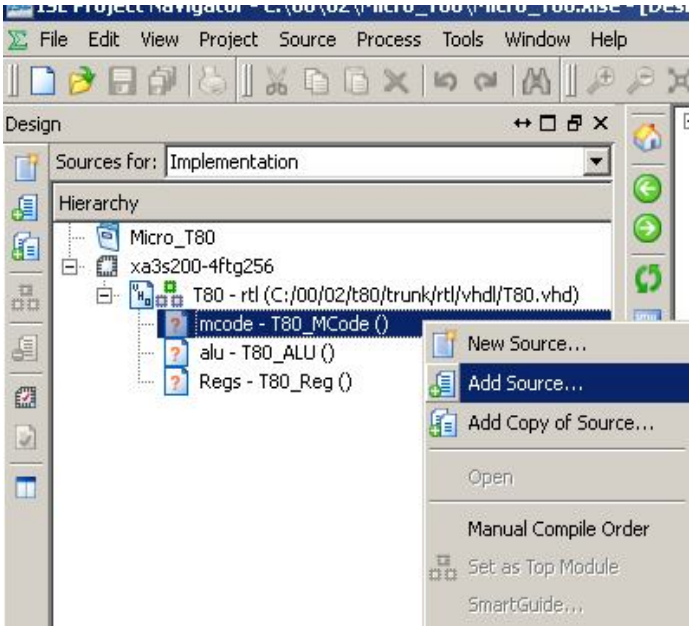



Figura 4.3.2.5[42]

En este caso se busca y se agrega el archivo T80_MCode.vhd[43]

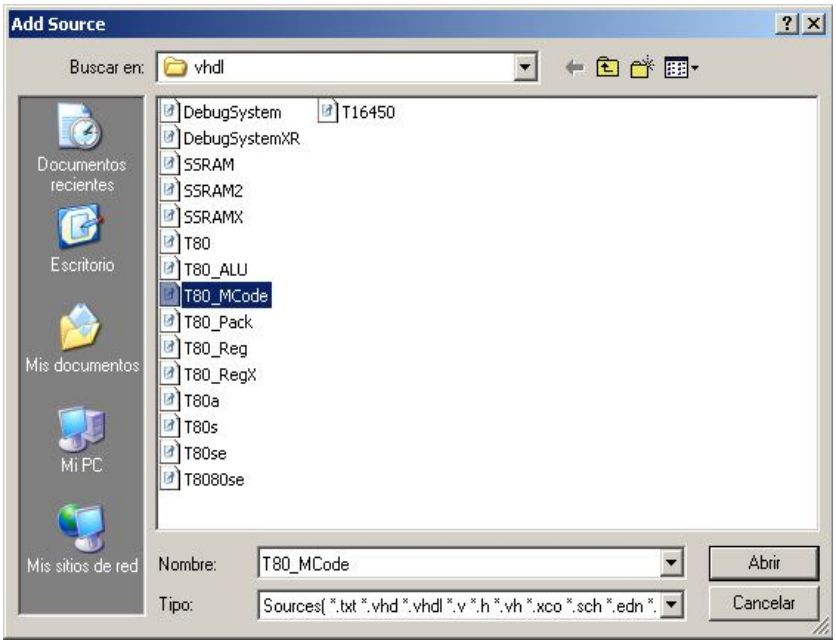



Figura 4.3.2.6[44]

42 Forma de agregar el archivo T80_MCode.vhd al proyecto.
43 El contenido del archivo T80_MCode.vhd puede consultarse en el apéndice, desde de la página 106 hasta la 133.
44 Menú de navegación para localizar y agregar el archivo.

Se verifica la integridad de la sintaxis.

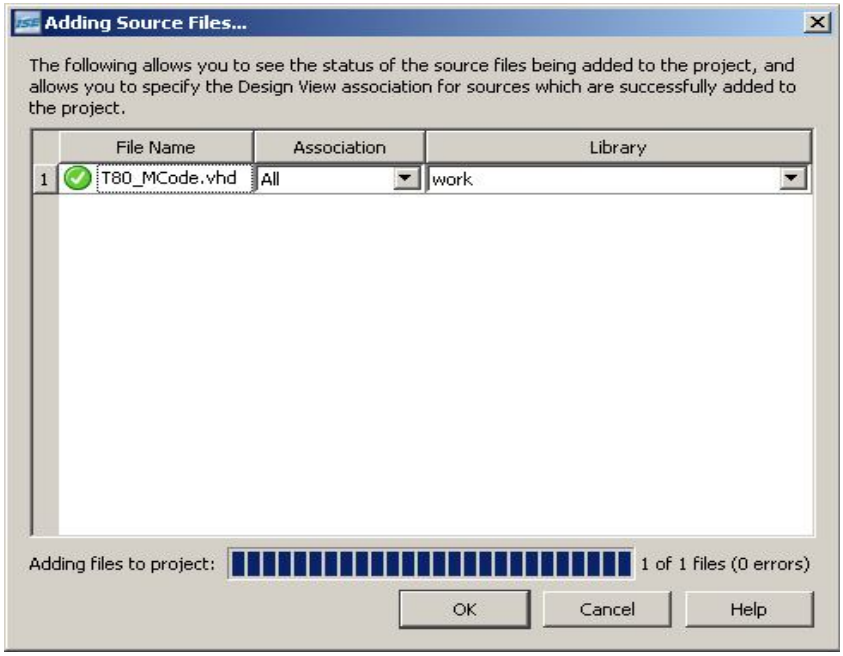



Figura 4.3.2.4[45]

Después de ello se puede observar que la etiqueta mcode se encuentra ya asociada a un archivo VHDL a diferencia de las etiquetas ALU y Regs. Se procede a continuación a agregar el archivo correspondiente con con la etiqueta alu – T80_ALU( )

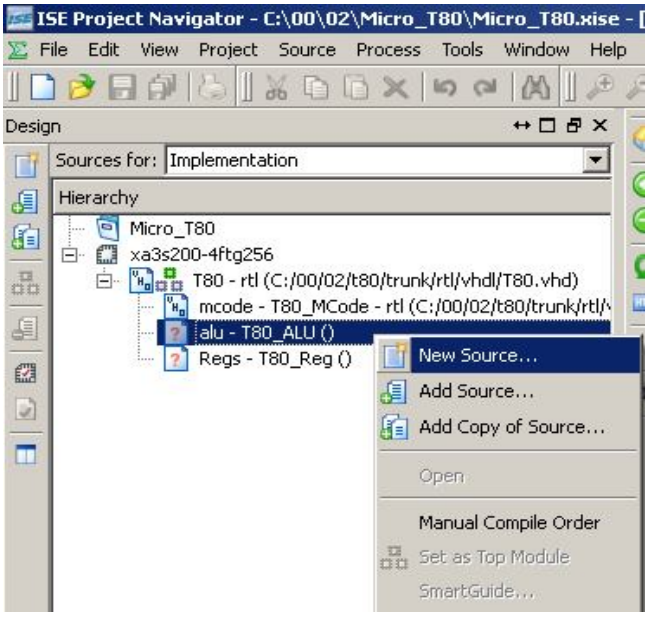



Figura 4.3.2.7[46]

45 Cuadro de verificación del archivo T80_MCODE.vhd.
46 Forma de agregar el archivo T80_ALU.vhd al proyecto.

Se busca  y se agrega el archivo T80_ALU.vhd[47].

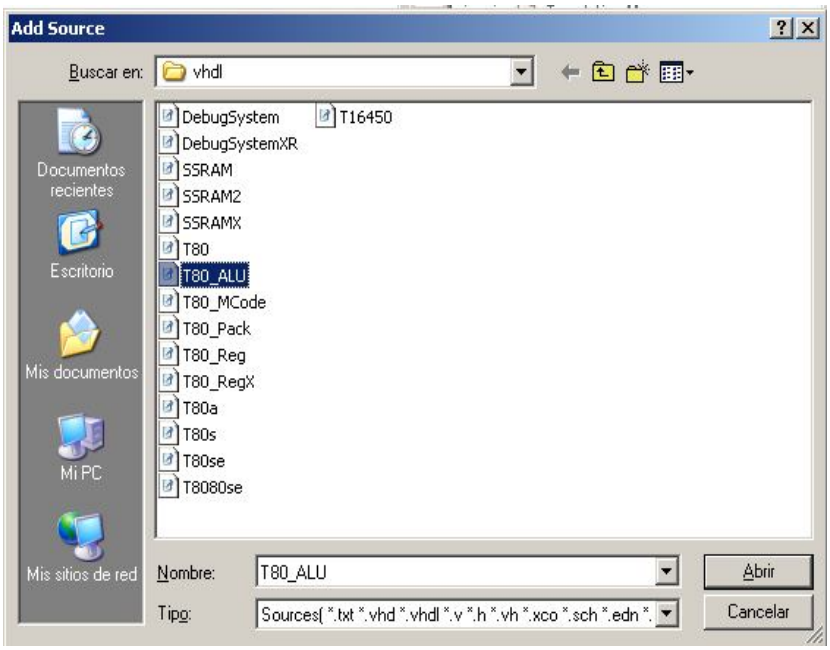



Figura 4.3.2.8[48]

Sigue la verificación automática de la sintaxis.

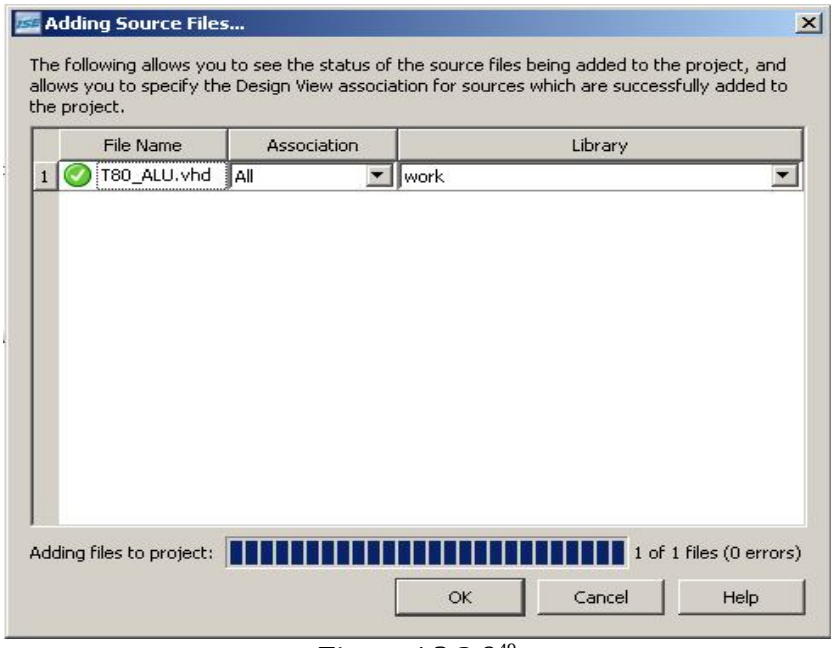



Figura 4.3.2.9[49]

47 El contenido del archivo T80_ALU.vhd puede consultarse en el apéndice, desde de la página 134 hasta la 138.
48 Menú de navegación para localizar y agregar el archivo T80_ALU.vhd
49 Cuadro de verificación del archivo agregado.

Aceptado el proceso de verificado se observa que la etiqueta ha sido asignada a un archivo vhd y su icono cambia. Se procede a agregar el archivo T80_Reg.vhd[50] que se solicita.

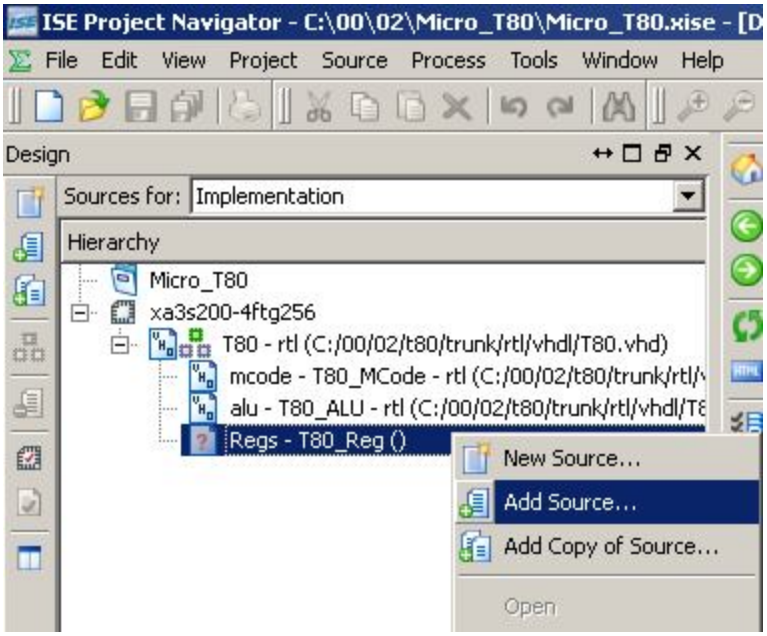



Figura 4.3.2.10[51]

Se localiza el archivo y se elige.

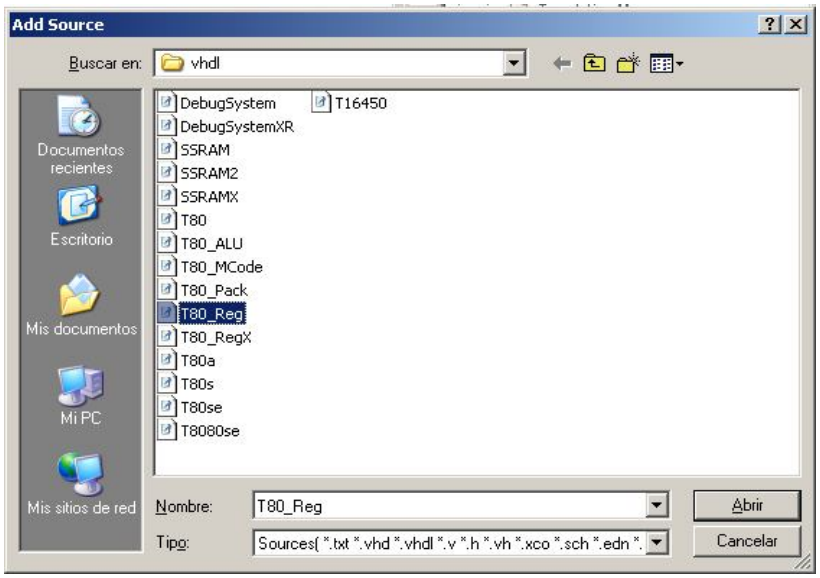



Figura 4.3.2.11[52]

50 El contenido del archivo T80_Reg.vhd puede consultarse en el apéndice, desde de la página 139 hasta la 140
51 Forma de agregar el archivo T80_Red.vhd al proyecto.
52 Ventana de navegación para localizar y agregar el archivo.

Una ves elegido la sintaxis es automáticamente verificada.

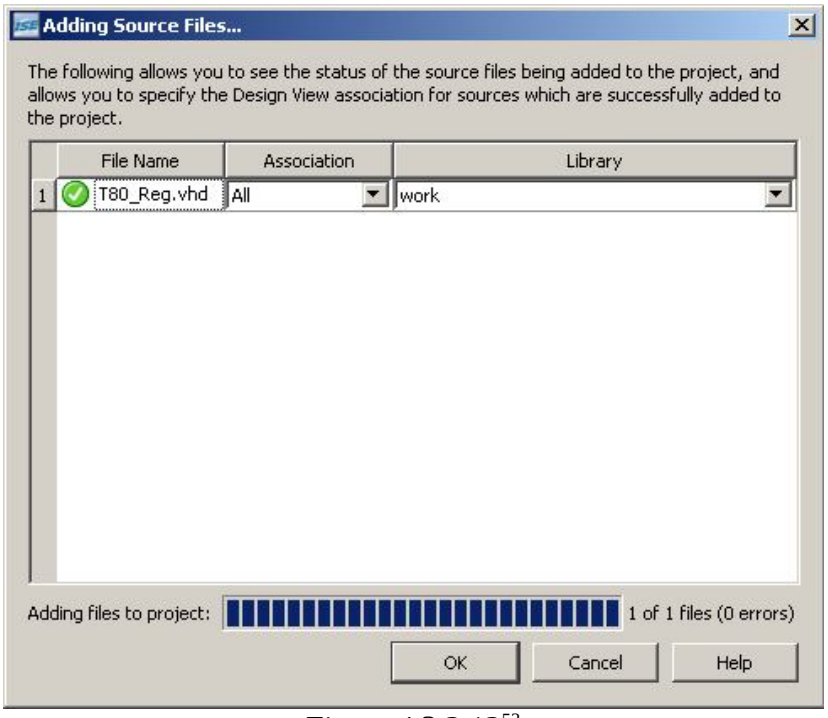



Figura 4.3.2.12[53]

Terminado el proceso de agregar los archivos que se piden, se encuentra la siguiente imagen

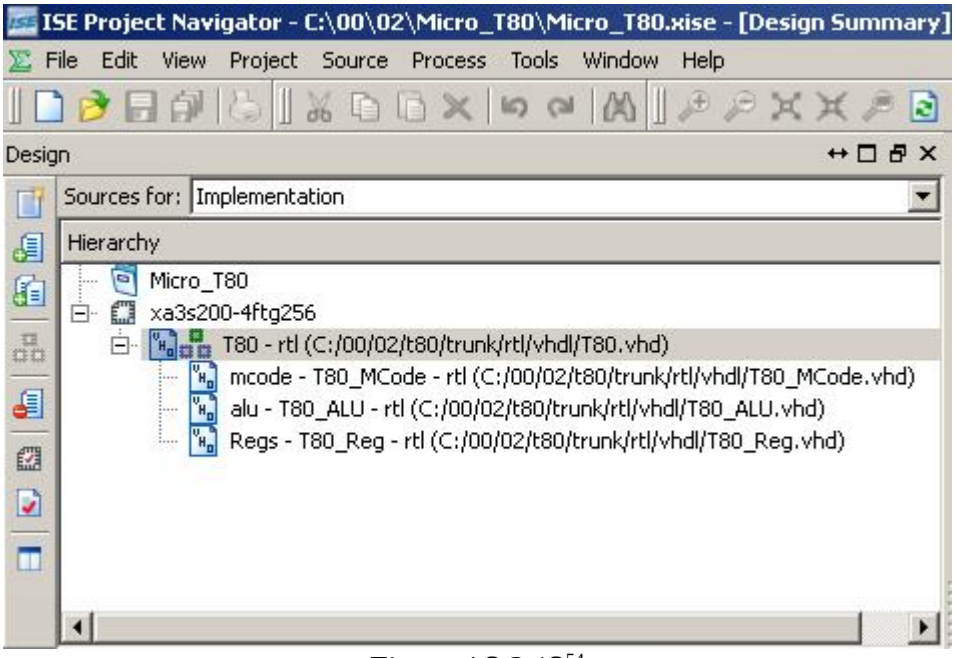



Figura 4.3.2.13[54]

53 Cuadro de verificación del archivo agregado.
54 Archivos principales ya agregados. Los iconos de las etiquetas cambian cuando ya se han agregado los archivos.

Aun no se ha terminado de agregar todos los archivos, si en este momento se ejecuta la herramienta **Synthesize XST** marcará un error ya que pide una librería llamada T80_Pack.vhd[55] la cual debe de estar disponible en las librerías del proyecto, por lo cual nos trasladamos a la biblioteca de las mismas dando clic en la pestaña de **Libraries** que se encuentra en la parte baja del recuadro de proyecto .

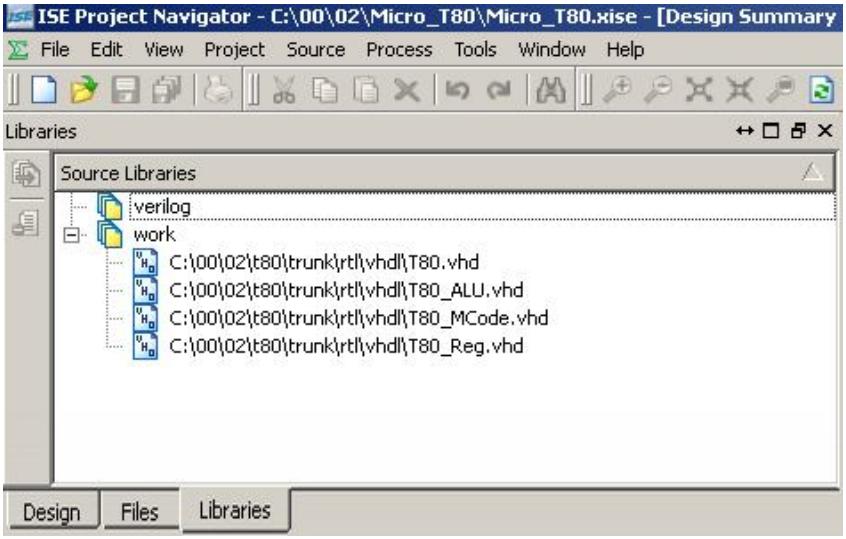



Figura 4.3.2.14[56]

Se siguen los mismos procedimientos anteriores para agregar archivos , en este caso el archivo T80_Pack.vhd


Figura 4.3.2.15[57]

55 El contenido del archivo T80_Pack.vhd puede consultarse en el apéndice, desde de la página 141 hasta la 143
56 Apartado de librerías en el área de trabajo.
57 Forma de agregar una librería. En este caso el archivo T80_Pack.vhd

Se busca el archivo en  al árbol de directorio en donde se descomprimió el archivo

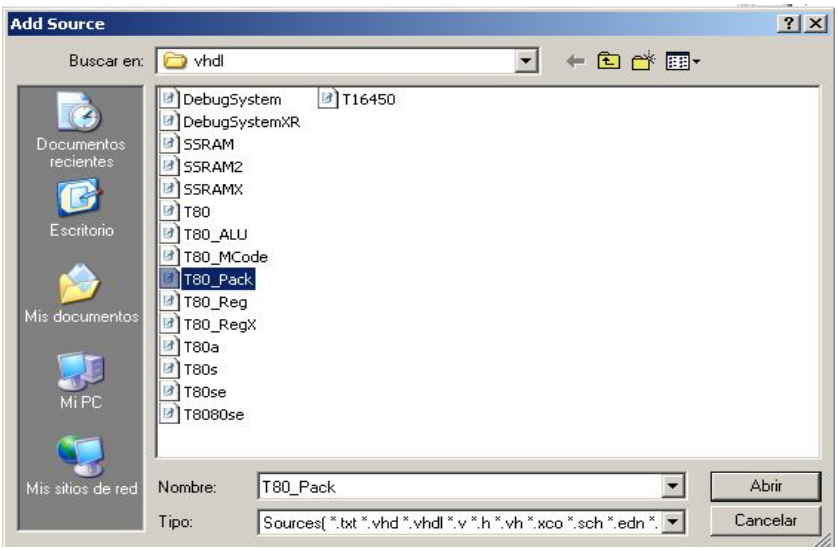



Figura 4.3.2.16[58]

Una ves seleccionado se verifica automáticamente la sintaxis.

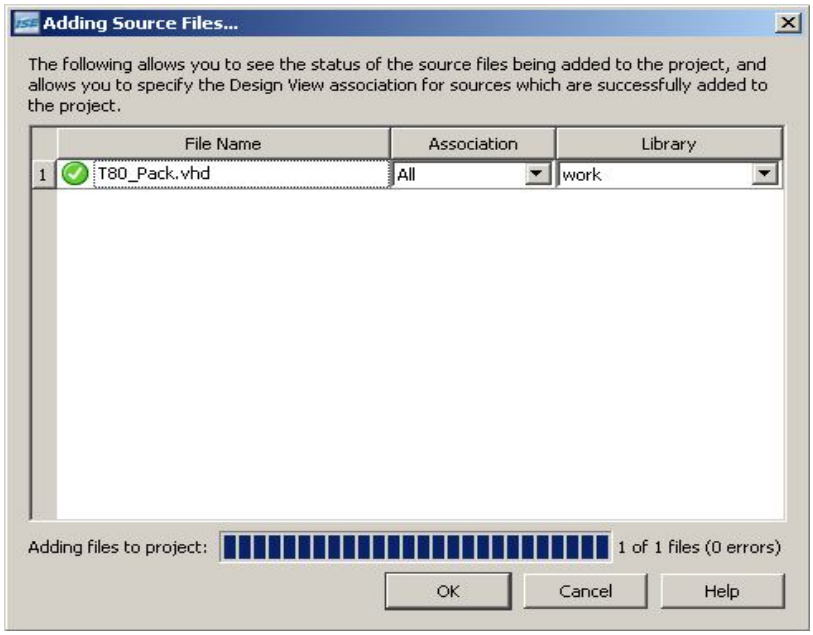



Figura 4.3.2.17[59]

58 Ventana de navegación para  la localización y el agregado del  archivo T80_Pack.vhd  al proyecto
59 Cuadro de verificación del archivo.

Y el archivo T80_Pack de  encuentra dentro de la librería de trabajo.

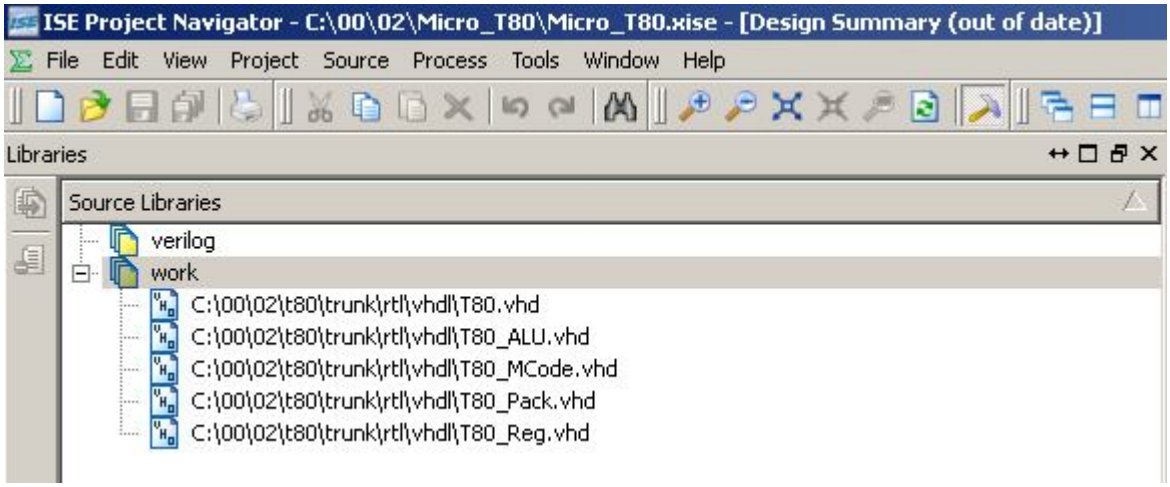



Figura 4.3.2.17[60]

Ahora se agrega el archivo  nombrado T80se.vhd[61] de la misma forma en que se han agregado hasta ahora  archivos al proyecto. Este archivo es el ultimo en esta serie de pasos antes de  esquematizar el microprocesador. Se localiza el archivo T80se.vhd

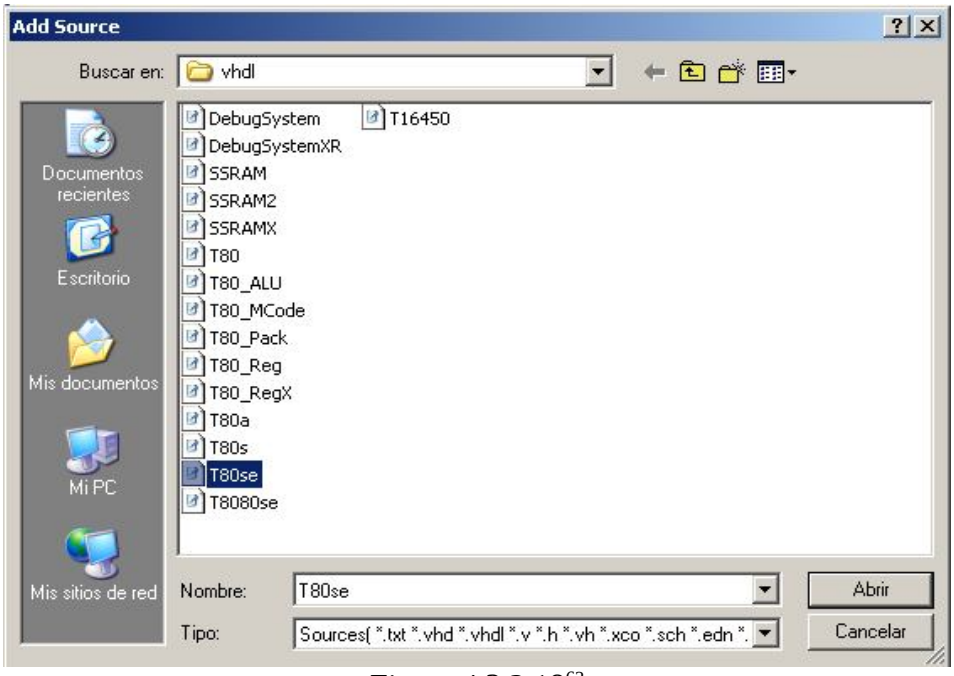



Figura 4.3.2.18[62]

60 Vista de todos los archivos agregados  hasta el momento.
61 El contenido del archivo T80se.vhd puede consultarse en el apéndice, desde de la página 144 hasta la 146
62 Menú de navegación para la localización y el agregado del archivo T80se.vhd

Se elige y se verifica la sintaxis.

Una vez concluido el agregado de todos los archivos y librerías se esta listo para proceder a la sistematización.

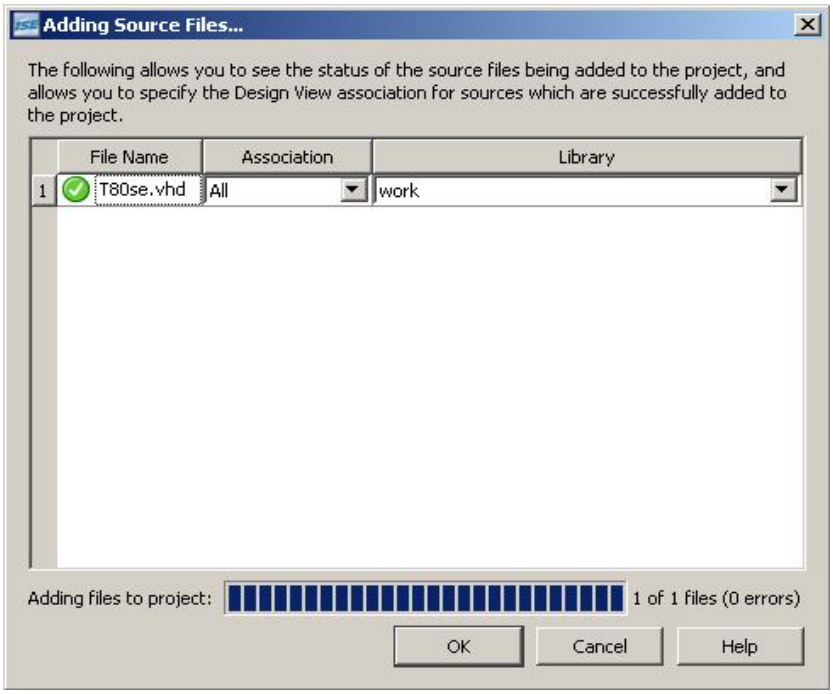



Figura 4.3.2.18[63]

63 Cuadro de verificación del archivo agregado.

### 4.3.3 Creación del esquemático del T80se

La creación del esquemático no reviste de mayores complicaciones, se elige el archivo VHDL dentro del área de trabajo al que se requiera esquematizar , se va después al atea debajo , que es la ventana de procesos y se elige la opción **Crate Schematic Symbol**. La operación tardará aproximadamente un minuto y el programa nos avisará si fue exitoso el proceso. En caso contrario nos avisa en que archivo existió algún fallo y si fue por sintaxis o por la falta de alguna fuente vhd.

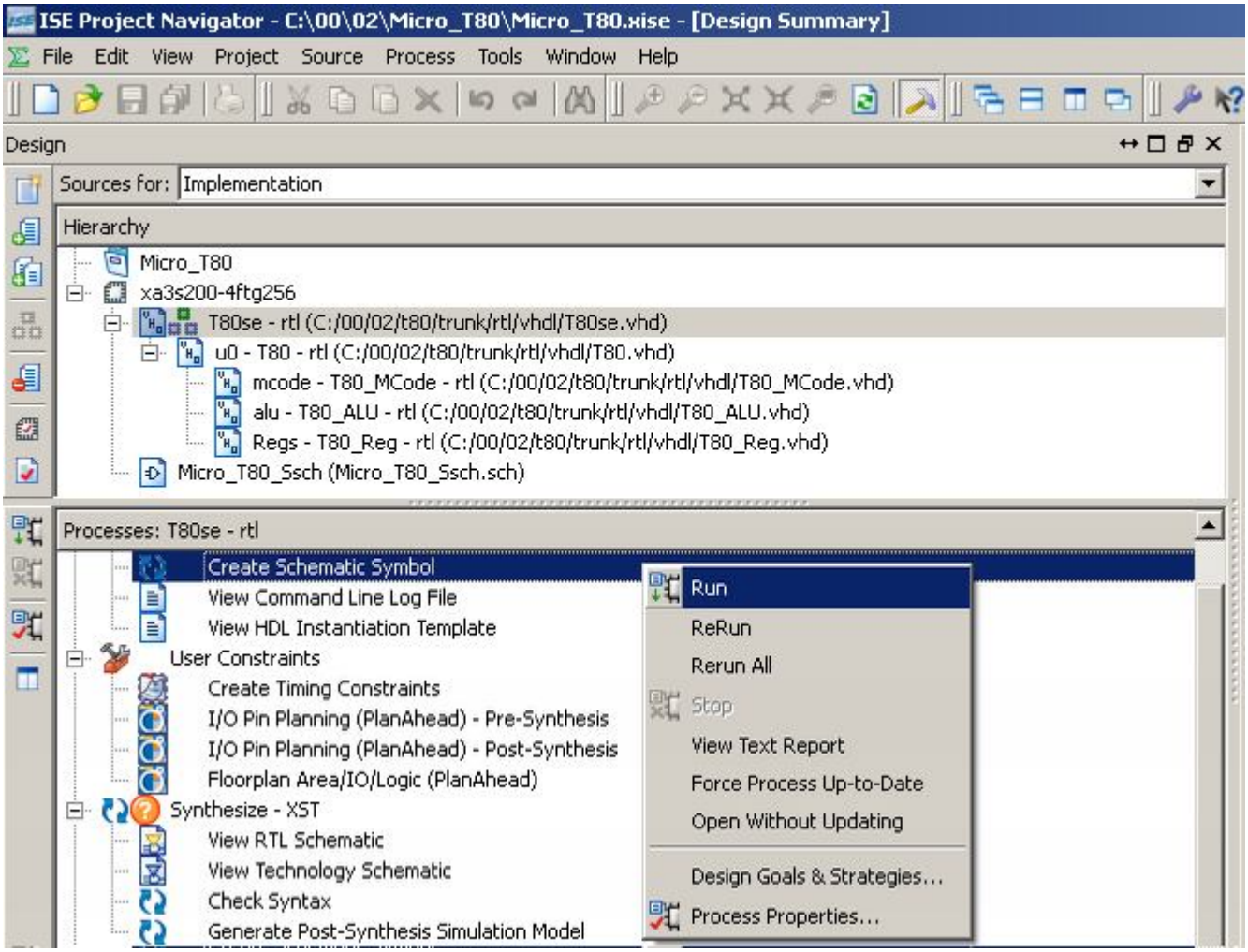



Figura 4.3.3.1[64]

Una ves que el programa informa que el archivo ha sido correctamente esquematizado, el siguiente paso es crear un área de trabajo para poder trabajar con el.

64 Creación del símbolo esquemático del núcleo T80se.

### 4.3.4 Crear un nueva fuente esquemática

Para esto se coloca el cursor sobre el dispositivo XC3s200 en la ventana de proyecto y agregamos una nueva fuente con el botón derecho ( **New Source**)

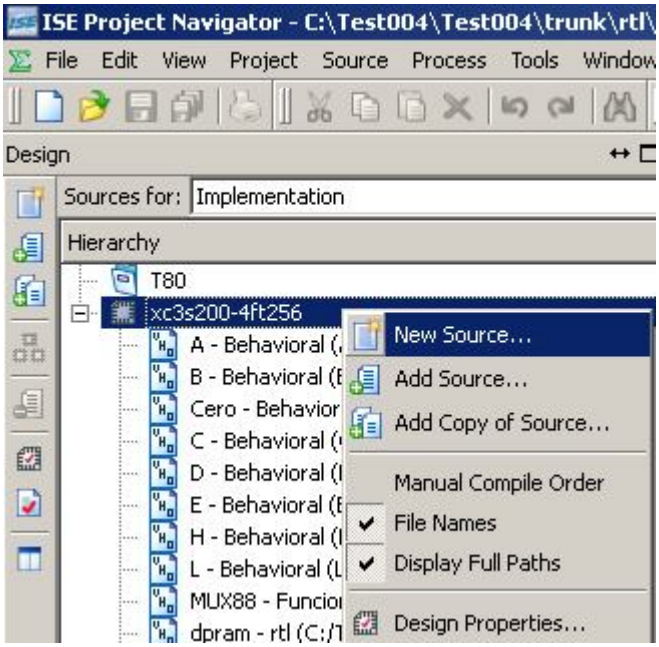



Figura 4.3.4.1

En la ventana siguiente , se selecciona **Schematic** para crear un espacio de trabajo.

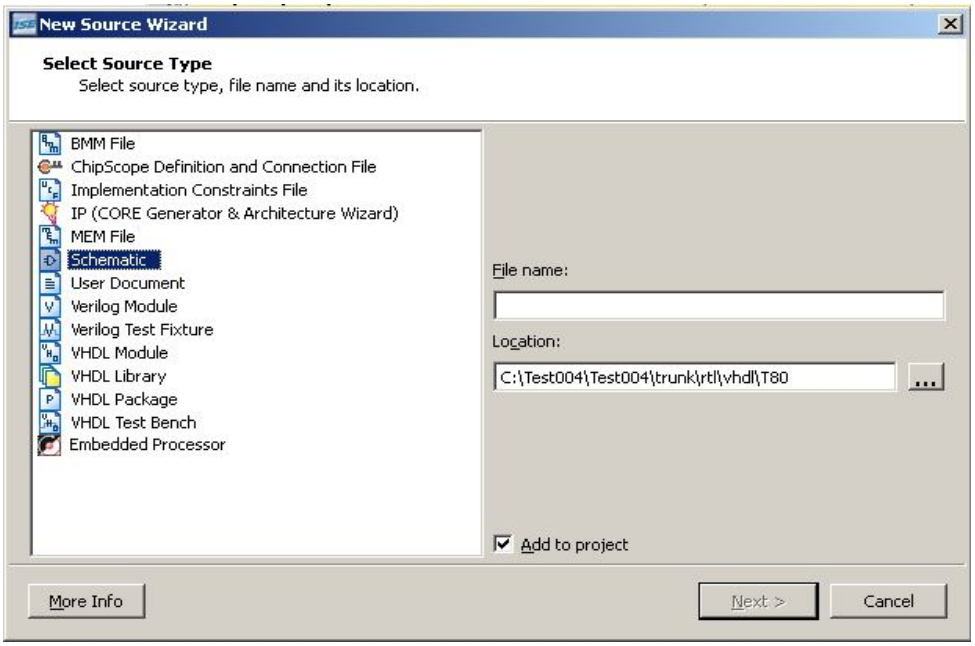



Figura 4.3.4.2[65]

65 Ventana de selección para agregar una nueva fuente al proyecto. En este caso un esquemático.

Se elije un nombre  para el área de esquemático

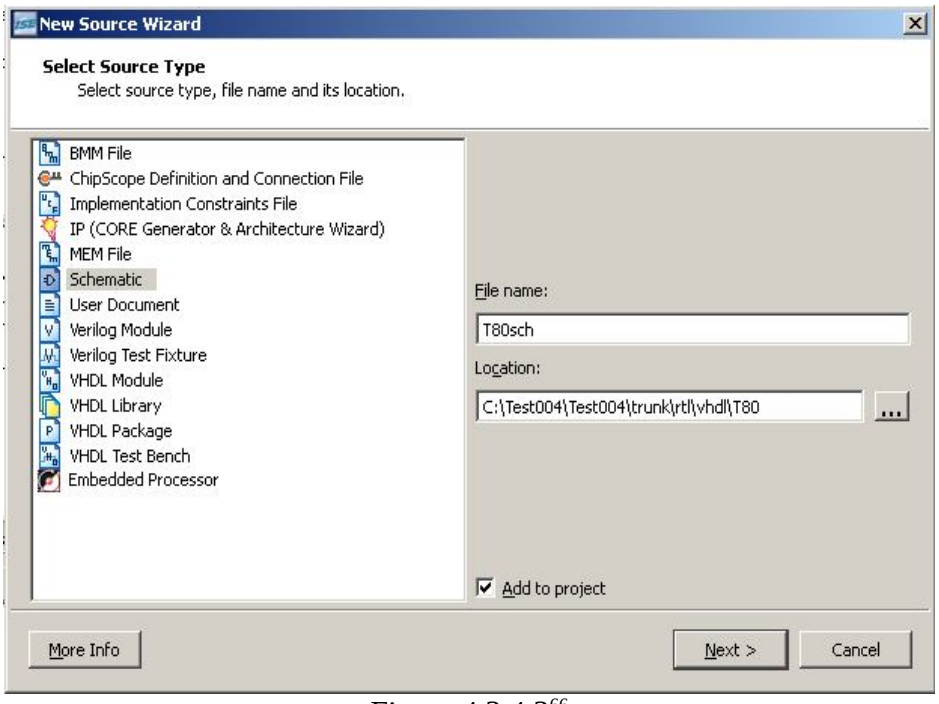



Figura 4.3.4.3[66]

Y  Al finalizar se presenta una ventana con información general.

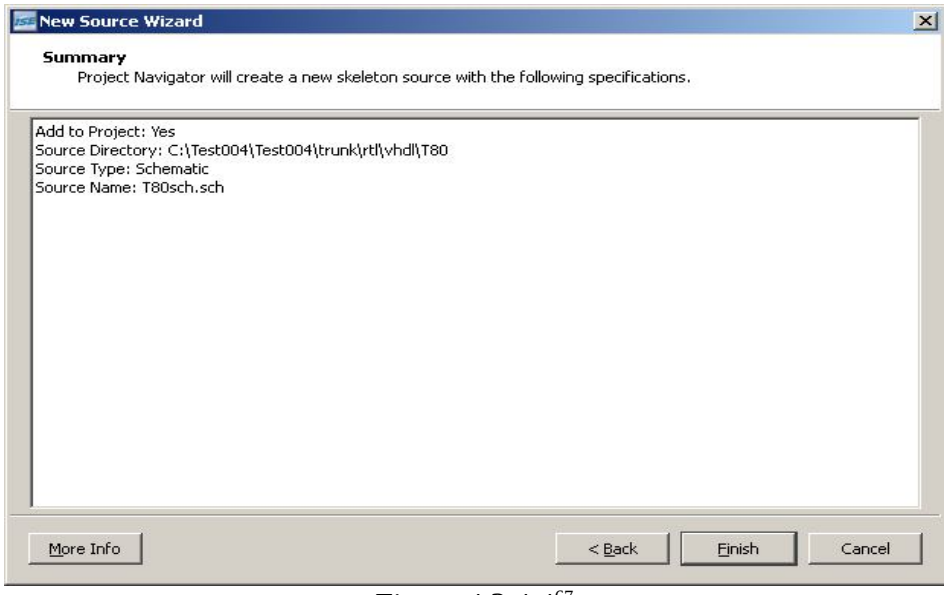



Figura 4.3.4.4[67]

66  Aquí se le asigna un nombre a la fuente esquemática con la cual se trabajará.
67  Ventana de información general.

A continuación se presenta el área de trabajo de **Schematic**. Con características parecidas a el área de trabajo de Electronic WorkBench u Orcad. Aquí se puede comenzar a atrabajar con símbolos, esquemáticos tal y como se hace en cualquier sistema CAD electrónico.

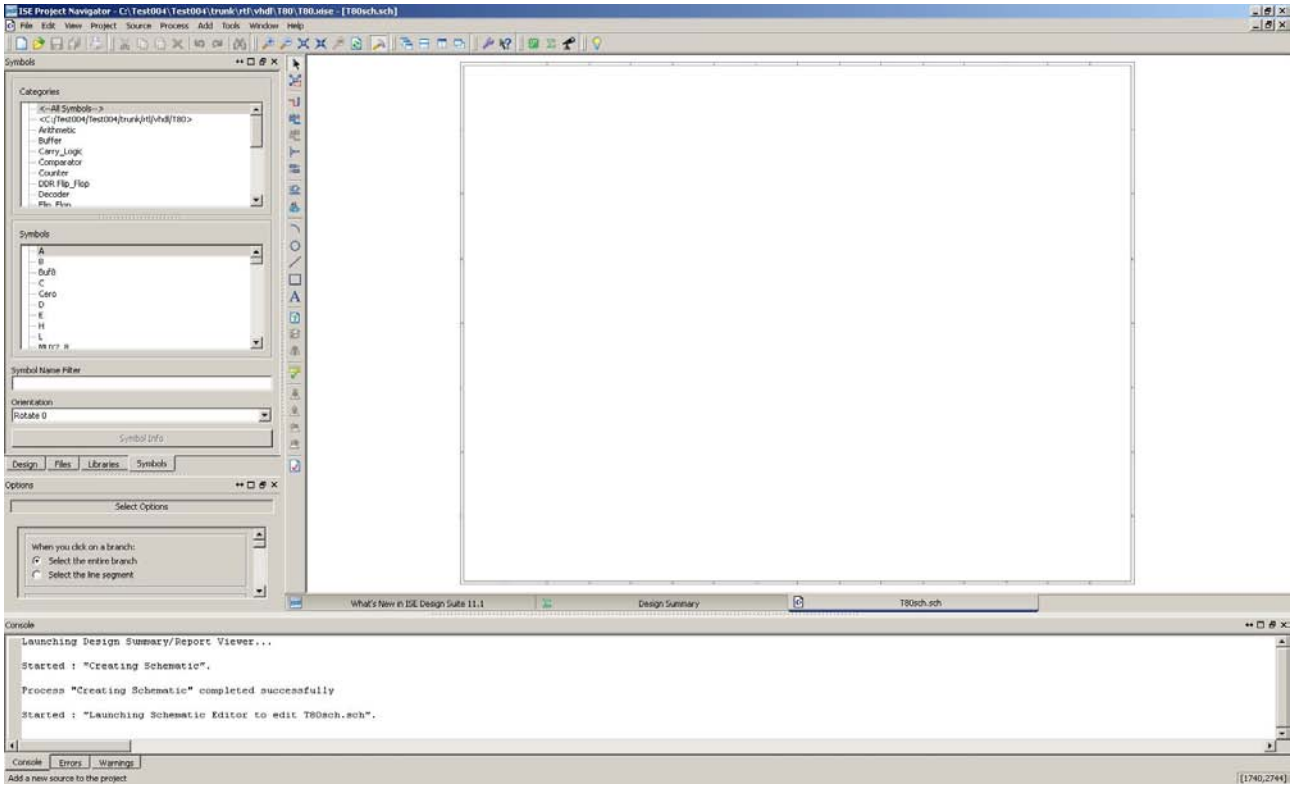



Figura 4.3.4.5[68]

68 Captura de pantalla del área de trabajo **Schematic**

En la Parte izquierda se encuentra en el apartado de símbolos, en ella se puede encontrar una sección que corresponde con el nombre del proyecto , si hemos creado esquemáticos a partir de código VHDL , el símbolo resultante se encuentra ahí. En este caso es el símbolo T80se A continuación se va al área de trabajo y se arrastra el símbolo. Aquí se puede observar el esquemático del T80se.

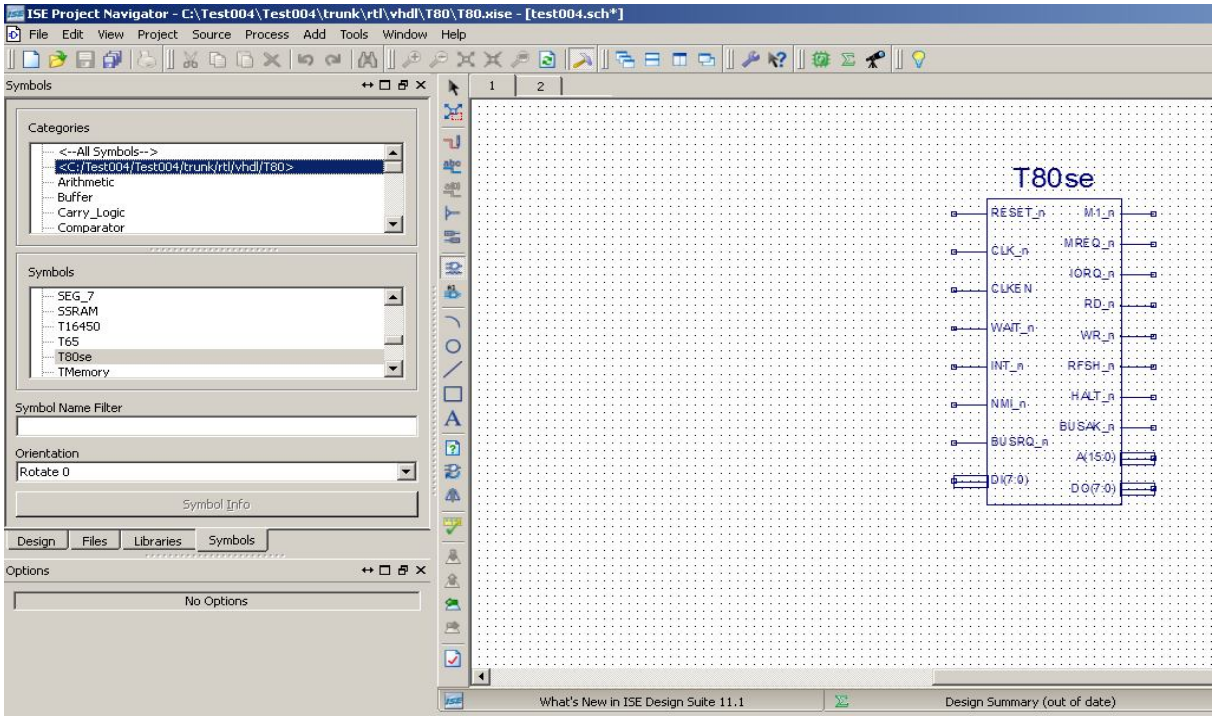



Figura 4.3.4.6[69]

En la siguiente imagen se encuentra una vista mas detallada.

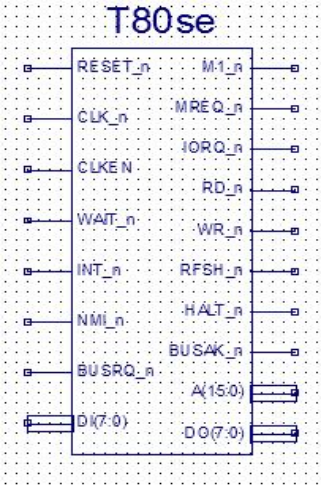



Figura 4.3.4.6[70]

69 Área de trabajo con el símbolo T80se
70 Símbolo T80se esquematizado a partir de los archivos fuente.

Se pueden observar las características generales del T80se.

Las terminales de entrada son:

- RESET_n terminal reset.
- CLK_n en el cual se suministra la señal de reloj .
- CLKEN para activar el funcionamiento del reloj.
- WAIT para dispositivos mas lentos.
- NT_n para solicitud de interrupciones mascarables.
- NMI_n para solicitud de interrupción no mascarable.
- BUSRQ_n para solicitud de uso de bus por un dispositivo externo.
- DI(7:0) Bus de datos de entrada de 8 Bits

Las terminales de salida:

- M1_n indicador de ciclo de máquina.
- MREQ_n requerimiento de memoria activa en nivel bajo
- IOREQ_n requerimiento de E/S activo en nivel bajo.
- RD_n lectura activo en nivel bajo
- WR_n escritura activo en nivel bajo.
- RFSH_n refresco de memoria dinámica
- HALT_n activa en nivel bajo indica que el CPU esta en estado halt
- BUSCAK_n entrega de las terminales del CPU
- A(15:0) líneas de direcciones
- DO(7:0) líneas de datos del bus de salida de 8 bits

Como se puede observar, las terminales listadas son muy similares a las de un microprocesador Z80 convencional con la salvedad de CLKEN y los buses DO y DI. En este caso CLKEN se usa para sincronizar el funcionamiento de el microprocesador dentro del FPGA. Los buses de datos separados se dan, por que dentro del dispositivo , físicamente no puede haber alta impedancia en las terminales esquemáticas dentro del FPGA por ello los buses se separan en entrada y salida. El estado de alta impedancia solo se da en las patillas del FPGA cuando este se conecta a dispositivos externos.

### 4.3.5 Memoria RAM 2k

Dentro de el repositorio de Opencores existen también núcleos de memoria RAM y algunos configurables, se ha elegido un núcleo de memoria RAM elaborado por Daniel Wallner (jesus@opencores.org) y se configura para una capacidad de 2 Kbytes, es decir, del mapa de memoria abarcara los últimos 2Kbytes desde la dirección 0xF800 hasta la dirección 0xFFFF.

En el apéndice puede encontrarse el contenido del archivo escrito en VHDL de este circuito, en la página 148.

### 4.3.6 Esquemático RAM

Para utilizar este archivo en forma esquemática y que sea mas fácil de manejar se procede a esquematizarlo como al archivo T80se , el resultado es el siguiente símbolo en la hoja de trabajo.

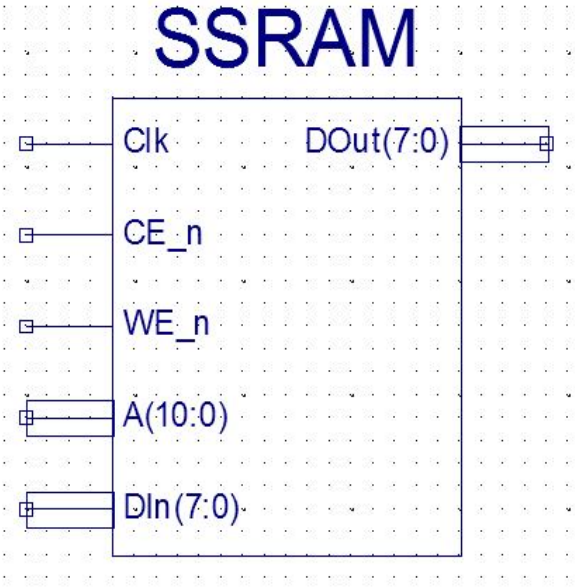



Figura 4.3.5.1[71]

71 Símbolo de la memoria RAM una ves que el archivo vhd se ha esquematizado.

Este tiene las siguientes características

Entradas .

1. Clk Señal de reloj
2. CE Chip Enable activo en bajo
3. WE Write Enalbe activo en bajo
4. A(10:0) bus de entrada de dirección de once bits
5. DI(7:0) bus de entrada de datos de 8 bits

Salidas.

1. Dout(7:0) bus de salida de datos de 8 bits.

Agregada la memoria RAM el siguiente paso es la memoria ROM en donde se guardaran los programas que ejecutara el microprocesador T80. Una ves que el archivo Intel Hex se ha generado ya sea a partir de un ensamblador o un compilador este se convierte en un archivo con extensión vhd con el programa hex2rom.exe .

### 4.3.7 Memoria ROM

Generado el programa y este contenido en un archivo Intel Hex se convierte a una archivo con extensión VHDL con el siguiente comando haciendo uso del programa hex2rom

[Tet@localhost pasmo-0.5.3]$wine hex2rom.exe 111.hex Bloque_Memoria 16l8

en donde 111.hex es el archivo en hexadecimal generado por el ensamblador, Bloque_Memoria es el nombre de la arquitectura que contendrá el listado de memoria ROM y 16l8 son las opciones de volcado con las que trabaja el programa. En este caso un bus de direcciones de16 bits, un bus de datos de 8 bits y para una arquitectura little-endian.

Descripción del programa.

- Carga el valor 0xbc en el acumulador y después se coloca en el puerto 0x00.

Archivo en ensamblador

```
        org     0x0000
inicio:
    ld          a,0xbc
    out         (0x00),a
    end
```

Contenido del archivo IntelHex generado por le programa pasmo.exe

```
:040000003EBCD3002F
:00000001FF
```

Resultado de la aplicación del programa hex2rom al contenido del archivo IntelHex anterior

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity Bloque_Memoria is
	port(
		A	: in std_logic_vector(15 downto 0);
		D	: out std_logic_vector(7 downto 0)
	);
end Bloque_Memoria;
architecture rtl of Bloque_Memoria is
begin
	process (A)
	begin
		case to_integer(unsigned(A)) is
		when 000000 => D <= "00111110"; -- 0x0000
		when 000001 => D <= "10111100"; -- 0x0001
		when 000002 => D <= "11010011"; -- 0x0002
		when 000003 => D <= "00000000"; -- 0x0003
		when others => D <= "--------";
		end case;
	end process;
end;
```

### 4.3.8 Esquemático de la Memoria ROM

Obtenido el archivo VHDL de la memoria ROM se adjunta al proyecto y se genera el esquemático del mismo, obteniendo el resultado siguiente en el área de trabajo **Schematic**.

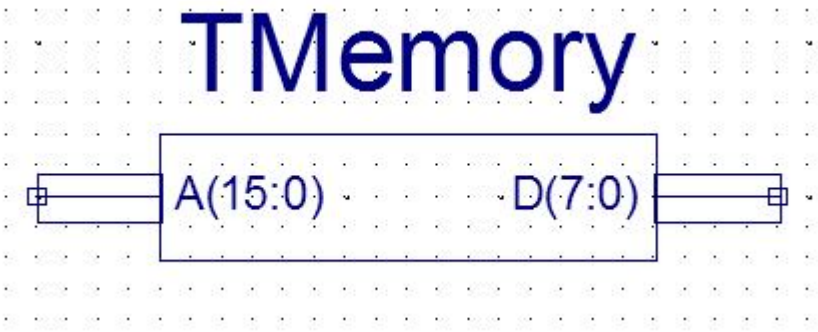



Figura 4.3.8.1[72]

72 Símbolo de la memoria ROM una ves que el archivo vhd se ha esquematizado.

4.3.9 Conexiones básicas para el arranque.

Agregando las conexiones necesarias para arrancar el circuito, en primer instancia de conecta a Vcc las patas CLKEN, WAIT_n, INT_n, NMI_n y BUSRQ_n.

| | |
|---|---|
| CLKEN | en alto para permitir la activación del reloj. |
| WAIT_n | en alto ya que el circuito no se conectará a dispositivos lentos. |
| INT_n | en alto, las interrupciones no se activarán para las pruebas básicas. |
| NMI_n | en alto, las interrupciones no mascarables estarían desactivadas para las pruebas básicas. |
| BUSRQ_n | en alto, la solicitud de uso de bus se desactivará para las pruebas. |

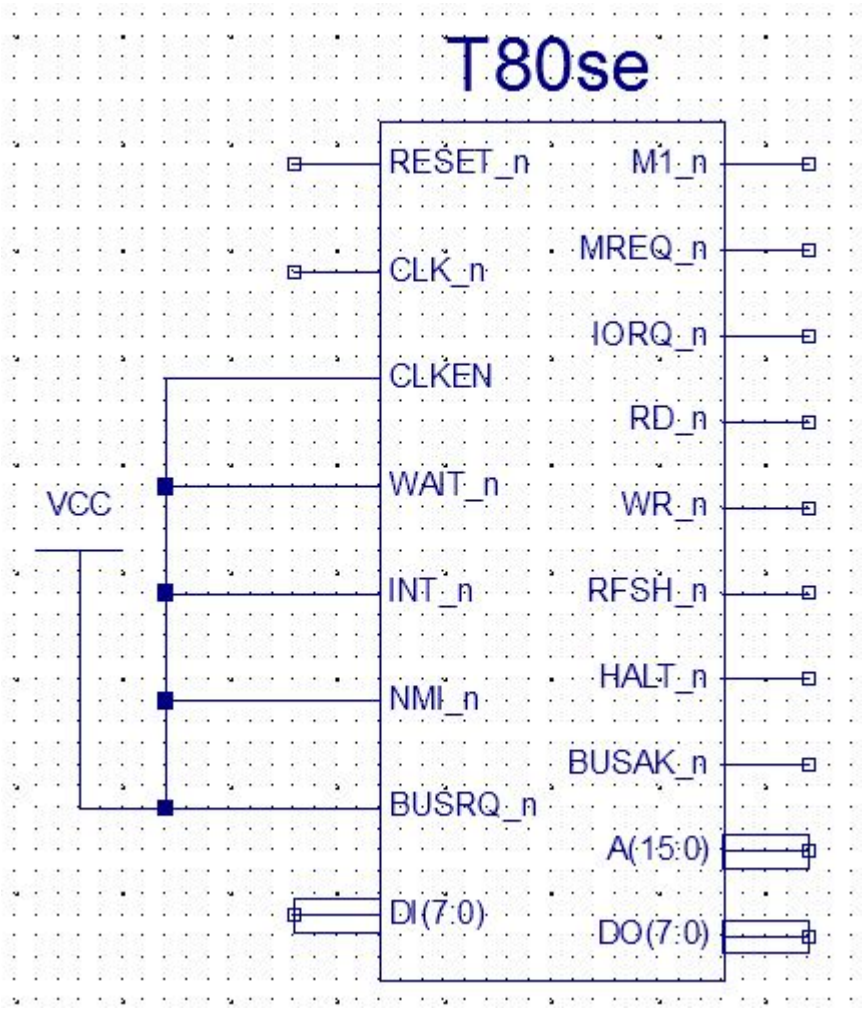



Figura 4.3.9.1[73]

73 Conexiones básicas para el funcionamiento del microprocesador.

4.3.10 Asignar nombres a las terminales del esquemático

Para tener control sobre a que pines serán asignadas las patillas del dispositivo FPGA es necesario que tengan un nombre asignado. Para ello usamos el **Add I|O Maker** . En la barra de menu **Add -> I/O Maker** .

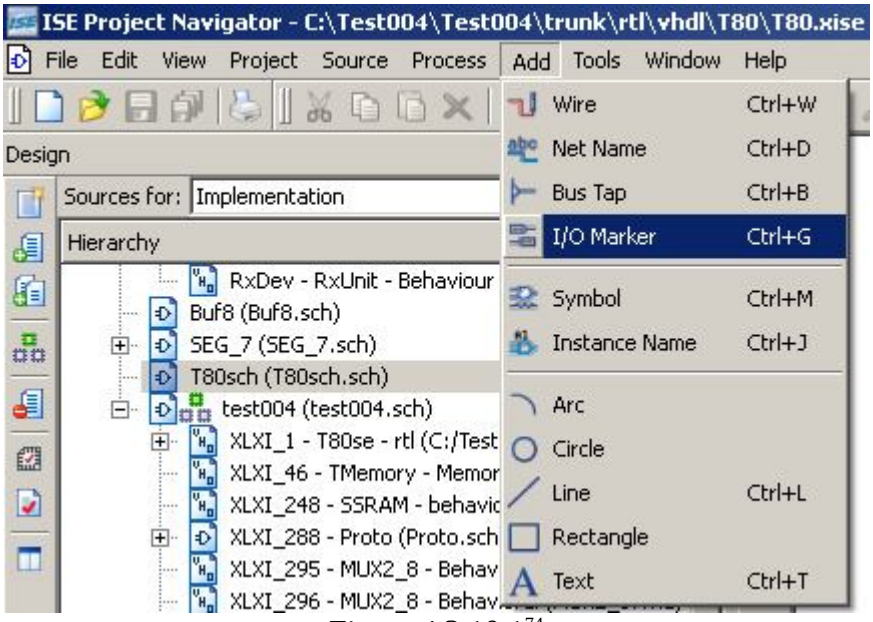


Figura 4.3.10.1[74]

Colocando la etiqueta en la terminal esquemática el nombre que se le asigna es de forma automática elegido por el sistema, con número consecutivo.

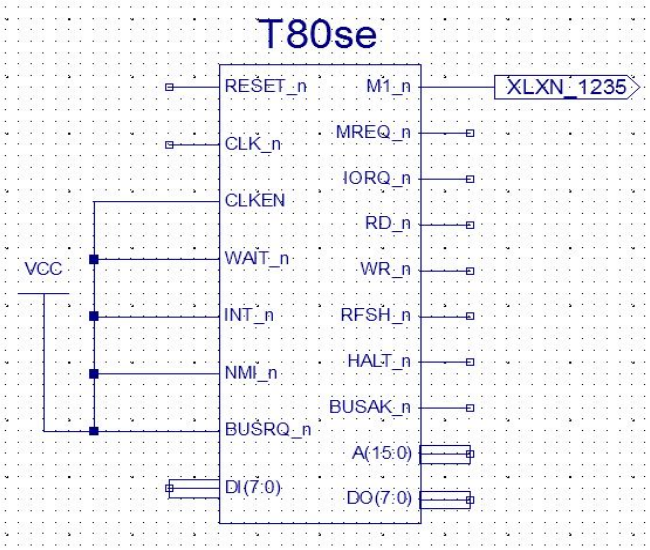


Figura 4.3.10.2[75]

74 Selección de la herramienta para nombrar las terminales.
75 Muestra de el nombre de la terminal con la asignación automática del programa.

Una vez que esto se ha echo se procede a modificar el nombre del mismo para localizar mejor la terminal una vez que se asignen los pines del FPGA. Se posiciona el puntero del ratón sobre la patilla a la que se le daría nombre y se da click, a continuación en el menú desplegable resultante de esta acción se selecciona **Rename Port .**

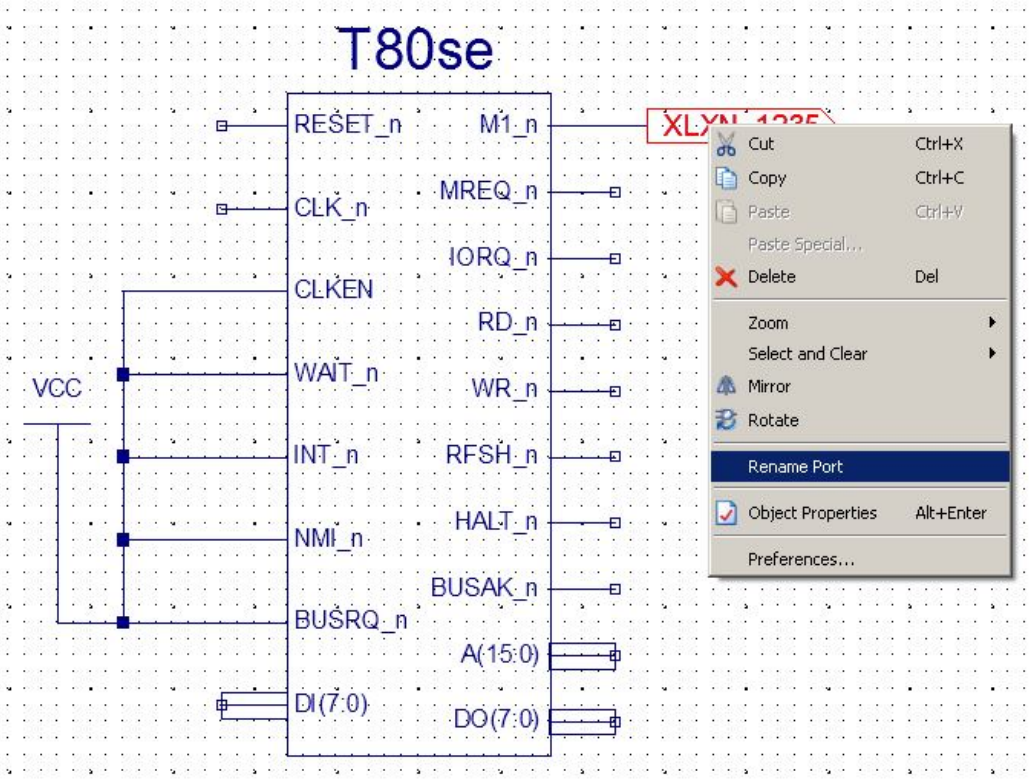



Figura 4.3.10.3[76]

El menú siguiente nos indica que coloquemos el nombre que se dará a esta patilla, este nombre es el que aparecerá en la aplicación que se usará para asignar las salidas y entradas a los pines del dispositivo FPGA Spartan 3. Es preferible que el nombre nos de una descripción de la patilla o el nombre del mismo en el esquemático

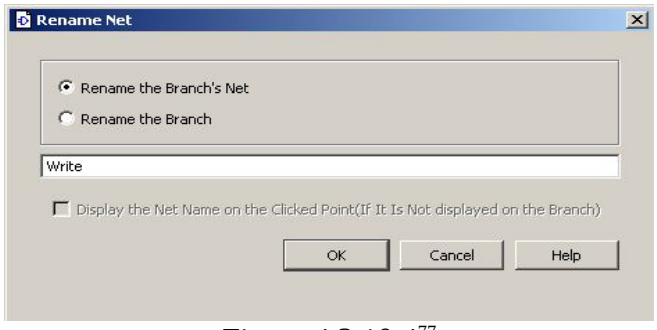



Figura 4.3.10.4[77]

76 Menú desplegable, se selecciona la opción de renombrar una terminal.
77 Cuadro de diálogo para asignar un nombre a la terminal en que se trabaja.

Se nombran las entradas y salidas principales, a continuación se observa el proceso con la mitad de las etiquetas, las terminales que no se usarán para conectarse al exterior y solo se usarán para el funcionamiento del circuito dentro del FPGA no necesitan ser nombradas.

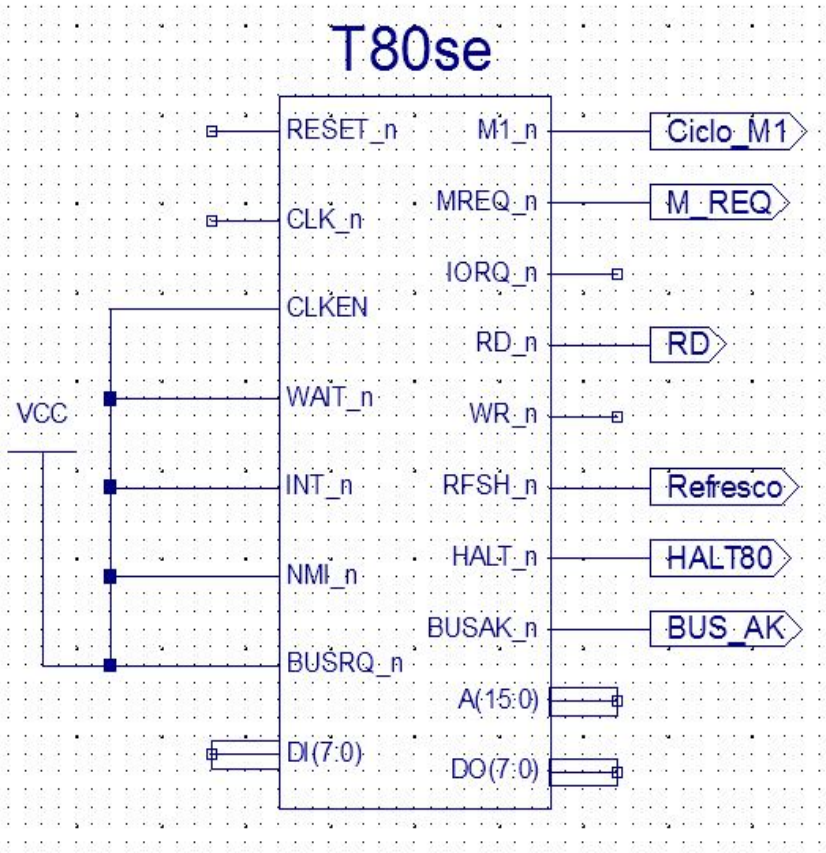



Figura 4.3.10.5[78]

78 Conexiones sobre el símbolo T80se , aun no se han agregado todas las conexiones ni nombrado todas las terminales.

Los dispositivos auxiliares así como codificadores de direcciones y circuitería necesaria se puede diseñar conforme a los requerimientos de cada sistema en particular. Los circuitos esquemáticos tienen un amplia gama de dispositivos integrados para su uso en el ***Schematic*** .

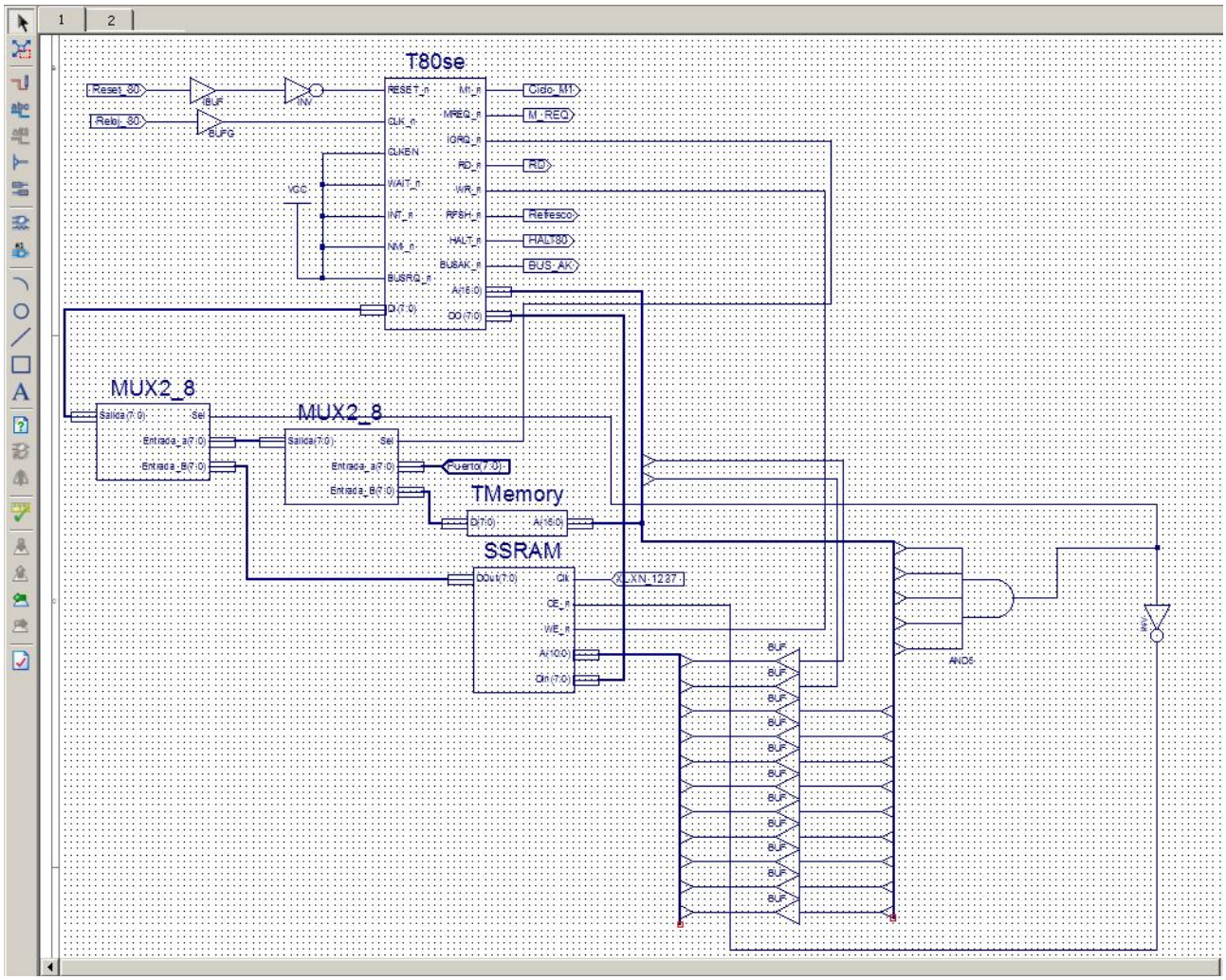



Figura 4.3.10.6[79]

79 Captura de pantalla del área de trabajo de esquemáticos. El núcleo T80 posee ya todas las conexiones necesarias.

### 4.3.11 Asignación de patillas del FPGA

Teniendo ya el circuito completo y las entradas y salidas etiquetadas se procede a asignar los pines del circuito con su correspondiente en el dispositivo FPGA. Un ejemplo de esto es el reloj, en la tarjeta de desarrollo se tiene en el pin M13 donde llega una señal de reloj de 50 Mhz y es ahí a donde el pin CLK del esquemático se conectará. Para iniciar la aplicación que lleva a cabo la tarea de asignación se elige de la barra de herramientas. ***Tools -> Floorplan Area/IO/Logic (PlanAhead)***

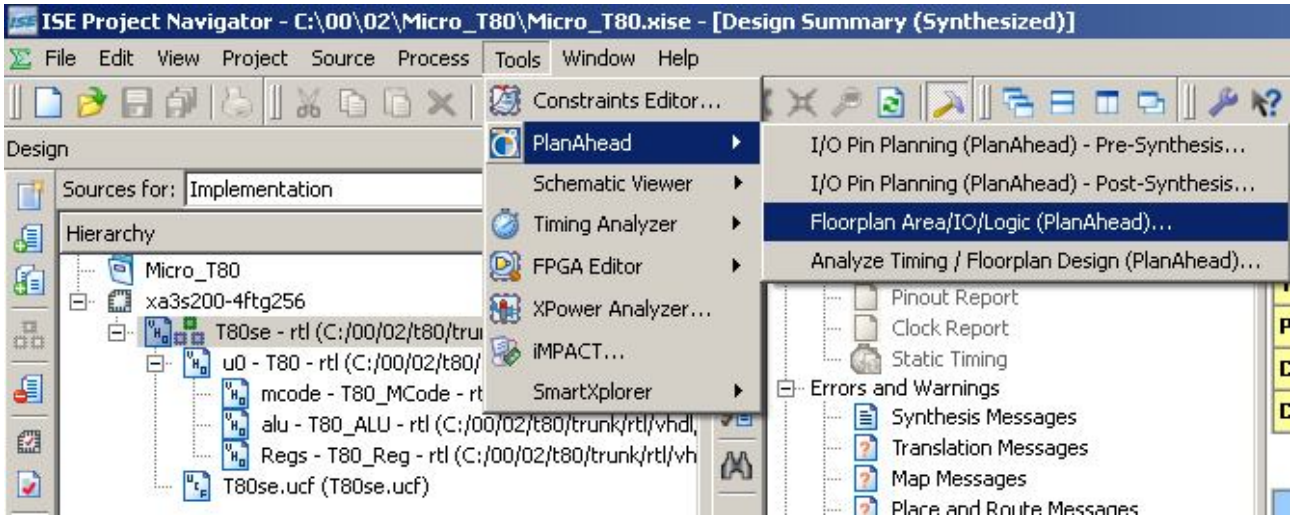



Figura 4.3.10.7[80]

Si es la primera vez que se ejecuta el programa **Plan Ahead** aparecerá una advertencia diciendo que se generará el archivo .ucf (Implementation Constraint File) y se acepta para que el programa lo genere

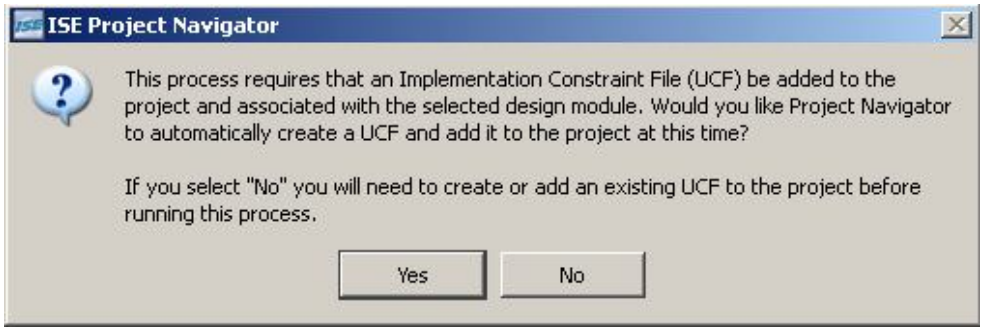



Figura 4.3.10.8[81]

80 Ejecución del programa para asignar las terminales físicas del FPGA Xilinx a las terminales del esquemático.
81 Aviso acerca de la implementación del archivo necesario UCF.

La siguiente pantalla es en donde se muestra físicamente la localización de los pines del FPGA así como sus conexiones físicas y así mismo las conexiones internas.

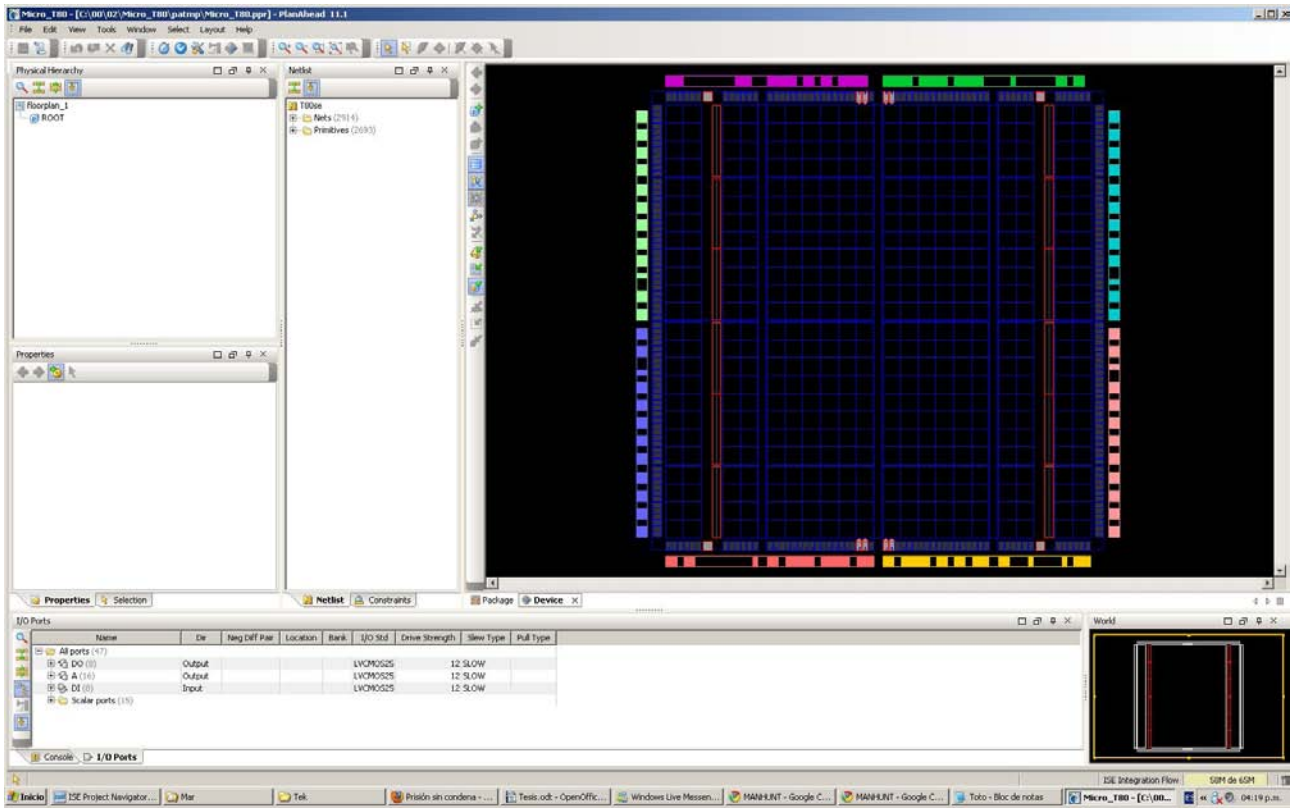


Figura 4.3.10.9[82]

En la parte inferior del área de trabajo se encuentran agrupados los nombres que previamente hemos asignado a los pines así como también a los buses. Aquí se observan las etiquetas que hemos colocado en el apartado **LOCATION** es donde colocaremos el nombre del pin del FPGA al cual se quiere que la etiqueta quede conectada. En el apartado Dir se observa la dirección tanto si es de entrada , salida o ambos.

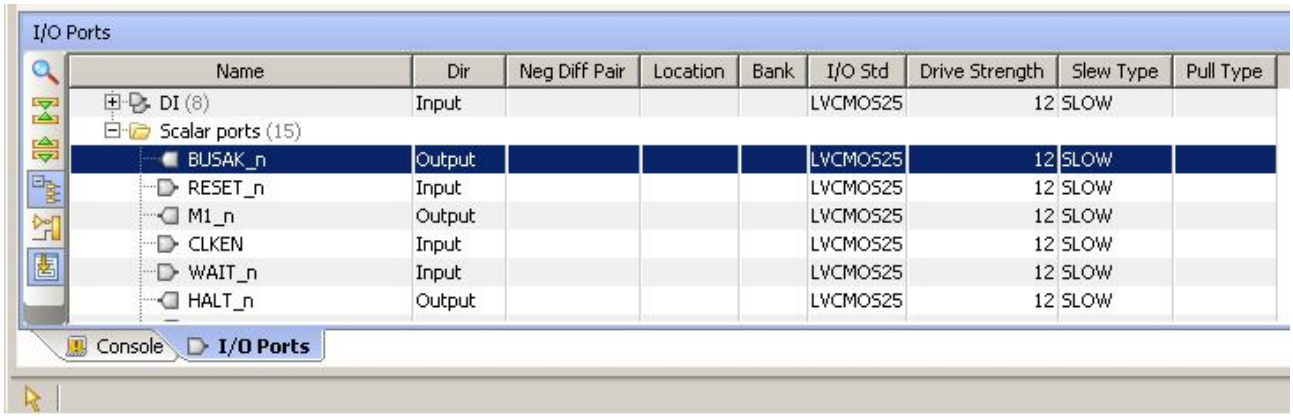



Figura 4.3.10.10[83]

82 Área de trabajo de Xilix Plan Ahead para la asignación de terminales.
83 Vista detallada de el área de puertos.

Para asignar un pin especifico del FPGA vamos a la parte inferior del área de trabajo seleccionamos el apartado ya sea de bus o escalar al cual pertenece la etiqueta que deseamos conectar con el exterior

En este caso conectaremos la del esquemático T80 nombrado BUSAK_n al pin del FPGA llamado K2. Seleccionado la etiqueta nos colocamos en la ventada **I/O Port Properties**.


Figura 4.3.10.10[84]

84 Cuadro de diálogo para la asignación física de los pines.

4.3.12 Síntesis de un proyecto.

Para sintetizar el proyecto y generar el archivo .bit que se necesita para programar el FPGA, se coloca en el cuadro de procesos se coloca en el apartado **Generate Programming File**. Con el botón derecho se abre el menú desplegable y se elige la opción **Run**

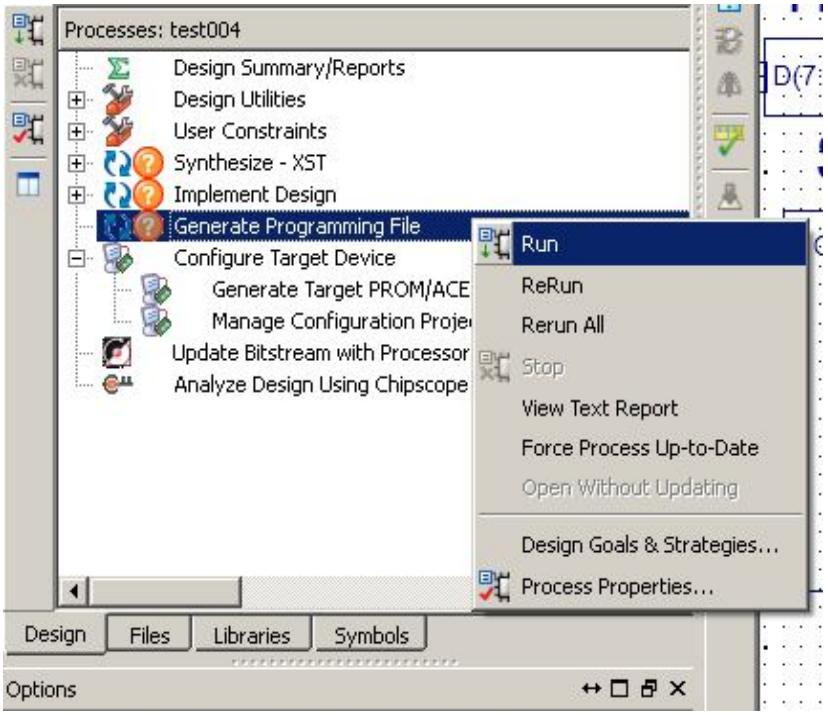



Figura 4.3.12.1[85]

Dependiendo de la complejidad del diseño y el poder de computo, la síntesis tardará de 6 a 10 minutos.

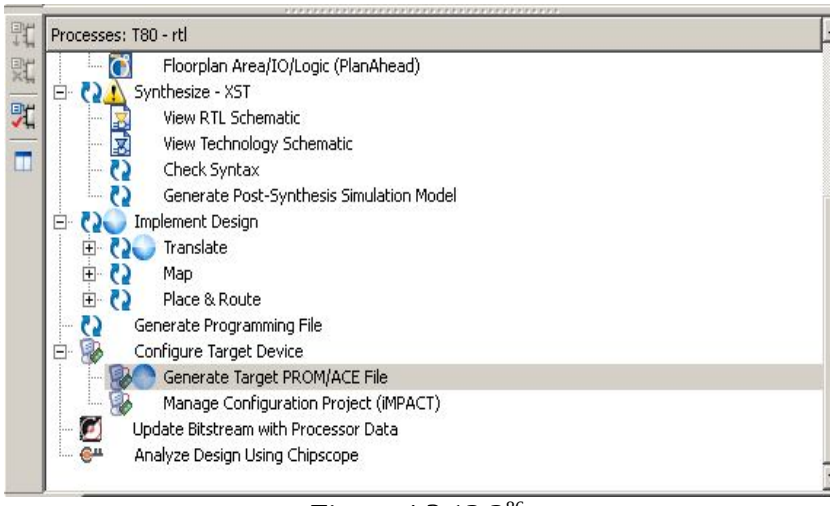



Figura 4.3.12.2[86]

85 Área de procesos. Aquí se inicia la generación del archivo .bit.
86 Procesos llevándose a cabo. Los círculos azules indican el proceso que se esta ejecutando.

Una vez concluida exitosamente la síntesis, el siguiente cuadro de aviso aparece, indicando con ello que se ha concluido el proceso

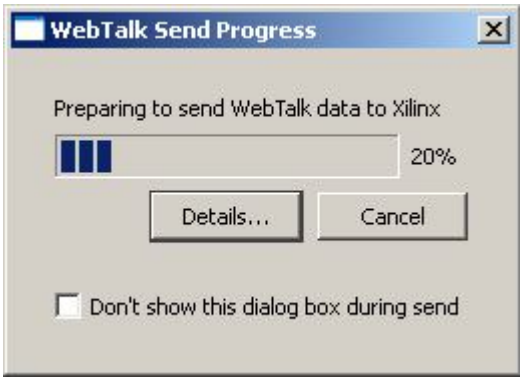



Figura 4.3.12.3[87]

Durante el desarrollo de todo el proceso puede modificarse el proyecto, los archivos vhdl , verilog, o el esquemático. Cuando se tiene que hacer una prueba sobre el dispositivo físico es cuando se genera el archivo de programación Si se ha modificado el proyecto y no se ha sintetizado un nuevo archivo .bit el sistema lo haría saber colocando un símbolo de interrogación en los procesos.

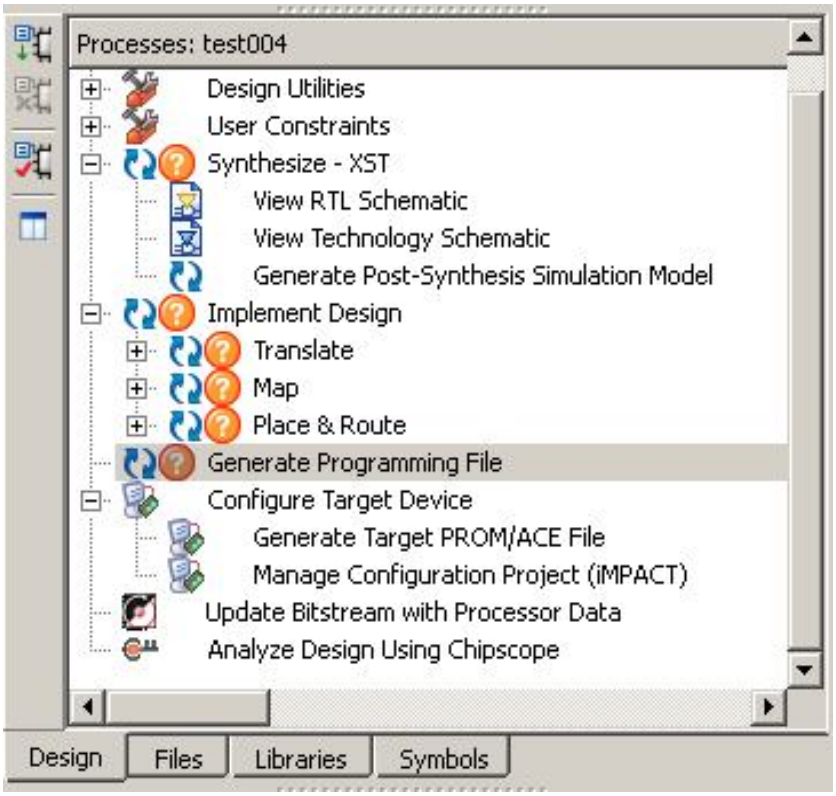



Figura 4.3.12.4[88]

87 Mensaje indicando que el programa enviara datos a la compañía Xilinx acerca de la síntesis.
88 Iconos (en forma de símbolo de interrogación) del área de procesos cuando alguna fuente ha cambiado su contenido.

### 4.3.13 Programación del FPGA

Una vez concluido el sintetizado se ejecuta el programa **iMPACT** que se utiliza para la programación de la tarjeta de desarrollo

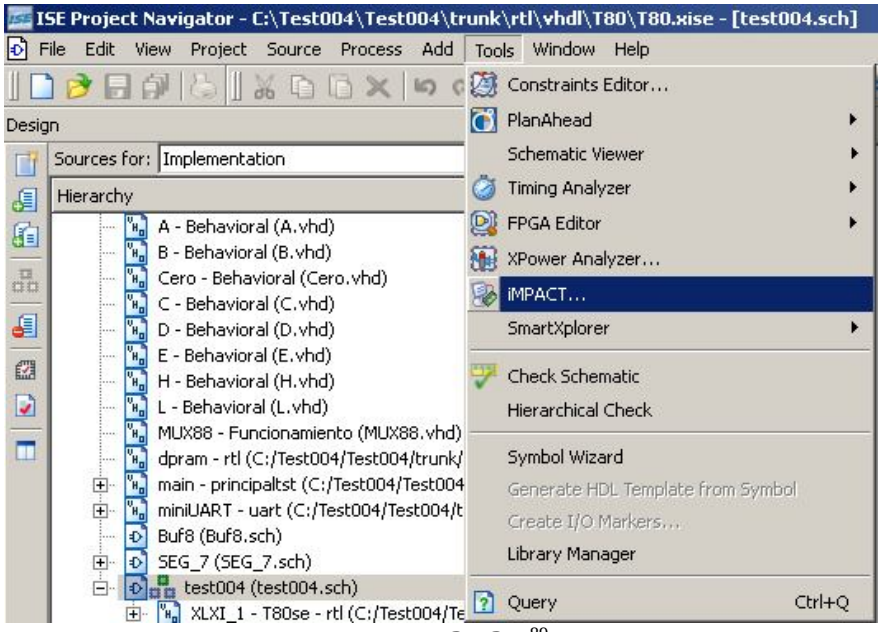
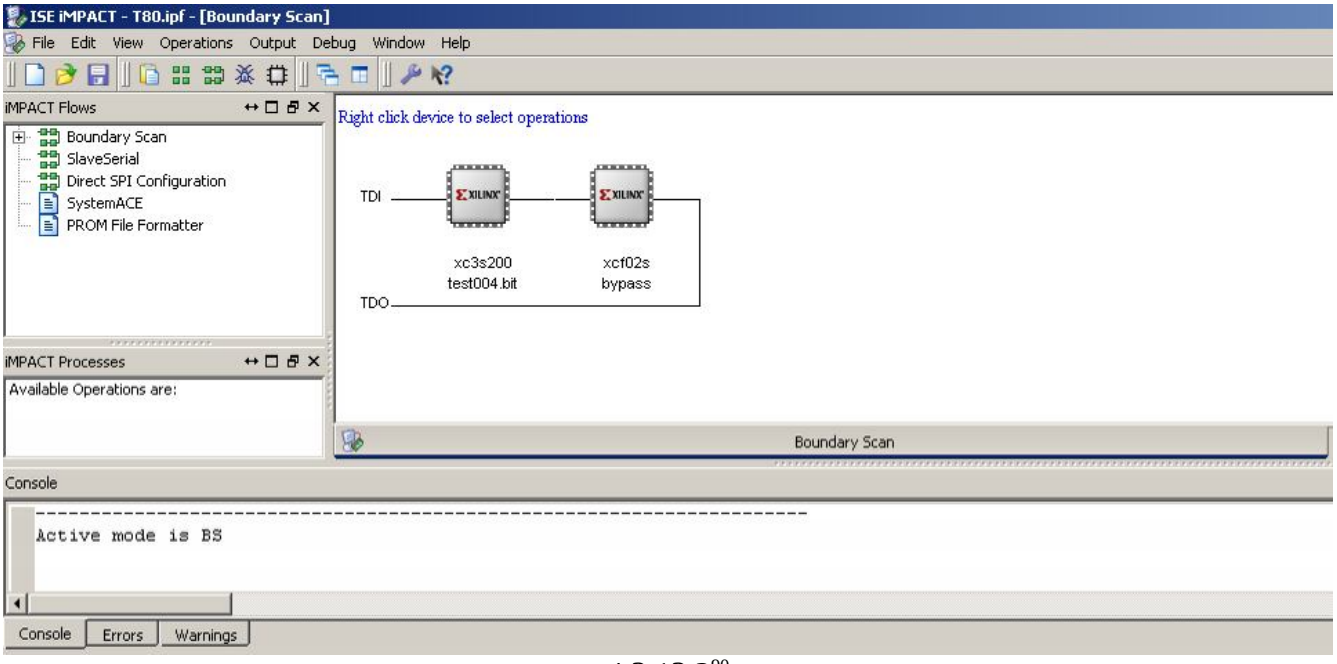



Figura 4.3.13.1[89]




4.3.13.2[90]

89 Ejecución del programa iMPACT para descargar el archivo .bit al FPGA.
90 Pantalla inicial de iMPACT . En esta se observa los dispositivos detectados.

Si la tarjeta esta energizada y el cable de programación JTAG esta conectado a la misma y a la computadora, el programa detecta los dispositivos, en este caso un FPGA Xilinx xc3s200 y una memoria Xilinx xcf02s

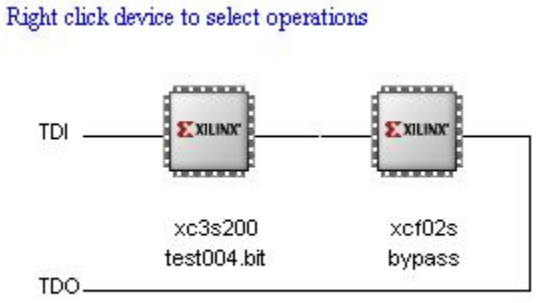



Figura 4.3.13.3[91]

EL FPGA pierde su programación una ves que la energía eléctrica se desconecta , en este caso se puede programar la memoria EEPROM serial xcf02s presente en la tarjeta de desarrollo para mantener el programa y una ves que se energiza el circuito el FPGA lee la programación de esta memoria no volátil.

Se procede a programar el dispositivo, se posiciona el puntero del ratón sobre el mismo y se abre un menú desplegable con el botón derecho

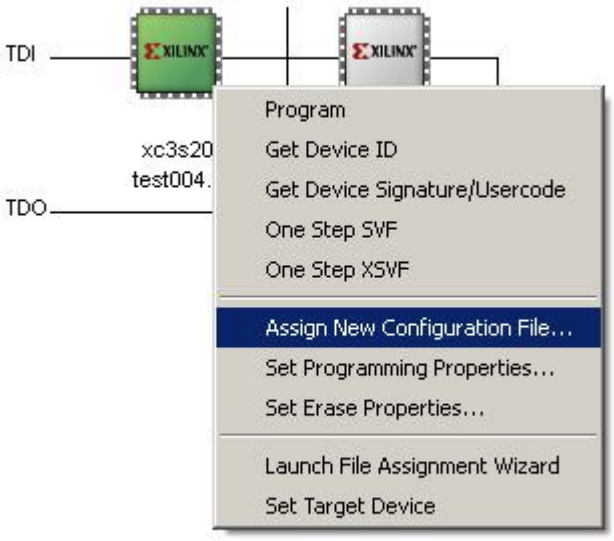



Figura 4.3.13.4[92]

91 Vista detallada de los dispositivos detectados en iMPACT. El FPGA es el dispositivo xc3s200

92 Menu desplegable del dispositivo. Se usara la opción **Assign New Configuration File** para asignar un archivo a descargar.

Desde este menú se selecciona **Assign New Configuration File** para elegir el archivo ( extensión .bit) que se cargara.

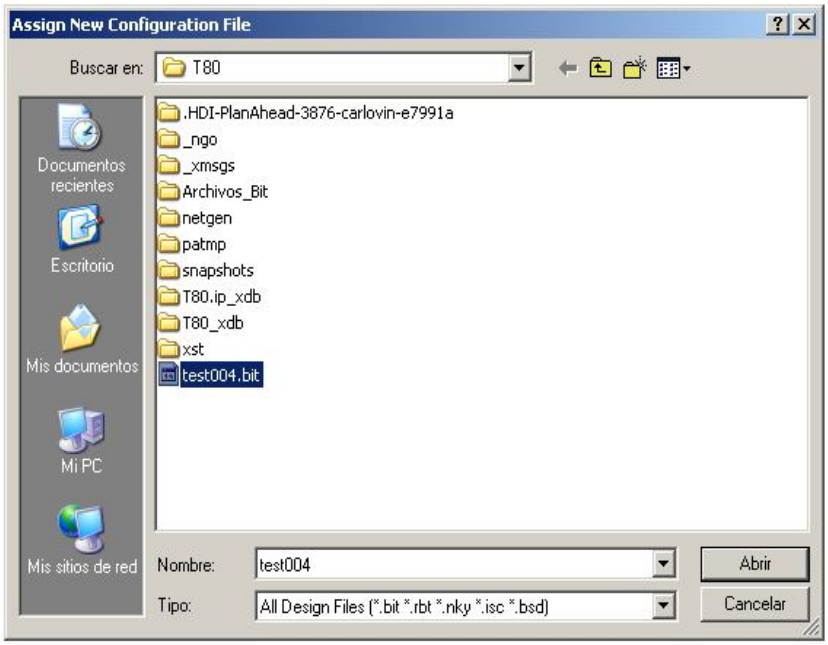



Figura 4.3.13.5[93]

Una ves que se ha elegido el archivo con el cual se programará el FPGA , se procede a la programación en si. Esto se hace colocando el puntero del ratón sobre el dispositivo y desplegando el menú con el botón derecho se elige **Program**

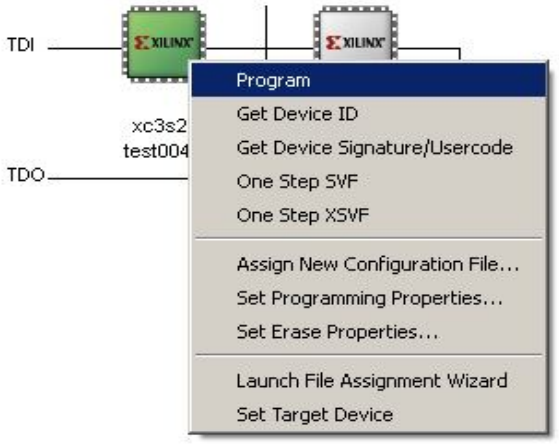



Figura 4.3.13.6[94]

93 Menú de navegación. Se busca y se selecciona el archivo .bit que se descargara al dispositivo.
94 Menú desplegable del dispositivo. La programación se lleva acabo eligiendo el apartado **Program**

Previo a la operación de programado se muestra el cuadro de diálogo indicándonos los dispositivos que se programaran

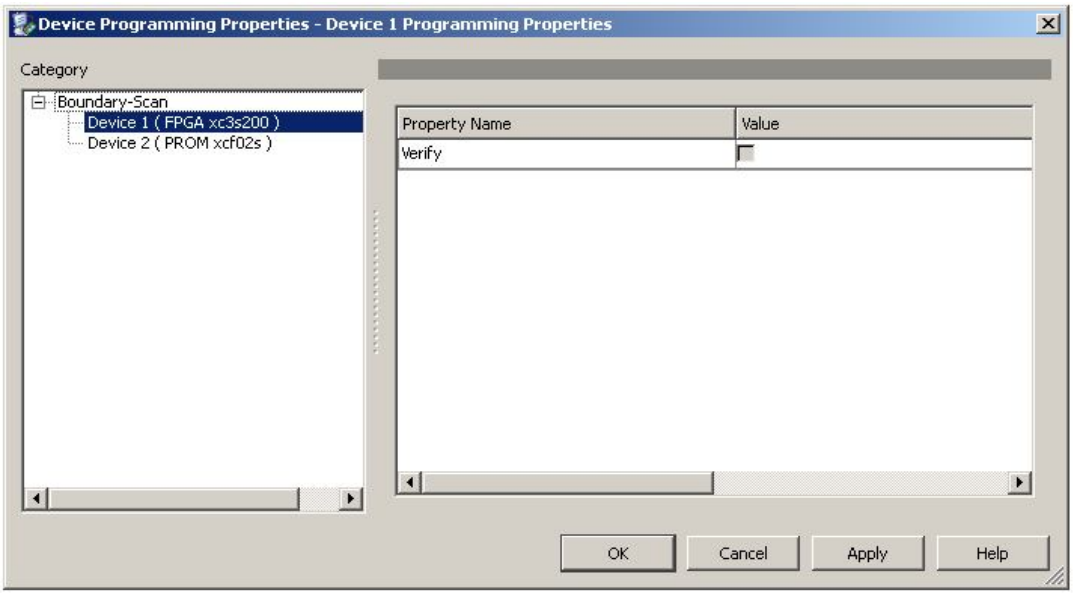



Figura 4.3.13.6[95]

Una ves que se esta de acuerdo se acepta y procede la programación La aplicación lanza un aviso acerca de la operación que se esta realizando

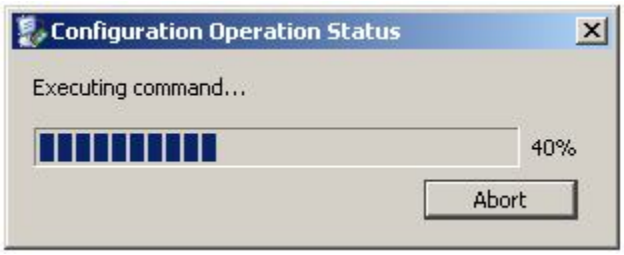



Figura 4.3.13.7[96]

La operación exitosa se anuncia al término.

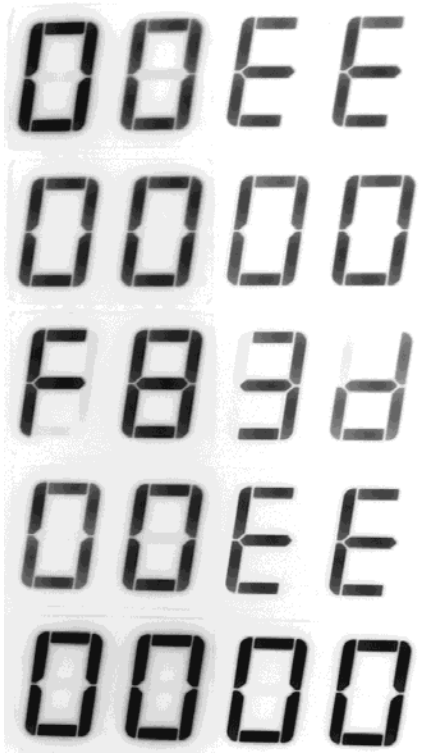

Figura 4.3.13.8[97]

95 Menú previo donde se informa las propiedades de los dispositivos a programar.
96 Cuadro de progreso de la programación del FPGA.
97 Aviso de operación de programación llevada a cabo con éxito.

# 5 Pruebas de programación

A continuación se harán pruebas de programación sobre el micro ya sintetizado y programado dentro del FPGA Xilinx Spartan 3E. Para poder visualizar mejor todos y cada uno de los registros, se creo circuitería externa para dar cabida a los registros A, B, C ,D ,E , H y L en un arreglo de display de 7 segmentos. La distribución de los registros a visualizarse puede apreciarse de la siguiente forma:

| | | | |
|---|---|---|---|
| $B_H$ | $B_L$ | $C_H$ | $C_L$ |
| $D_H$ | $D_L$ | $E_H$ | $E_L$ |
| $H_H$ | $H_L$ | $L_H$ | $L_L$ |
| $0_H$ | $0_L$ | $A_H$ | $A_L$ |
| $1_H$ | $1_L$ | $2_H$ | $2_L$ |




En donde $X_H$ es la parte alta del registro y $X_L$ la parte baja, los números se mostraran en formato de 7 segmentos. Los display 0, 1 y 2 son registros especiales para colocar datos en caso de se necesite mostrar resultados extras. En este caso en particular se observa que los contenidos de los registros del microprocesador son:

| Registro | Valor Hexadecimal contenido |
|---|---|
| **A** | **0xEE** |
| **B** | **0x00** |
| **C** | **0xEE** |
| **D** | **0x00** |
| **E** | **0x00** |
| **H** | **0xF8** |
| **L** | **0x3D** |

El código ensamblador del programa es el siguiente:

include "/home/Tet/pasmo/cabecera_00.asm" [98]

```
      ld    hl,0xf83d
      ld    a,0xee
      ld    (hl),a
      ld    c,(hl)
      call  seg
      halt
```

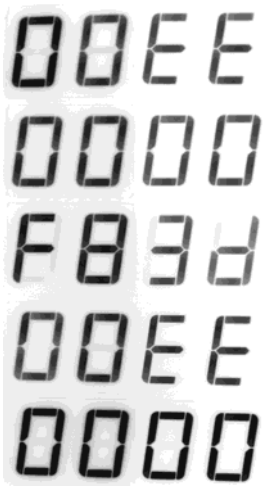
En este caso particular la instrucción que se requiere comprobar es

```
      ld    c,(hl)
```

Las demás instrucciones son necesarias para cargar datos y moverlos entre los registros para comprobar que la instrucción cumple su propósito, además de las instrucciones contenidas en el archivo llamado cabecera_00.asm que envían el contenido de los registros a los display de 7 segmentos. Una vez ensamblado se puede volcar el contenido del archivo hexadecimal, el cual en este caso es:

```
:1000000021F0FFF93E0006000E0016001E0026003B
:100010002E00CD1800C33500D30778D30079D30163
:100020007AD3027BD3037CD3047DD3053E00D30671
:100030003E00D308C900000000000000000000000DE
:0B004000213DF83EEE774ECD18007613
:00000001FF
```

El segmento en negritas corresponden a los códigos de máquina de las instrucciones, que también se presentan en negritas dentro del listado del archivo ensamblador. El código, la fotografía de la prueba, el listado hexadecimal y del archivo ensamblador así como la instrucción que se esta probando quedarían integrados en la tabla de la siguiente forma.




*Fotografía de la prueba

```
include "/home/Tet/pasmo/cabecera_00.asm"
      ld    hl,0xf83d
      ld    a,0xee
      ld    (hl),a
      ld    c,(hl)
      call  seg
      halt              *Listado del archivo en ensamblador
```

```
      ld    c,(hl)      *Instrucción a probar
```

```
:1000000021F0FFF93E0006000E0016001E0026003B
:100010002E00CD1800C33500D30778D30079D30163
:100020007AD3027BD3037CD3047DD3053E00D30671
:100030003E00D308C900000000000000000000000DE
:0B004000213DF83EEE774ECD18007613
:00000001FF           *Contenido del archivo hexadecimal
```

98 El contenido del archivo cabecera_00.asm puede consultarse en el apéndice, página 149.

Para entender el resultado de los programas ejecutados es necesario tener conocimiento previo del funcionamiento del microprocesador Z80, registros, funciones, instrucciones, así como la forma en que el mismo microprocesador ejecuta el programa. En cada una de las pruebas presentadas se omite una explicación del programa, sin embargo, en todas las pruebas se corroboró que el resultado obtenido en forma practica es igual al obtenido en forma teórica. Para una referencia acerca de los métodos de programación, se sugiere leer ***Programación del microprocesador Z-80*** *de Elizabeth A. Nichols y Joseph C. Nichols Peter R. Rony y* ***Programación del Z80 con ensamblador*** *de Oliver LEPAPE.*

Las fotografías fueron tomadas a color y después tratadas por un programa de procesamiento de imágenes[99] para transferirlas a escala de grises. Esto con la finalidad de que las fotografías tuvieran un mejor contraste y el archivo electrónico no fuera difícil de manejar y procesar, ya que de haber incluido las fotos en color su tamaño hubiera crecido de forma importante. El T80 al igual que su símil el Z80 , posee mas de 150 instrucciones y cada una de ellas puede interactuar con varios registros por lo que de haber fotografiado todas esas instrucciones con cada uno de los registros con los que puede operar se habrían necesitado mas de 1000 fotografías, lo cual no esta dentro del alcance del presente trabajo. Sin embargo se tomaron instrucciones de cada grupo de operación del microprocesador para hacer las pruebas, estos grupos son: carga de 8 bits, carga de 16 bits, intercambio, transferencia y búsqueda, aritmética de 8 bits , de propósito general, aritmético de 16 bits , rotación y desplazamiento , manipulación de bits, salto , entrada salida, de llamada y retorno. Se comprobaron todas las instrucciones y se incluyeron en este trabajo solo una muestra de ellas.

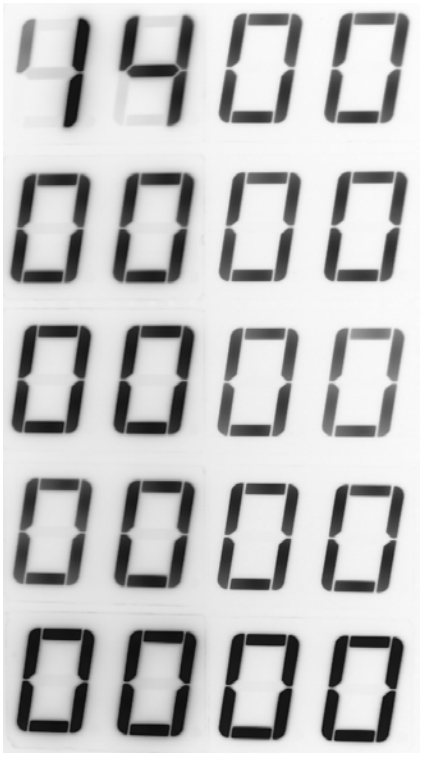
99 El programa es IrfanView , se consigue de forma gratuita y cuenta con múltiples herramientas para la manipulación de imágenes

include "/home/Tet/pasmo/cabecera_00.asm"

```
    ld    b,0x19
    ld    l,0x77
    ld    a,b
    ld    b,l
    call  seg
    halt
```

```
    ld    b,l
```

```
:1000000021F0FFF93E0006000E0016001E0026003B
:100010002E00CD1800C33500D30778D30079D30163
:100020007AD3027BD3037CD3047DD3053E00D30671
:100030003E00D308C900000000000000000000000DE
:0A004000 06192E777845 CD180076DA
:00000001FF
```

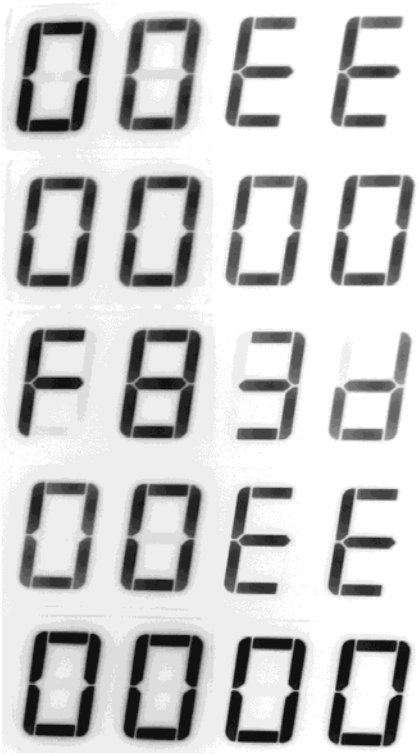


include "/home/Tet/pasmo/cabecera_00.asm"

```
    ld    b,0x14
    call  seg
    halt
```

```
    ld    b,0x14
```

```
:1000000021F0FFF93E0006000E0016001E0026003B
:100010002E00CD1800C33500D30778D30079D30163
:100020007AD3027BD3037CD3047DD3053E00D30671
:100030003E00D308C900000000000000000000000DE
:06004000 0614 CD18007645
:00000001FF
```

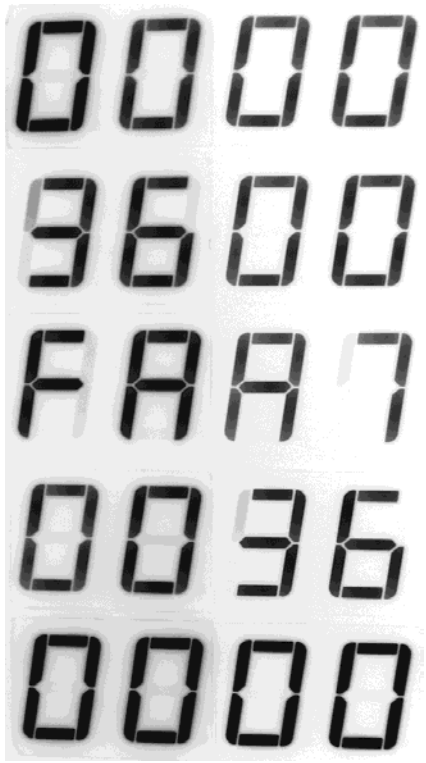



include "/home/Tet/pasmo/cabecera_00.asm"

```
      ld    hl,0xf83d
      ld    a,0xee
      ld    (hl),a
      ld    c,(hl)
      call  seg
      halt
```

```
      ld    c,(hl)
```

```
:1000000021F0FFF93E0006000E0016001E0026003B
:100010002E00CD1800C33500D30778D30079D30163
:100020007AD3027BD3037CD3047DD3053E00D30671
:100030003E00D308C900000000000000000000000DE
:0B004000213DF83EEE774ECD18007613
:00000001FF
```

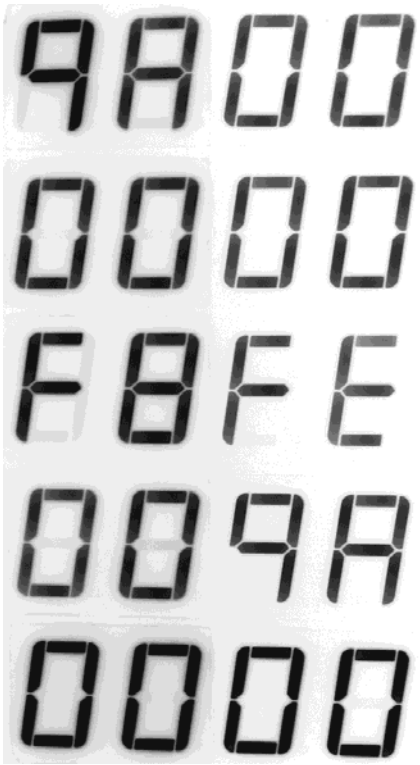



include "/home/Tet/pasmo/cabecera_00.asm"

```
      ld    hl,0xfaa7
      ld    a,0x36
      ld    (hl),a
      ld    ix,0xfa9d
      ld    d,(ix+0x0a)
      call  seg
      halt
```

```
      ld    d,(ix+0x0a)
```

```
:1000000021F0FFF93E0006000E0016001E0026003B
:100010002E00CD1800C33500D30778D30079D30163
:100020007AD3027BD3037CD3047DD3053E00D30671
:100030003E00D308C900000000000000000000000DE
:1000400021A7FA3E3677DD219DFADD560ACD18004C
:010050007639
:00000001FF
```

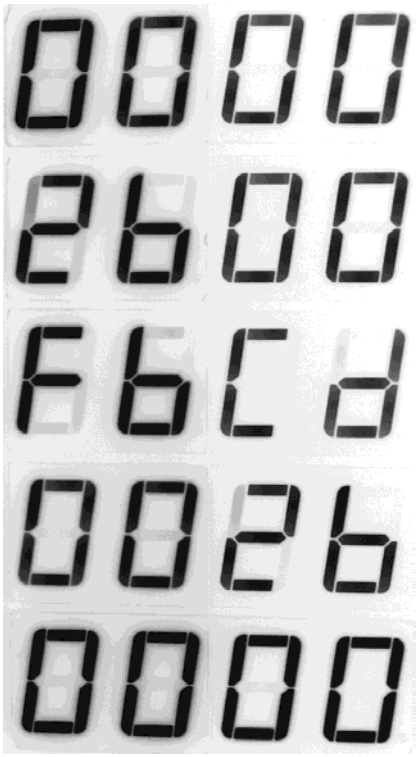



include "/home/Tet/pasmo/cabecera_00.asm"

```
    ld    hl,0xf8fe
    ld    b,0x9a
    ld    (hl),b
    ld    a,(hl)
    call  seg
    halt
```

```
    ld    (hl),b
```

```
:1000000021F0FFF93E0006000E0016001E0026003B
:100010002E00CD1800C33500D30778D30079D30163
:100020007AD3027BD3037CD3047DD3053E00D30671
:100030003E00D308C900000000000000000000000DE
:0B00400021FEF8069A707ECD180076B5
:00000001FF
```

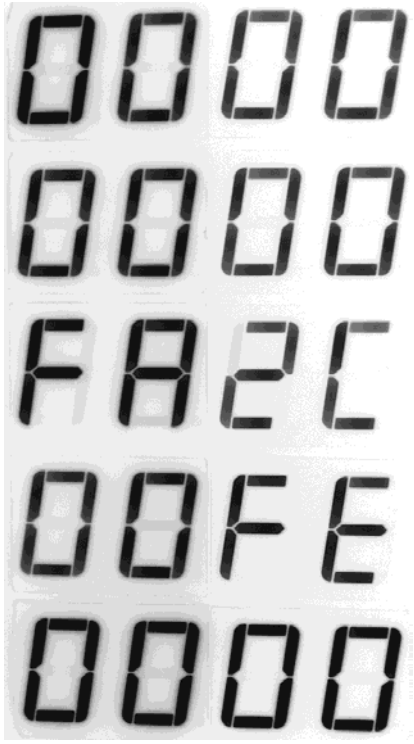



include "/home/Tet/pasmo/cabecera_00.asm"

```
    ld    d,0x2b
    ld    ix,0xfbaa
    ld    (ix+0x23),d
    ld    a,0x00
    ld    hl,0xfbcd
    ld    a,(hl)
    call  seg
    halt
```

```
    ld    (ix+0x23),d
```

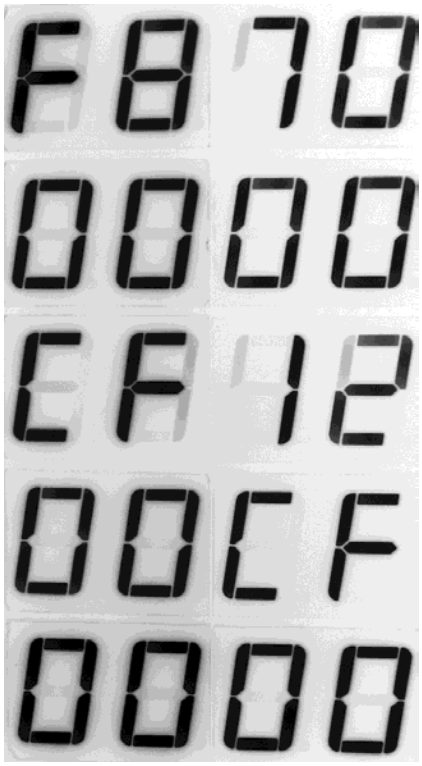
```
:1000000021F0FFF93E0006000E0016001E0026003B
:100010002E00CD1800C33500D30778D30079D30163
:100020007AD3027BD3037CD3047DD3053E00D30671
:100030003E00D308C900000000000000000000000DE
:10040000162BDD21AAFBDD72233E0021CDFB7ECDE8
:030050001800761F
:00000001FF
```




```
include "/home/Tet/pasmo/cabecera_00.asm"
      ld    hl,0xfa2c
      ld    (hl),0xfe
      ld    a,(0xfa2c)
      call   seg
      halt
```

```
      ld    a,(0xfa2c)
```

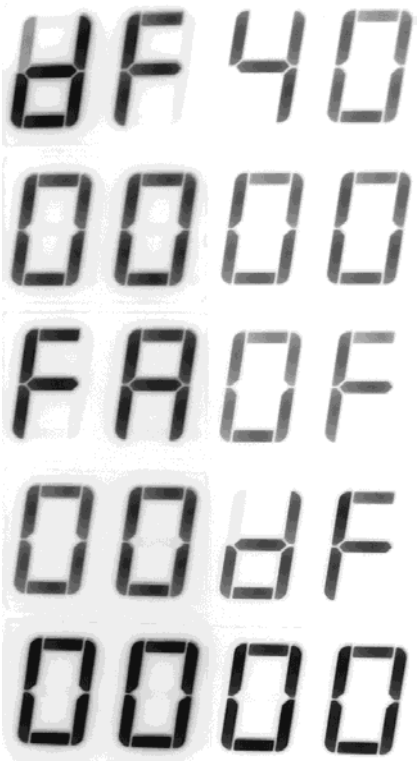
```
:1000000021F0FFF93E0006000E0016001E0026003B
:100010002E00CD1800C33500D30778D30079D30163
:100020007AD3027BD3037CD3047DD3053E00D30671
:100030003E00D308C900000000000000000000DE
:0C004000212CFA36FE3A2CFACD1800767E
:00000001FF
```




```
include "/home/Tet/pasmo/cabecera_00.asm"
      ld    hl,0xf870
      ld    b,h
      ld    c,l
      ld    a,0x12
      ld    (hl),a
      ld    a,0xcf
      inc    hl
      ld    (hl),a
      ld    hl,(0xf870)
      call   seg
      halt
```

```
      ld    hl,(0xf870)
```

```
:1000000021F0FFF93E0006000E0016001E0026003B
:100010002E00CD1800C33500D30778D30079D30163
:100020007AD3027BD3037CD3047DD3053E00D30671
:100030003E00D308C900000000000000000000DE
:10040002170F8444D3E12773ECF23772A70F8CDC9
:030050001800761F
:00000001FF
```

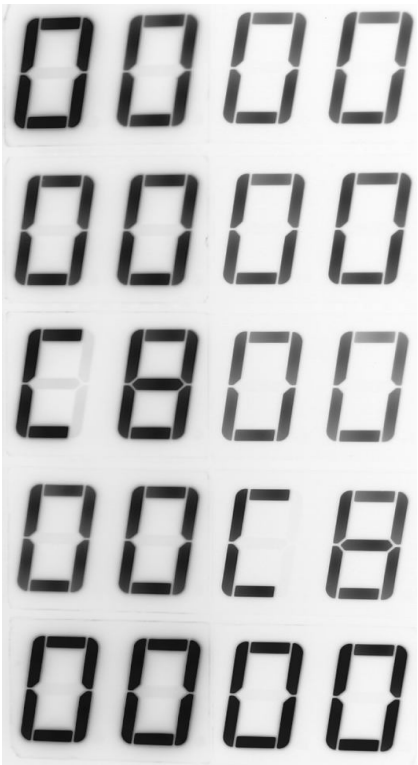



include "/home/Tet/pasmo/cabecera_00.asm"

```
      ld     hl,0xfa0e
      ld     a,0x40
      ld     (hl),a
      ld     a,0xdf
      inc    hl
      ld     (hl),a
      ld     bc,(0xfa0e)
      call   seg
      halt
```

**ld bc,(0xfa0e)**

```
:1000000021F0FFF93E0006000E0016001E0026003B
:100010002E00CD1800C33500D30778D30079D30163
:100020007AD3027BD3037CD3047DD3053E00D30671
:100030003E00D308C900000000000000000000DE
:10004000210EFA3E40773EDF2377ED4B0EFACD18B6
:02005000007638
:00000001FF
```

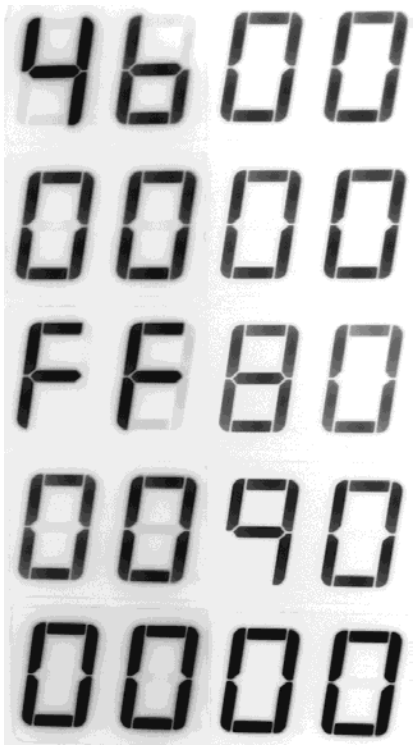



include "/home/Tet/pasmo/cabecera_00.asm"

```
      ld     a,0x5a
      ld     a,b
      ld     h,0xc8
      add    a,h
      call   seg
      halt
```

**ld h,0xc8**

```
:1000000021F0FFF93E0006000E0016001E0026003B
:100010002E00CD1800C33500D30778D30079D30163
:100020007AD3027BD3037CD3047DD3053E00D30671
:100030003E00D308C900000000000000000000DE
:0A0040003E5A7826C884CD180076D9
:00000001FF
```

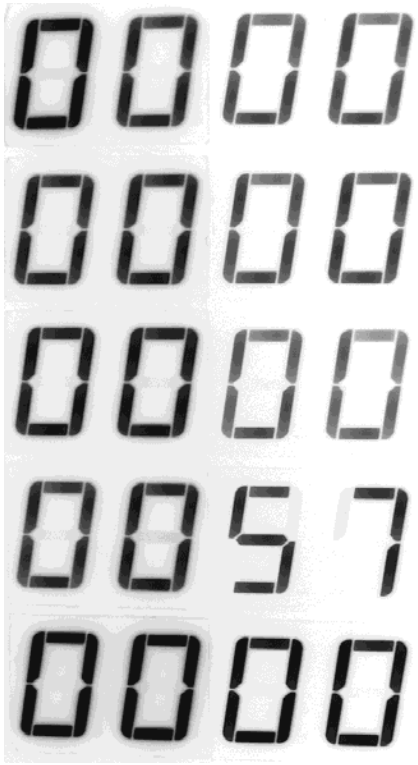



include "/home/Tet/pasmo/cabecera_00.asm"

```
        ld      hl,0xff80
        ld      a,0x4b
        ld      b,a
        ld      (hl),a
        ld      a,0x45
        add     a,(hl)
        call    seg
        halt
```

**add a,(hl)**

```
:1000000021F0FFF93E0006000E0016001E0026003B
:100010002E00CD1800C33500D30778D30079D30163
:100020007AD3027BD3037CD3047DD3053E00D30671
:100030003E00D308C900000000000000000000000DE
:0E004000 2180FF3E4B47773E4586 CD18007667
:00000001FF
```

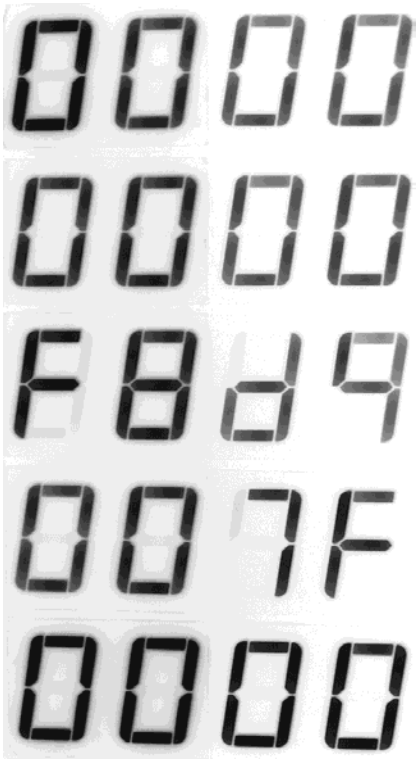



include "/home/Tet/pasmo/cabecera_00.asm"

```
        scf
        ld      a,0xab
        adc     a,a
        call    seg
        halt
```

**adc a,a**

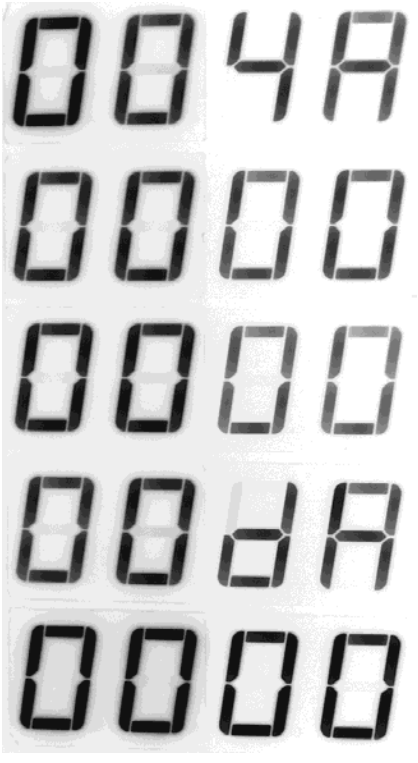
```
:1000000021F0FFF93E0006000E0016001E0026003B
:100010002E00CD1800C33500D30778D30079D30163
:100020007AD3027BD3037CD3047DD3053E00D30671
:100030003E00D308C900000000000000000000000DE
:08004000 373EAB8F CD180076AE
:00000001FF
```



```
include "/home/Tet/pasmo/cabecera_00.asm"
      ld    hl,0xf8d9
      ld    a,0xc9
      ld    (hl),a
      ld    ix,0xf8cb
      ld    a,0x48
      sub   (ix+0x0e)
      out   (0x00),a
      call  seg
      halt
```

```
      sub   (ix+0x0e)
```

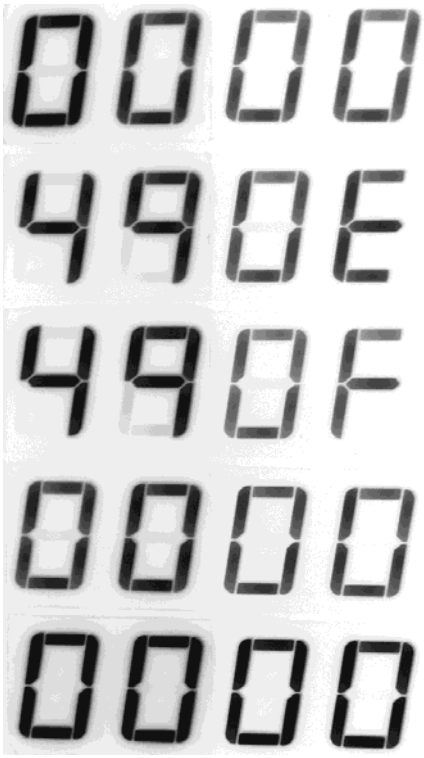
```
:1000000021F0FFF93E0006000E0016001E0026003B
:100010002E00CD1800C33500D30778D30079D30163
:100020007AD3027BD3037CD3047DD3053E00D30671
:100030003E00D308C900000000000000000000000DE
:10004000 21D9F83EC977DD21CBF83E48DD960ED3 A5
:05005000 00 CD18007650
:00000001FF
```



```
include "/home/Tet/pasmo/cabecera_00.asm"
      ld    a,0x25
      ld    c,0x4a
      scf
      sbc   a,c
      call  seg
      halt
```

```
     sbc    a,c
```

```
:1000000021F0FFF93E0006000E0016001E0026003B
:100010002E00CD1800C33500D30778D30079D30163
:100020007AD3027BD3037CD3047DD3053E00D30671
:100030003E00D308C900000000000000000000000DE
:0A0040003E250E4A3799CD180076D0
:00000001FF
```

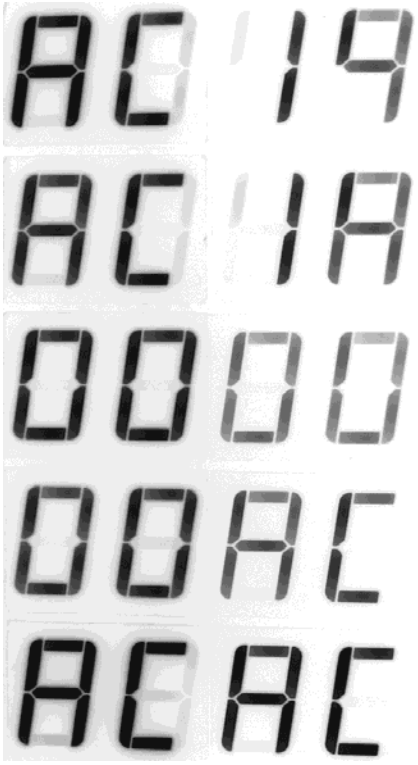


include "/home/Tet/pasmo/cabecera_00.asm"

```
      ld    hl,0x490e
      ld    d,h
      ld    e,l
      inc   hl
      call  seg
      halt

seg   inc   hl
```

```
:1000000021F0FFF93E0006000E0016001E0026003B
:100010002E00CD1800C33500D30778D30079D30163
:100020007AD3027BD3037CD3047DD3053E00D30671
:100030003E00D308C900000000000000000000000DE
:0A004000210E49545D23CD1800760F
:00000001FF
```

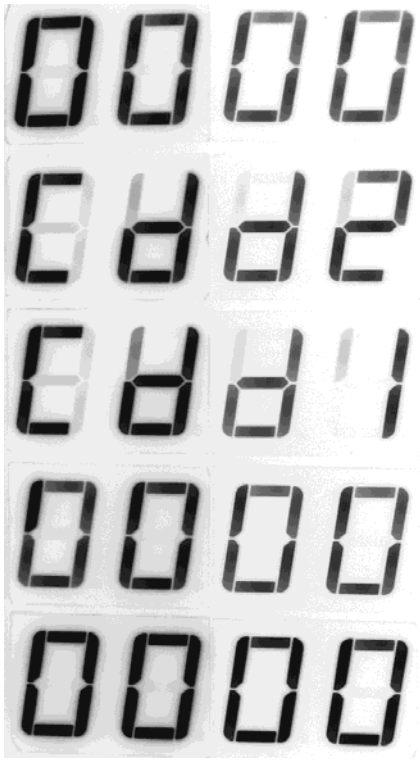


include "/home/Tet/pasmo/cabecera_00.asm"

```
      ld    bc,0xac1a
      ld    d,b
      ld    e,c
      dec   bc
      call  seg
      halt

seg   dec   bc
```

```
:1000000021F0FFF93E0006000E0016001E0026003B
:100010002E00CD1800C33500D30778D30079D30163
:100020007AD3027BD3037CD3047DD3053E00D30671
:100030003E00D308C900000000000000000000000DE
:0A004000011AAC50590BCD180076E0
:00000001FF
```

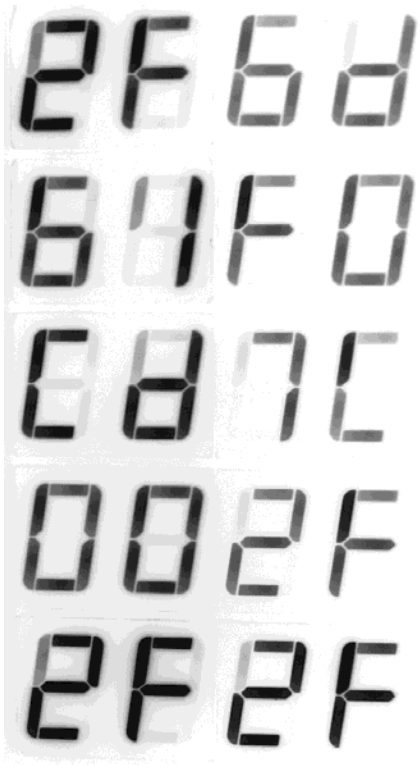


include "/home/Tet/pasmo/cabecera_00.asm"

```
    ld    de, 0xcdd1
    ld    h,d
    ld    l,e
    inc   de
    call  seg
    halt
```

**inc de**

```
:1000000021F0FFF93E0006000E0016001E0026003B
:100010002E00CD1800C33500D30778D30079D30163
:100020007AD3027BD3037CD3047DD3053E00D30671
:100030003E00D308C900000000000000000000000DE
:0A00400011D1CD626B13CD180076CC
:00000001FF
```

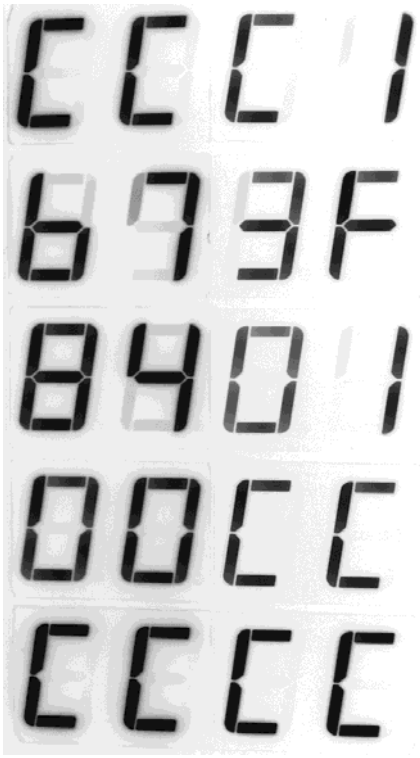


include "/home/Tet/pasmo/cabecera_00.asm"

```
    ld    hl, 0x2f6d
    ld    b,h
    ld    c,l
    ld    de,0x61f0
    scf
    sbc   hl,de
    call  seg
    halt
```

**sbc hl,de**

```
:1000000021F0FFF93E0006000E0016001E0026003B
:100010002E00CD1800C33500D30778D30079D30163
:100020007AD3027BD3037CD3047DD3053E00D30671
:100030003E00D308C900000000000000000000000DE
:0F004000216D2F444D11F06137ED52CD18007630
:00000001FF
```



```
include "/home/Tet/pasmo/cabecera_00.asm"
        ld hl,0xe910
        ld      hl,0xccc1
        ld      b,h
        ld      c,l
        ld      de,0xb73f
        scf
        adc     hl,de
        call    seg
        halt
```

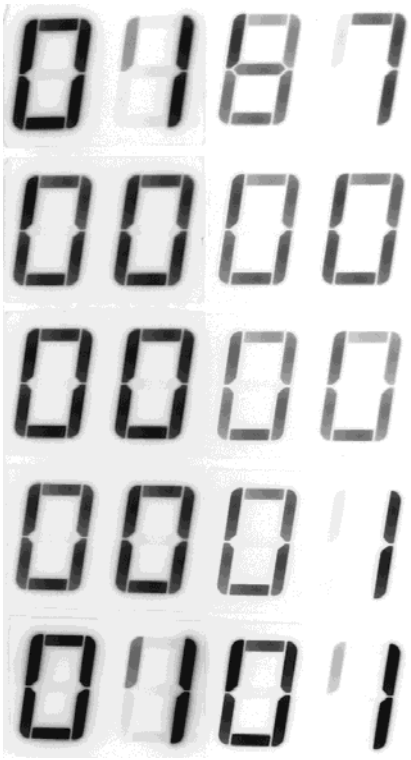
```
        adc     hl,de
```

```
:1000000021F0FFF93E0006000E0016001E0026003B
:100010002E00CD1800C33500D30778D30079D30163
:100020007AD3027BD3037CD3047DD3053E00D30671
:100030003E00D308C900000000000000000000000DE
:100040002110E921C1CC444D113FB737ED5ACD18ED
:02005000007638
:00000001FF
```

```
include "/home/Tet/pasmo/cabecera_00.asm"
        ld      hl,0xf5a5
        ld      d,h
        ld      e,l
        ld      bc,0xb4a8
        add     hl,bc
        call    seg
        halt
```

```
        add     hl,bc
```

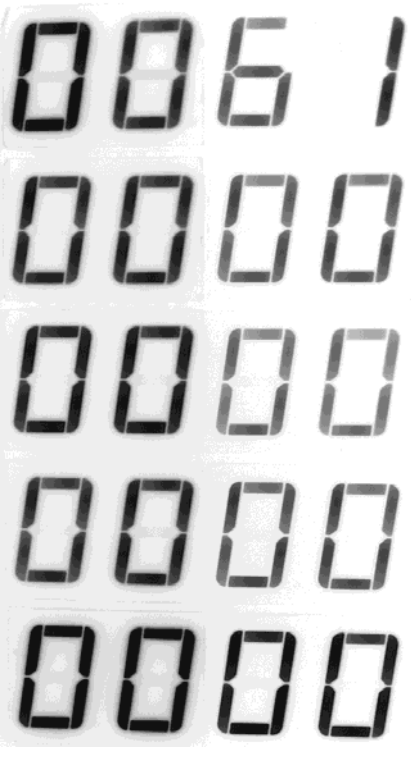
```
:1000000021F0FFF93E0006000E0016001E0026003B
:100010002E00CD1800C33500D30778D30079D30163
:100020007AD3027BD3037CD3047DD3053E00D30671
:100030003E00D308C900000000000000000000000DE
:0D00400021A5F5545D01A8B409CD18007686
:00000001FF
```



```
include "/home/Tet/pasmo/cabecera_00.asm"
      ld     c,0xc3
      ld     a,c
      out    (0x01),a
      scf
      rl     c
      jr     c,setC
      jp     term
setC: ld           b,0x01
term:
      call   seg
      halt
```

```
      rl     c
```

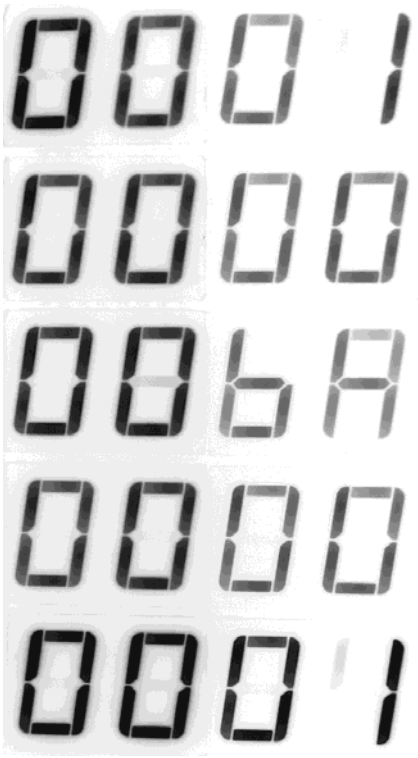
```
:1000000021F0FFF93E0006000E0016001E0026003B
:100010002E00CD1800C33500D30778D30079D30163
:100020007AD3027BD3037CD3047DD3053E00D30671
:100030003E00D308C900000000000000000000DE
:10004000 0EC379D30137CB113803C34F000601 CD5E
:030050001800761F
:00000001FF
```



```
include "/home/Tet/pasmo/cabecera_00.asm"
      ld     c,0xc2
      ld     a,c
      scf
      rrc    c
      jr     c,setC
      jp     term
setC: ld           b,0x01
term:
      call   seg
      halt
```

```
      rrc    c
```

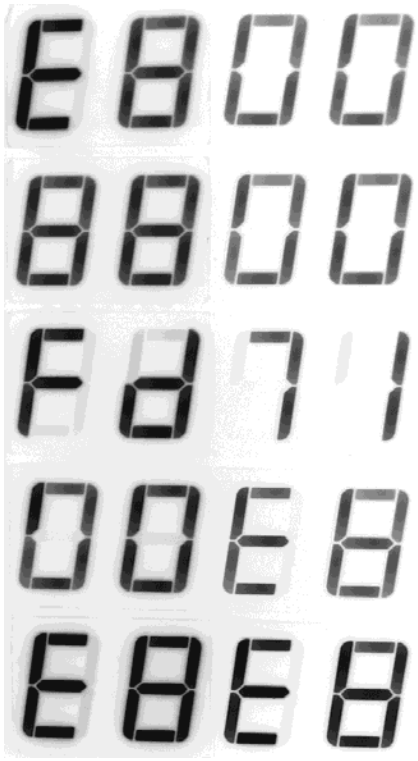
```
:1000000021F0FFF93E0006000E0016001E0026003B
:100010002E00CD1800C33500D30778D30079D30163
:100020007AD3027BD3037CD3047DD3053E00D30671
:100030003E00D308C900000000000000000000DE
:10004000 0EC27937CB093803C34D000601 CD180025
:010050007639
:00000001FF
```




```
include "/home/Tet/pasmo/cabecera_00.asm"
        ld      l,0x75
        ld      a,l
        rr      l
        jr      c,setC
        jp      term
setC:   ld      c,0x01
term:
        call    seg
        halt
```

```
        rr      l
```

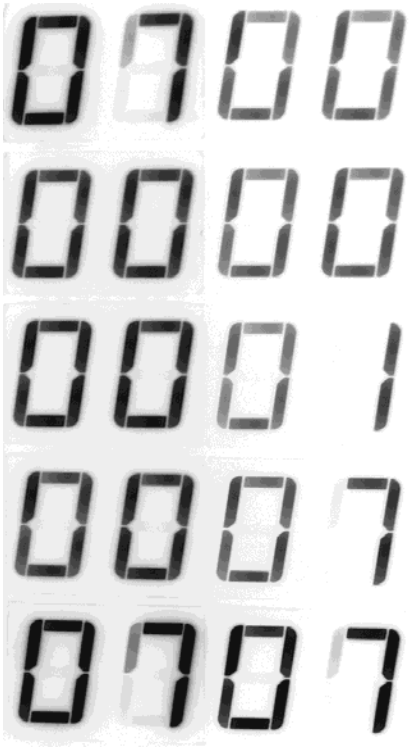
```
:1000000021F0FFF93E0006000E0016001E0026003B
:100010002E00CD1800C33500D30778D30079D30163
:100020007AD3027BD3037CD3047DD3053E00D30671
:100030003E00D308C900000000000000000000000DE
:10004000**2E757DCB1D3803C34C000E01**CD180076F4
:00000001FF
```




```
include "/home/Tet/pasmo/cabecera_00.asm"
        ld      hl,0xfd71
        ld      a,0x88
        ld      d,a
        ld      (hl),a
        ld      a,0xe1
        rld
        ld      b,a
        ld      a,(hl)
        call    seg
        halt
```

```
        rld
```

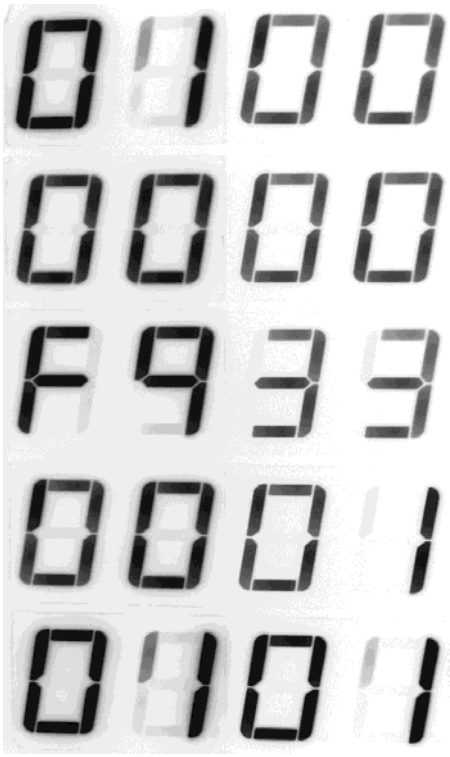
```
:1000000021F0FFF93E0006000E0016001E0026003B
:100010002E00CD1800C33500D30778D30079D30163
:100020007AD3027BD3037CD3047DD3053E00D30671
:100030003E00D308C900000000000000000000000DE
:10004000**2171FD3E8857773EE1ED6F477E**CD180068
:010050007639
:00000001FF
```



```
include "/home/Tet/pasmo/cabecera_00.asm"
      ld    a,0x07
      ld    b,a
      bit   3,a
      jp    z,cer
un
      ld    h,0x01
      jp    fin
cer
      ld    l,0x01
fin
      call  seg
      halt
```

```
      bit   3,a
```

```
:1000000021F0FFF93E0006000E0016001E0026003B
:100010002E00CD1800C33500D30778D30079D30163
:100020007AD3027BD3037CD3047DD3053E00D30671
:100030003E00D308C900000000000000000000000DE
:100040003E0747CB5FCA4D002601C34F002E01CDAE
:030050001800761F
:00000001FF
```



```
include "/home/Tet/pasmo/cabecera_00.asm"
      ld    hl,0xf933
      ld    a, 0xf6
      ld    (hl),a
      bit   4,l
      jp    z,cer
un
      ld    b,0x01
      jp    fin
cer
      ld    c,0x01
fin
      call  seg
      halt
```

```
      bit   4,l
```

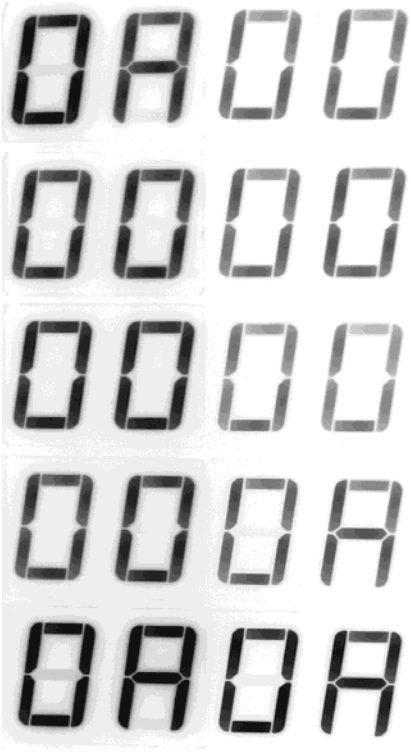
```
:1000000021F0FFF93E0006000E0016001E0026003B
:100010002E00CD1800C33500D30778D30079D30163
:100020007AD3027BD3037CD3047DD3053E00D30671
:100030003E00D308C900000000000000000000000DE
:100040002133F93EF677CB65CA50000601C3520052
:060050000E01CD18007640
:00000001FF
```

include "/home/Tet/pasmo/cabecera_00.asm"

```
      ld    l, 0x04
      ld    h,l
      set   5,l
      call  seg
      halt

      set   5,l
```

```
:1000000021F0FFF93E0006000E0016001E0026003B
:100010002E00CD1800C33500D30778D30079D30163
:100020007AD3027BD3037CD3047DD3053E00D30671
:100030003E00D308C900000000000000000000000DE
:090040002E0465CBEDCD1800760D
:00000001FF
```

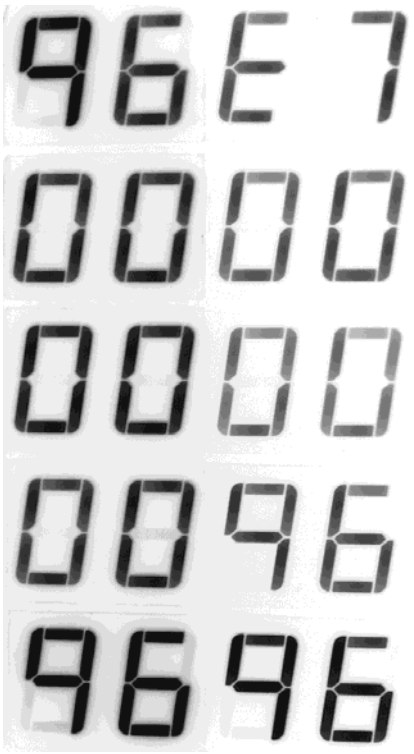


include "/home/Tet/pasmo/cabecera_00.asm"

```
      ld    b,0x2a
      ld    a,b
      res   5,b
      call  seg
      halt

      res   5,b
```

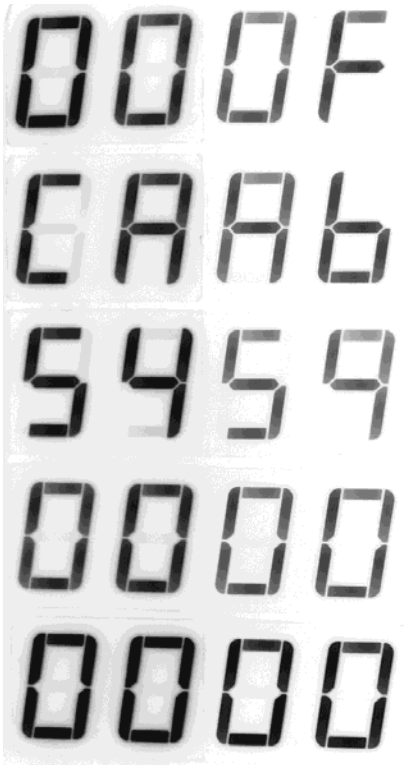
```
:1000000021F0FFF93E0006000E0016001E0026003B
:100010002E00CD1800C33500D30778D30079D30163
:100020007AD3027BD3037CD3047DD3053E00D30671
:100030003E00D308C900000000000000000000000DE
:090040002A78CBA8CD18007641
:00000001FF
```




```
include "/home/Tet/pasmo/cabecera_00.asm"
        ld      a, 0xe3
        ld      b, 0x96
        ld      c, 0xe7
jp      salto
        ld      d,0x07
        ld      e,0x9b
        ld      h, 0x94
salto
        call    seg
        halt
```

```
jp      salto
```

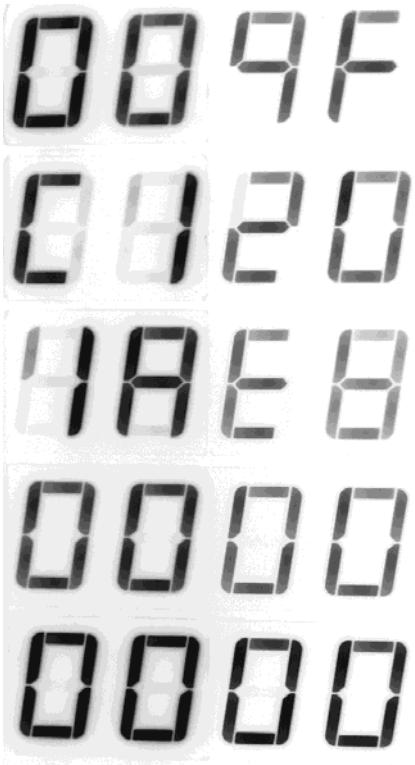
```
:1000000021F0FFF93E0006000E0016001E0026003B
:100010002E00CD1800C33500D30778D30079D30163
:100020007AD3027BD3037CD3047DD3053E00D30671
:100030003E00D308C900000000000000000000DE
:100040003EE306960EE7C34F0016071E9B2694CD8F
:030050001800761F
:00000001FF
```




```
include "/home/Tet/pasmo/cabecera_00.asm"
        ld      a, 0x03
        jr      $+4
        ld      b, 0x80
        ld      c, 0x0f
        ld      d, 0xca
        ld      e, 0xab
        ld      h, 0x54
        ld      l, 0x59
        call    seg
        halt
```

```
        jr      $+4
```

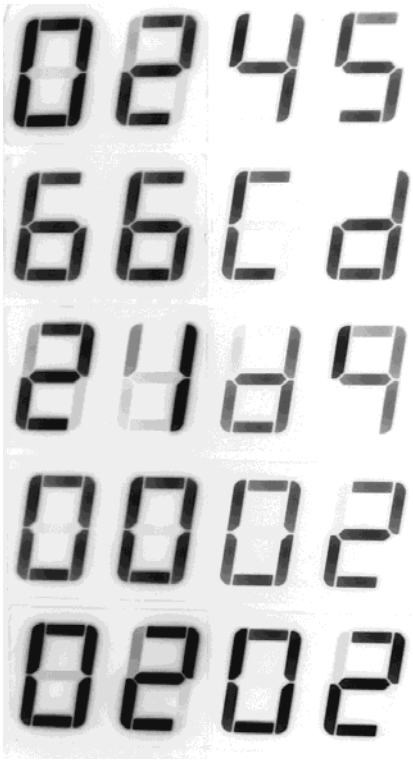
```
:1000000021F0FFF93E0006000E0016001E0026003B
:100010002E00CD1800C33500D30778D30079D30163
:100020007AD3027BD3037CD3047DD3053E00D30671
:100030003E00D308C900000000000000000000DE
:100040003E03180206800E0F16CA1EAB26542E5908
:04005000CD18007651
:00000001FF
```



```
include "/home/Tet/pasmo/cabecera_00.asm"
      ld     a,  0x3f
      scf
      jr     c,$+4
      ld     b,  0x0e
      ld     c,  0x9f
      ld     d,  0xc1
      ld     e,  0x20
      ld     h,  0x1a
      ld     l,  0xe8
      call   seg
      halt
```

```
      jr     c,$+4
```

```
:1000000021F0FFF93E0006000E0016001E0026003B
:100010002E00CD1800C33500D30778D30079D30163
:100020007AD3027BD3037CD3047DD3053E00D30671
:100030003E00D308C900000000000000000000DE
:100040003E3F373802060E0E9F16C11E20261A2E7E
:05005000E8CD18007668
:00000001FF
```



```
include "/home/Tet/pasmo/cabecera_00.asm"
      ld     a, 0x7f
      scf
      ccf
      jr     c,$+2
      ld     b, 0x02
      ld     c, 0x45
      ld     d, 0x66
      ld     e, 0xcd
      ld     h, 0x21
      ld     l, 0xd9
      call   seg
      halt
```

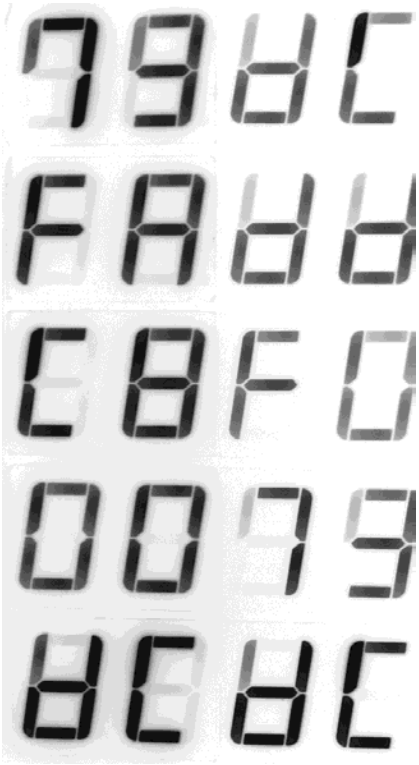
```
      jr     c,$+2
```

```
:1000000021F0FFF93E0006000E0016001E0026003B
:100010002E00CD1800C33500D30778D30079D30163
:100020007AD3027BD3037CD3047DD3053E00D30671
:100030003E00D308C900000000000000000000DE
:100040003E7F373F380006020E4516661ECD26213C
:060050002ED9CD18007648
:00000001FF
```

```
include "/home/Tet/pasmo/cabecera_00.asm"
      ld     a,0x0f
      bit    7,a
      jp     z,$+2
      ld     b,0xe5
      ld     c,0xae
      ld     d,0x0a
      ld     e,0xdf
      ld     h,0xac
      ld     l,0xc7
      call   seg
      halt
```

```
      jp     z,$+2
```

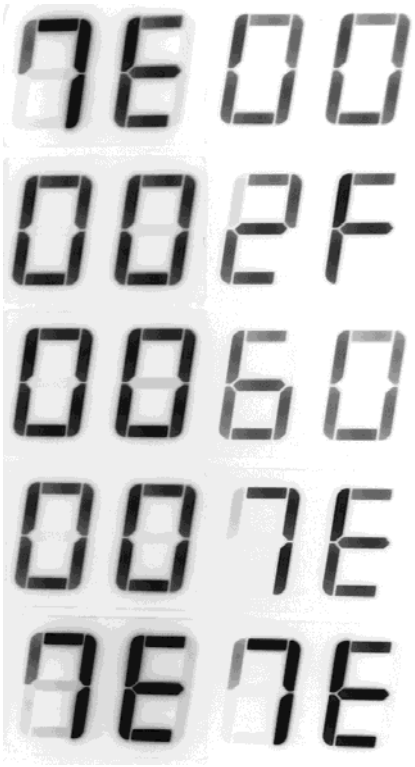
```
:1000000021F0FFF93E0006000E0016001E0026003B
:100010002E00CD1800C33500D30778D30079D30163
:100020007AD3027BD3037CD3047DD3053E00D30671
:100030003E00D308C900000000000000000000000DE
:100040003E0FCB7FCA460006E50EAE160A1EDF261F
:07005000AC2EC7CD180076AD
:00000001FF
```



```
include "/home/Tet/pasmo/cabecera_00.asm"
      ld     a,0x0f
      bit    0,a
      jp     z,$+2
      ld     b,0x73
      ld     c,0xdc
      ld     d,0xfa
      ld     e,0xdd
      ld     h,0xc8
      ld     l,0xf0
      call   seg
      halt
```

```
      jp     z,$+2
```

```
:1000000021F0FFF93E0006000E0016001E0026003B
:100010002E00CD1800C33500D30778D30079D30163
:100020007AD3027BD3037CD3047DD3053E00D30671
:100030003E00D308C900000000000000000000000DE
:100040003E0FCB47CA460006730EDC16FA1EDD26AD
:07005000C82EF0CD18007668
:00000001FF
```



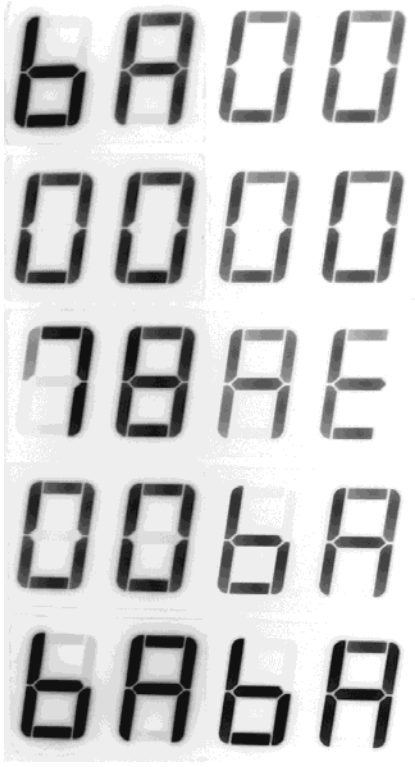
include "/home/Tet/pasmo/cabecera_00.asm"

```
      ld    a,0xbe
      ld    b, 0x7e
      ld    hl,0x0060
      jp    (hl)
      ld    c, 0x6f
      ld    d,0x9a

      org    0x0060
      ld    e, 0x2f
      call   seg
      halt
```

```
      jp    (hl)
```

```
:1000000021F0FFF93E0006000E0016001E0026003B
:100010002E00CD1800C33500D30778D30079D30163
:100020007AD3027BD3037CD3047DD3053E00D30671
:100030003E00D308C900000000000000000000000DE
:100040003EBE067E216000E90E6F169A00000000 99
:1000500000000000000000000000000000000000 A0
:0600600001E2FCD180076F2
:00000001FF
```



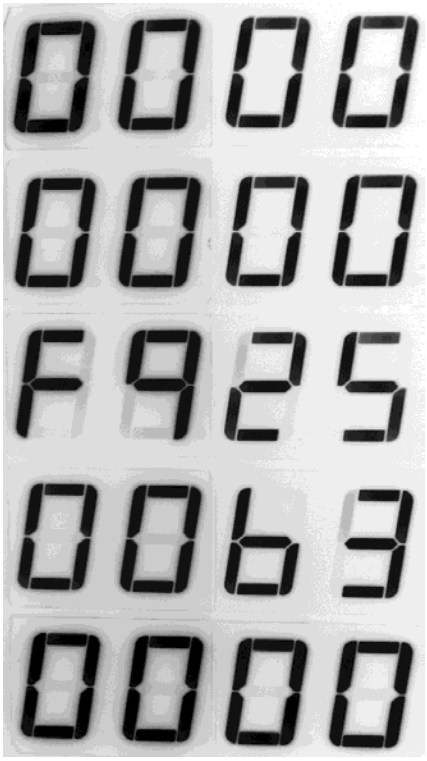
include "/home/Tet/pasmo/cabecera_00.asm"

```
      ld    ix,0x0070
      ld    a,0x6a
      ld    b,0xba
      jp    (ix)
      ld    c,0x45
      ld    d,0x12

      org    0x0070
      ld    hl,0x78ae
      call   seg
      halt
```

```
      jp    (ix)
```

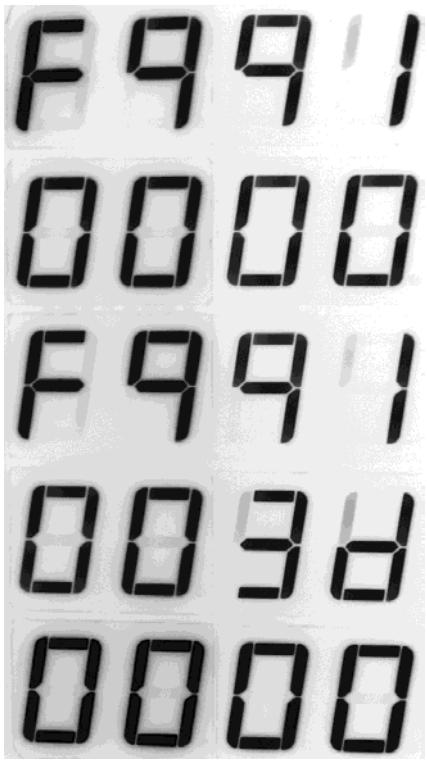
```
:1000000021F0FFF93E0006000E0016001E0026003B
:100010002E00CD1800C33500D30778D30079D30163
:100020007AD3027BD3037CD3047DD3053E00D30671
:100030003E00D308C900000000000000000000000DE
:10004000DD2170003E6A06BADDE90E4516120000 99
:1000500000000000000000000000000000000000 A0
:1000600000000000000000000000000000000000 90
:0700700021AE78CD180076E7
:00000001FF
```



```
include "/home/Tet/pasmo/cabecera_00.asm"
      ld    hl,0xf925
      ld    (hl),0xb3
      ld    a,(hl)
      call   seg
      halt
```

```
      ld    (hl),0xb3
```

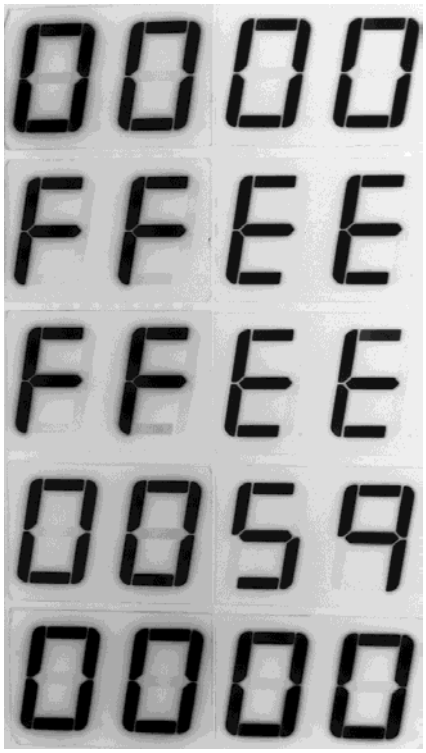
```
1000000021F0FFF93E0006000E0016001E0026003B
:100010002E00CD1800C33500D30778D30079D30163
:100020007AD3027BD3037CD3047DD3053E00D30671
:100030003E00D308C900000000000000000000000DE
:0A004000 2125F936B37E CD180076B5
:00000001FF
```



```
include "/home/Tet/pasmo/cabecera_00.asm"
      ld    hl,0xf991
      ld    (hl),0x3d
      ld    a,0x00
      ld    b,0xf9
      ld    c,0x91
      ld    a,0x00
      ld    a,(bc)
      call   seg
      halt
```

```
      ld    a,(bc)
```

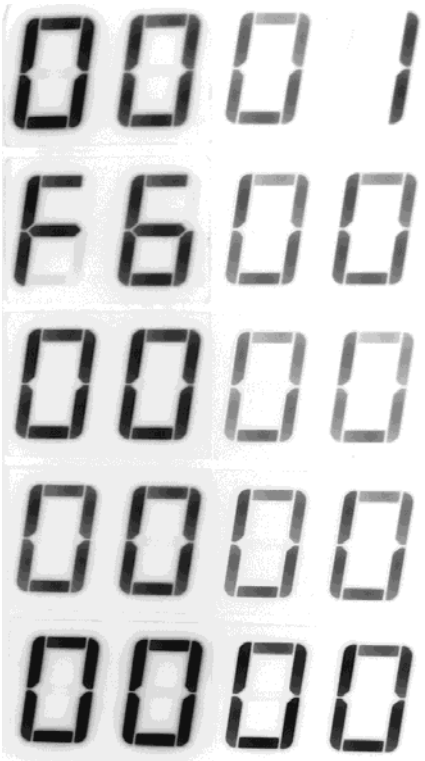
```
:1000000021F0FFF93E0006000E0016001E0026003B
:100010002E00CD1800C33500D30778D30079D30163
:100020007AD3027BD3037CD3047DD3053E00D30671
:100030003E00D308C900000000000000000000000DE
:10040000 2191F9363D3E0006F90E913E000A CD1889
:02005000007638
:00000001FF
```




```
include "/home/Tet/pasmo/cabecera_00.asm"
        ld      hl,0xffee
        ld      (hl),0x59
        ld      d,0xff
        ld      e,0xee
        ld      a,0x00
        ld      a,(de)
        call    seg
        halt
```

**ld a,(de)**

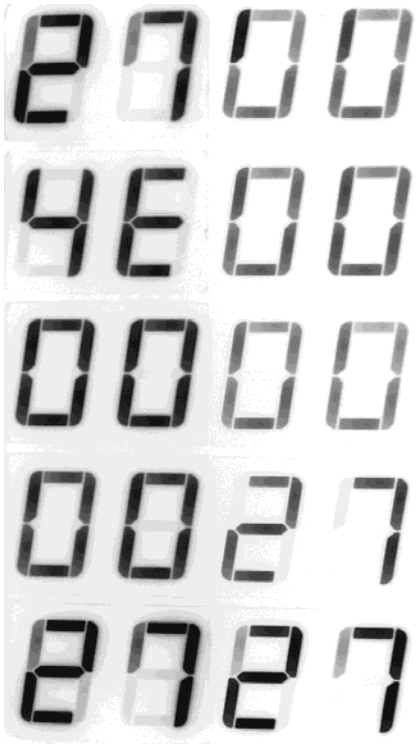
```
:1000000021F0FFF93E0006000E0016001E0026003B
:100010002E00CD1800C33500D30778D30079D30163
:100020007AD3027BD3037CD3047DD3053E00D30671
:100030003E00D308C900000000000000000000000DE
:1000400021EEFF365916FF1EEE3E001ACD1800763F
:00000001FF
```




```
include "/home/Tet/pasmo/cabecera_00.asm"
        ld      d,0xfb
        ld      a,d
        sla     d
        jr      c,setC
        jp      term
setC:   ld      c,0x01
term:
        call    seg
        halt
```

**sla d**

```
:1000000021F0FFF93E0006000E0016001E0026003B
:100010002E00CD1800C33500D30778D30079D30163
:100020007AD3027BD3037CD3047DD3053E00D30671
:100030003E00D308C900000000000000000000000DE
:1000400016FB7ACB223803C34C000E01CD18007684
:00000001FF
```




```
include "/home/Tet/pasmo/cabecera_00.asm"
        ld      b,0x4e
        ld      d,b
        sra     b
        jr      c,setC
        jp      term
setC:   ld      c,0x01
term:
        call    seg
        halt
```

```
      sra     b
```

```
:1000000021F0FFF93E0006000E0016001E0026003B
:100010002E00CD1800C33500D30778D30079D30163
:100020007AD3027BD3037CD3047DD3053E00D30671
:100030003E00D308C900000000000000000000000DE
:10004000064E50CB283803C34C000E01CD18007665
:00000001FF
```

# 6 Conclusiones.

El presente proyecto se dirigió a desarrollar métodos, y usar distintas herramientas para la implementación de un núcleo T80 en un dispositivo Xilinx Spartan 3E.

- Se obtuvieron los archivos fuentes en VHDL del núcleo T80 disponibles bajo licencia GPL la cual nos otorga libertad de uso , distribución y modificación de los mismos sin ningún temor a infringir derechos de copyright.
- Se modificaron los archivos para su correcta implementación dado que los originales usaban una palabra para el nombre de una terminal , esto debido a que el código estaba echo para una versión anterior de VHDL dado que fueron escritos en el año 2002
- Se eligió y demostró la forma de instalación y uso de el programa ensamblador PASMO también bajo licencia GPL.
- Se mostró la instalación de la suite de diseño Xilinx ISE así como la forma de implementar el núcleo T80 siguiendo todos los pasos necesarios desde el agregado de los archivos hasta la síntesis de los mismos.
- Se programó el dispositivo FPGA y se hicieron pruebas de funcionamiento haciendo uso del lenguaje ensamblador especifico del Z80, en todas las pruebas, el T80 funcionó de la forma esperada.

Las posibilidades de aplicación de este trabajo es muy amplio, se tienen las herramientas para usar el núcleo IP T80 en un FPGA no solo del fabricante Xilinx, este microprocesador sintetizado emula un micro Z80 en sus funciones esenciales. Dentro de las ventajas obtenidas, tenemos.

- El microprocesador sintetizado puede manejar velocidades mayores a las que nominalmente trabajan los micros tradicionales adquiridos en un proveedor de electrónica. Un Z80 en encapsulado DIP maneja una velocidad de 4 Mhz , en encapsulado PLCC se pueden alcanzar los 10 Mhz. La tarjeta de desarrollo Digilent en la que se sintetizó el proyecto puede proporcionar una velocidad de 50Mhz al núcleo T80.
- Manejar el desarrollo de sistemas embebidos ya sea por medio de esquemáticos, lenguaje VHDL , Verilog u otros lenguajes de descripción de hardware es mas rápido, dado de que los ciclos de implementación-prueba se acortan. Con esto un error detectado en el funcionamiento puede ser corregido de manera eficaz y no tener que esperar a seguir los fallos en alambrado, conexión o alimentación de forma constante.
- Las herramientas de software compatible con el procesador T80 (ZilogZ80 ) es muy variado, desde compiladores, ensambladores, y simuladores tanto comerciales como gratuitos pueden ser obtenidos de manera muy accesible.
- La tecnología que se presentó es independiente de la arquitectura del FPGA empleado, pueden implementarse núcleos T80 en plataformas de distintos fabricantes.

Se obtuvieron los resultados esperados, se desarrolló la teoría y se implementó en forma practica la síntesis del núcleo en hardware FPGA Xilinx Spartan . Se hicieron pruebas que comprobaron el funcionamiento del microprocesador programado con software generado por herramientas destinadas a ser usadas por el Z80.

# 7 Bibliografía.


1. Construya una microcomputadora basada en el z80: Guia de diseño y funcionamiento
   Steve Ciarcia ; tr. Luis Joyanes Aguilar
   McGraw-Hill 1988

2. Programación del Z80
   Rodnay Zaks
   Anaya Multimedia 1990

3. Programación del Z80 con ensamblador
   Oliver LEPAPE
   Paraninfo 1985

4. Programación del microprocesador z-80
   Elizabeth A. Nichols, Joseph C. Nichols Peter R. Rony ; traducido por Manuel Puigbo Rocafort
   Barcelona Mexico Marcombo c1981

5. Programacion del microprocesador z-80
   Elizabeth A. Nichols, Joseph C. Nichols Peter R. Rony ; traducido por Manuel Puigbo Rocafort
   Barcelona Mexico Marcombo c1981

6. Rapid system prototyping with FPGAs
   R. C. Cofer and Benjamin F. Harding
   Boston Elsevier 2005

7. Diseño de sistemas digitales con VHDL
   Serafín Alfonso Pérez López, Enrique Soto Campos /2002

8. Dispositivos lógicos programables (PLDs) :diseño práctico de aplicaciones
   Garcia Iglesias, Jose Manuel /2006

9. Electrónica digital :aplicaciones y problemas con VHDL
   Artigas Maestre, Jose Ignacio /c2002

10. Fundamentals of digital logic with VHDL design
    Stephen Brown and Zvonko G. Vranesic /2009

11. VHDL :lenguaje para síntesis y modelado de circuitos
    Fernando Pardo Carpio, José A. Boluda /c2004

# Apéndice

## 8.1 T80.vhd

IMPORTANTE -> Por cuestiones de compatibilidad ya que el código fuente fue escrito en el 2002, se ha sustituido en todos los archivos ( T80.vhd, T80_MCODE.vhd , T80_ALU.vhd, T80_REG.vhd ,T80_Pack.vhd, y T80se.vhd ) la leyenda Write por Write_80 con el fin de evitar errores en la compilación. En los listados de este apéndice se encuentra el contenido de todos los archivos, de forma resaltada y en negritas se observa el lugar en donde tuvo lugar la sustitución.

```
--
-- Z80 compatible microprocessor core
--
-- Version : 0247
--
-- Copyright (c) 2001-2002 Daniel Wallner (jesus@opencores.org)
--
-- All rights reserved
--
-- Redistribution and use in source and synthezised forms, with or without
-- modification, are permitted provided that the following conditions are met:
--
-- Redistributions of source code must retain the above copyright notice,
-- this list of conditions and the following disclaimer.
--
-- Redistributions in synthesized form must reproduce the above copyright
-- notice, this list of conditions and the following disclaimer in the
-- documentation and/or other materials provided with the distribution.
--
-- Neither the name of the author nor the names of other contributors may
-- be used to endorse or promote products derived from this software without
-- specific prior written permission.
--
-- THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
-- AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO,
-- THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
-- PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE
-- LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
-- CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
-- SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
-- INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
-- CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
-- ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
-- POSSIBILITY OF SUCH DAMAGE.
--
-- Please report bugs to the author, but before you do so, please
-- make sure that this is not a derivative work and that
-- you have the latest version of this file.
--
-- The latest version of this file can be found at:
--      http://www.opencores.org/cvsweb.shtml/t80/
--
-- Limitations :
--
-- File history :
--
--      0208 : First complete release
--
--      0210 : Fixed wait and halt
--
--      0211 : Fixed Refresh addition and IM 1
--
--      0214 : Fixed mostly flags, only the block instructions now fail the zex regression test
--
--      0232 : Removed refresh address output for Mode > 1 and added DJNZ M1_n fix by Mike Johnson
```

```
--
--	0235 : Added clock enable and IM 2 fix by Mike Johnson
--
--	0237 : Changed 8080 I/O address output, added IntE output
--
--	0238 : Fixed (IX/IY+d) timing and 16 bit ADC and SBC zero flag
--
--	0240 : Added interrupt ack fix by Mike Johnson, changed (IX/IY+d) timing and changed flags in GB mode
--
--	0242 : Added I/O wait, fixed refresh address, moved some registers to RAM
--
--	0247 : Fixed bus req/ack cycle
--

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;
use work.T80_Pack.all;

entity T80 is
	generic(
		Mode : integer := 1;	-- 0 => Z80, 1 => Fast Z80, 2 => 8080, 3 => GB
		IOWait : integer := 0; -- 1 => Single cycle I/O, 1 => Std I/O cycle
		Flag_C : integer := 0;
		Flag_N : integer := 1;
		Flag_P : integer := 2;
		Flag_X : integer := 3;
		Flag_H : integer := 4;
		Flag_Y : integer := 5;
		Flag_Z : integer := 6;
		Flag_S : integer := 7
	);
	port(
		RESET_n		: in std_logic;
		CLK_n		: in std_logic;
		CEN			: in std_logic;
		WAIT_n		: in std_logic;
		INT_n		: in std_logic;
		NMI_n		: in std_logic;
		BUSRQ_n		: in std_logic;
		M1_n		: out std_logic;
		IORQ		: out std_logic;
		NoRead		: out std_logic;
		Write_80	: out std_logic;
		RFSH_n		: out std_logic;
		HALT_n		: out std_logic;
		BUSAK_n		: out std_logic;
		A			: out std_logic_vector(15 downto 0);
		DInst		: in std_logic_vector(7 downto 0);
		DI			: in std_logic_vector(7 downto 0);
		DO			: out std_logic_vector(7 downto 0);
		MC			: out std_logic_vector(2 downto 0);
		TS			: out std_logic_vector(2 downto 0);
		IntCycle_n	: out std_logic;
		IntE		: out std_logic;
		Stop		: out std_logic
	);
end T80;

architecture rtl of T80 is

	constant aNone	: std_logic_vector(2 downto 0) := "111";
	constant aBC	: std_logic_vector(2 downto 0) := "000";
	constant aDE	: std_logic_vector(2 downto 0) := "001";
	constant aXY	: std_logic_vector(2 downto 0) := "010";
	constant aIOA	: std_logic_vector(2 downto 0) := "100";
	constant aSP	: std_logic_vector(2 downto 0) := "101";
	constant aZI	: std_logic_vector(2 downto 0) := "110";

	-- Registers
```

```
signal ACC, F                   : std_logic_vector(7 downto 0);
signal Ap, Fp                   : std_logic_vector(7 downto 0);
signal I                        : std_logic_vector(7 downto 0);
signal R                        : unsigned(7 downto 0);
signal SP, PC                   : unsigned(15 downto 0);
signal RegDIH                   : std_logic_vector(7 downto 0);
signal RegDIL                   : std_logic_vector(7 downto 0);
signal RegBusA                  : std_logic_vector(15 downto 0);
signal RegBusB                  : std_logic_vector(15 downto 0);
signal RegBusC                  : std_logic_vector(15 downto 0);
signal RegAddrA_r       : std_logic_vector(2 downto 0);
signal RegAddrA                 : std_logic_vector(2 downto 0);
signal RegAddrB_r       : std_logic_vector(2 downto 0);
signal RegAddrB                 : std_logic_vector(2 downto 0);
signal RegAddrC                 : std_logic_vector(2 downto 0);
signal RegWEH                   : std_logic;
signal RegWEL                   : std_logic;
signal Alternate        : std_logic;

-- Help Registers
signal TmpAddr                  : std_logic_vector(15 downto 0); -- Temporary address register
signal IR                       : std_logic_vector(7 downto 0);          -- Instruction register
signal ISet                     : std_logic_vector(1 downto 0);          -- Instruction set selector
signal RegBusA_r        : std_logic_vector(15 downto 0);

signal ID16                             : signed(15 downto 0);
signal Save_Mux                 : std_logic_vector(7 downto 0);

signal TState                   : unsigned(2 downto 0);
signal MCycle                   : std_logic_vector(2 downto 0);
signal IntE_FF1                 : std_logic;
signal IntE_FF2                 : std_logic;
signal Halt_FF                  : std_logic;
signal BusReq_s                 : std_logic;
signal BusAck                   : std_logic;
signal ClkEn                    : std_logic;
signal NMI_s                    : std_logic;
signal INT_s                    : std_logic;
signal IStatus                  : std_logic_vector(1 downto 0);

signal DI_Reg                   : std_logic_vector(7 downto 0);
signal T_Res                    : std_logic;
signal XY_State                 : std_logic_vector(1 downto 0);
signal Pre_XY_F_M       : std_logic_vector(2 downto 0);
signal NextIs_XY_Fetch  : std_logic;
signal XY_Ind                   : std_logic;
signal No_BTR                   : std_logic;
signal BTR_r                    : std_logic;
signal Auto_Wait        : std_logic;
signal Auto_Wait_t1     : std_logic;
signal Auto_Wait_t2     : std_logic;
signal IncDecZ                  : std_logic;

-- ALU signals
signal BusB                             : std_logic_vector(7 downto 0);
signal BusA                             : std_logic_vector(7 downto 0);
signal ALU_Q                    : std_logic_vector(7 downto 0);
signal F_Out                    : std_logic_vector(7 downto 0);

-- Registered micro code outputs
signal Read_To_Reg_r    : std_logic_vector(4 downto 0);
signal Arith16_r        : std_logic;
signal Z16_r                    : std_logic;
signal ALU_Op_r                 : std_logic_vector(3 downto 0);
signal Save_ALU_r       : std_logic;
signal PreserveC_r      : std_logic;
signal MCycles                  : std_logic_vector(2 downto 0);

-- Micro code outputs
signal MCycles_d        : std_logic_vector(2 downto 0);
```

```
	signal TStates			: std_logic_vector(2 downto 0);
	signal IntCycle			: std_logic;
	signal NMICycle			: std_logic;
	signal Inc_PC			: std_logic;
	signal Inc_WZ			: std_logic;
	signal IncDec_16		: std_logic_vector(3 downto 0);
	signal Prefix			: std_logic_vector(1 downto 0);
	signal Read_To_Acc		: std_logic;
	signal Read_To_Reg		: std_logic;
	signal Set_BusB_To		: std_logic_vector(3 downto 0);
	signal Set_BusA_To		: std_logic_vector(3 downto 0);
	signal ALU_Op			: std_logic_vector(3 downto 0);
	signal Save_ALU			: std_logic;
	signal PreserveC		: std_logic;
	signal Arith16			: std_logic;
	signal Set_Addr_To		: std_logic_vector(2 downto 0);
	signal Jump				: std_logic;
	signal JumpE			: std_logic;
	signal JumpXY			: std_logic;
	signal Call				: std_logic;
	signal RstP				: std_logic;
	signal LDZ				: std_logic;
	signal LDW				: std_logic;
	signal LDSPHL			: std_logic;
	signal IORQ_i			: std_logic;
	signal Special_LD		: std_logic_vector(2 downto 0);
	signal ExchangeDH		: std_logic;
	signal ExchangeRp		: std_logic;
	signal ExchangeAF		: std_logic;
	signal ExchangeRS		: std_logic;
	signal I_DJNZ			: std_logic;
	signal I_CPL			: std_logic;
	signal I_CCF			: std_logic;
	signal I_SCF			: std_logic;
	signal I_RETN			: std_logic;
	signal I_BT				: std_logic;
	signal I_BC				: std_logic;
	signal I_BTR			: std_logic;
	signal I_RLD			: std_logic;
	signal I_RRD			: std_logic;
	signal I_INRC			: std_logic;
	signal SetDI			: std_logic;
	signal SetEI			: std_logic;
	signal IMode			: std_logic_vector(1 downto 0);
	signal Halt				: std_logic;

begin

	mcode : T80_MCode
		generic map(
			Mode => Mode,
			Flag_C => Flag_C,
			Flag_N => Flag_N,
			Flag_P => Flag_P,
			Flag_X => Flag_X,
			Flag_H => Flag_H,
			Flag_Y => Flag_Y,
			Flag_Z => Flag_Z,
			Flag_S => Flag_S)
		port map(
			IR => IR,
			ISet => ISet,
			MCycle => MCycle,
			F => F,
			NMICycle => NMICycle,
			IntCycle => IntCycle,
			MCycles => MCycles_d,
			TStates => TStates,
			Prefix => Prefix,
			Inc_PC => Inc_PC,
```

```
                Inc_WZ => Inc_WZ,
                IncDec_16 => IncDec_16,
                Read_To_Acc => Read_To_Acc,
                Read_To_Reg => Read_To_Reg,
                Set_BusB_To => Set_BusB_To,
                Set_BusA_To => Set_BusA_To,
                ALU_Op => ALU_Op,
                Save_ALU => Save_ALU,
                PreserveC => PreserveC,
                Arith16 => Arith16,
                Set_Addr_To => Set_Addr_To,
                IORQ => IORQ_i,
                Jump => Jump,
                JumpE => JumpE,
                JumpXY => JumpXY,
                Call => Call,
                RstP => RstP,
                LDZ => LDZ,
                LDW => LDW,
                LDSPHL => LDSPHL,
                Special_LD => Special_LD,
                ExchangeDH => ExchangeDH,
                ExchangeRp => ExchangeRp,
                ExchangeAF => ExchangeAF,
                ExchangeRS => ExchangeRS,
                I_DJNZ => I_DJNZ,
                I_CPL => I_CPL,
                I_CCF => I_CCF,
                I_SCF => I_SCF,
                I_RETN => I_RETN,
                I_BT => I_BT,
                I_BC => I_BC,
                I_BTR => I_BTR,
                I_RLD => I_RLD,
                I_RRD => I_RRD,
                I_INRC => I_INRC,
                SetDI => SetDI,
                SetEI => SetEI,
                IMode => IMode,
                Halt => Halt,
                NoRead => NoRead,
                Write_80 => Write_80);

    alu : T80_ALU
        generic map(
                Mode => Mode,
                Flag_C => Flag_C,
                Flag_N => Flag_N,
                Flag_P => Flag_P,
                Flag_X => Flag_X,
                Flag_H => Flag_H,
                Flag_Y => Flag_Y,
                Flag_Z => Flag_Z,
                Flag_S => Flag_S)
        port map(
                Arith16 => Arith16_r,
                Z16 => Z16_r,
                ALU_Op => ALU_Op_r,
                IR => IR(5 downto 0),
                ISet => ISet,
                BusA => BusA,
                BusB => BusB,
                F_In => F,
                Q => ALU_Q,
                F_Out => F_Out);

    ClkEn <= CEN and not BusAck;

    T_Res <= '1' when TState = unsigned(TStates) else '0';
```

```
NextIs_XY_Fetch <= '1' when XY_State /= "00" and XY_Ind = '0' and
                                                        ((Set_Addr_To = aXY) or
                                                        (MCycle = "001" and IR = "11001011") or
                                                        (MCycle = "001" and IR = "00110110")) else '0';

Save_Mux <= BusB when ExchangeRp = '1' else
        DI_Reg when Save_ALU_r = '0' else
        ALU_Q;

process (RESET_n, CLK_n)
begin
        if RESET_n = '0' then
                PC <= (others => '0');  -- Program Counter
                A <= (others => '0');
                TmpAddr <= (others => '0');
                IR <= "00000000";
                ISet <= "00";
                XY_State <= "00";
                IStatus <= "00";
                MCycles <= "000";
                DO <= "00000000";

                ACC <= (others => '1');
                F <= (others => '1');
                Ap <= (others => '1');
                Fp <= (others => '1');
                I <= (others => '0');
                R <= (others => '0');
                SP <= (others => '1');
                Alternate <= '0';

                Read_To_Reg_r <= "00000";
                F <= (others => '1');
                Arith16_r <= '0';
                BTR_r <= '0';
                Z16_r <= '0';
                ALU_Op_r <= "0000";
                Save_ALU_r <= '0';
                PreserveC_r <= '0';
                XY_Ind <= '0';

        elsif CLK_n'event and CLK_n = '1' then

                if ClkEn = '1' then

                ALU_Op_r <= "0000";
                Save_ALU_r <= '0';
                Read_To_Reg_r <= "00000";

                MCycles <= MCycles_d;

                if IMode /= "11" then
                        IStatus <= IMode;
                end if;

                Arith16_r <= Arith16;
                PreserveC_r <= PreserveC;
                if ISet = "10" and ALU_OP(2) = '0' and ALU_OP(0) = '1' and MCycle = "011" then
                        Z16_r <= '1';
                else
                        Z16_r <= '0';
                end if;

                if MCycle  = "001" and TState(2) = '0' then
                -- MCycle = 1 and TState = 1, 2, or 3

                        if TState = 2 and Wait_n = '1' then
                                if Mode < 2 then
                                        A(7 downto 0) <= std_logic_vector(R);
                                        A(15 downto 8) <= I;
```

```
                                        R(6 downto 0) <= R(6 downto 0) + 1;
                                end if;

                                if Jump = '0' and Call = '0' and NMICycle = '0' and IntCycle = '0' and not (Halt_FF = '1' or Halt =
'1') then
                                        PC <= PC + 1;
                                end if;

                                if IntCycle = '1' and IStatus = "01" then
                                        IR <= "11111111";
                                elsif Halt_FF = '1' or (IntCycle = '1' and IStatus = "10") or NMICycle = '1' then
                                        IR <= "00000000";
                                else
                                        IR <= DInst;
                                end if;

                                ISet <= "00";
                                if Prefix /= "00" then
                                        if Prefix = "11" then
                                                if IR(5) = '1' then
                                                        XY_State <= "10";
                                                else
                                                        XY_State <= "01";
                                                end if;
                                        else
                                                if Prefix = "10" then
                                                        XY_State <= "00";
                                                        XY_Ind <= '0';
                                                end if;
                                                ISet <= Prefix;
                                        end if;
                                else
                                        XY_State <= "00";
                                        XY_Ind <= '0';
                                end if;
                        end if;

                else
                -- either (MCycle > 1) OR (MCycle = 1 AND TState > 3)

                        if MCycle = "110" then
                                XY_Ind <= '1';
                                if Prefix = "01" then
                                        ISet <= "01";
                                end if;
                        end if;

                        if T_Res = '1' then
                                BTR_r <= (I_BT or I_BC or I_BTR) and not No_BTR;
                                if Jump = '1' then
                                        A(15 downto 8) <= DI_Reg;
                                        A(7 downto 0) <= TmpAddr(7 downto 0);
                                        PC(15 downto 8) <= unsigned(DI_Reg);
                                        PC(7 downto 0) <= unsigned(TmpAddr(7 downto 0));
                                elsif JumpXY = '1' then
                                        A <= RegBusC;
                                        PC <= unsigned(RegBusC);
                                elsif Call = '1' or RstP = '1' then
                                        A <= TmpAddr;
                                        PC <= unsigned(TmpAddr);
                                elsif MCycle = MCycles and NMICycle = '1' then
                                        A <= "0000000001100110";
                                        PC <= "0000000001100110";
                                elsif MCycle = "011" and IntCycle = '1' and IStatus = "10" then
                                        A(15 downto 8) <= I;
                                        A(7 downto 0) <= TmpAddr(7 downto 0);
                                        PC(15 downto 8) <= unsigned(I);
                                        PC(7 downto 0) <= unsigned(TmpAddr(7 downto 0));
                                else
                                        case Set_Addr_To is
```

```
		when aXY =>
			if XY_State = "00" then
				A <= RegBusC;
			else
				if NextIs_XY_Fetch = '1' then
					A <= std_logic_vector(PC);
				else
					A <= TmpAddr;
				end if;
			end if;
		when aIOA =>
			if Mode = 3 then
				-- Memory map I/O on GBZ80
				A(15 downto 8) <= (others => '1');
			elsif Mode = 2 then
				-- Duplicate I/O address on 8080
				A(15 downto 8) <= DI_Reg;
			else
				A(15 downto 8) <= ACC;
			end if;
			A(7 downto 0) <= DI_Reg;
		when aSP =>
			A <= std_logic_vector(SP);
		when aBC =>
			if Mode = 3 and IORQ_i = '1' then
				-- Memory map I/O on GBZ80
				A(15 downto 8) <= (others => '1');
				A(7 downto 0) <= RegBusC(7 downto 0);
			else
				A <= RegBusC;
			end if;
		when aDE =>
			A <= RegBusC;
		when aZI =>
			if Inc_WZ = '1' then
				A <= std_logic_vector(unsigned(TmpAddr) + 1);
			else
				A(15 downto 8) <= DI_Reg;
				A(7 downto 0) <= TmpAddr(7 downto 0);
			end if;
		when others =>
			A <= std_logic_vector(PC);
		end case;
end if;

Save_ALU_r <= Save_ALU;
ALU_Op_r <= ALU_Op;

if I_CPL = '1' then
		-- CPL
		ACC <= not ACC;
		F(Flag_Y) <= not ACC(5);
		F(Flag_H) <= '1';
		F(Flag_X) <= not ACC(3);
		F(Flag_N) <= '1';
end if;
if I_CCF = '1' then
		-- CCF
		F(Flag_C) <= not F(Flag_C);
		F(Flag_Y) <= ACC(5);
		F(Flag_H) <= F(Flag_C);
		F(Flag_X) <= ACC(3);
		F(Flag_N) <= '0';
end if;
if I_SCF = '1' then
		-- SCF
		F(Flag_C) <= '1';
		F(Flag_Y) <= ACC(5);
		F(Flag_H) <= '0';
		F(Flag_X) <= ACC(3);
```

```
                                F(Flag_N) <= '0';
                        end if;
                end if;

                if TState = 2 and Wait_n = '1' then
                        if ISet = "01" and MCycle = "111" then
                                IR <= DInst;
                        end if;
                        if JumpE = '1' then
                                PC <= unsigned(signed(PC) + signed(DI_Reg));
                        elsif Inc_PC = '1' then
                                PC <= PC + 1;
                        end if;
                        if BTR_r = '1' then
                                PC <= PC - 2;
                        end if;
                        if RstP = '1' then
                                TmpAddr <= (others =>'0');
                                TmpAddr(5 downto 3) <= IR(5 downto 3);
                        end if;
                end if;
                if TState = 3 and MCycle = "110" then
                        TmpAddr <= std_logic_vector(signed(RegBusC) + signed(DI_Reg));
                end if;

                if (TState = 2 and Wait_n = '1') or (TState = 4 and MCycle = "001") then
                        if IncDec_16(2 downto 0) = "111" then
                                if IncDec_16(3) = '1' then
                                        SP <= SP - 1;
                                else
                                        SP <= SP + 1;
                                end if;
                        end if;
                end if;

                if LDSPHL = '1' then
                        SP <= unsigned(RegBusC);
                end if;
                if ExchangeAF = '1' then
                        Ap <= ACC;
                        ACC <= Ap;
                        Fp <= F;
                        F <= Fp;
                end if;
                if ExchangeRS = '1' then
                        Alternate <= not Alternate;
                end if;
        end if;

        if TState = 3 then
                if LDZ = '1' then
                        TmpAddr(7 downto 0) <= DI_Reg;
                end if;
                if LDW = '1' then
                        TmpAddr(15 downto 8) <= DI_Reg;
                end if;

                if Special_LD(2) = '1' then
                        case Special_LD(1 downto 0) is
                        when "00" =>
                                ACC <= I;
                                F(Flag_P) <= IntE_FF2;
                        when "01" =>
                                ACC <= std_logic_vector(R);
                                F(Flag_P) <= IntE_FF2;
                        when "10" =>
                                I <= ACC;
                        when others =>
                                R <= unsigned(ACC);
                        end case;
```

```
		end if;
end if;

if (I_DJNZ = '0' and Save_ALU_r = '1') or ALU_Op_r = "1001" then
		if Mode = 3 then
			F(6) <= F_Out(6);
			F(5) <= F_Out(5);
			F(7) <= F_Out(7);
			if PreserveC_r = '0' then
				F(4) <= F_Out(4);
			end if;
		else
			F(7 downto 1) <= F_Out(7 downto 1);
			if PreserveC_r = '0' then
				F(Flag_C) <= F_Out(0);
			end if;
		end if;
end if;
if T_Res = '1' and I_INRC = '1' then
		F(Flag_H) <= '0';
		F(Flag_N) <= '0';
		if DI_Reg(7 downto 0) = "00000000" then
			F(Flag_Z) <= '1';
		else
			F(Flag_Z) <= '0';
		end if;
		F(Flag_S) <= DI_Reg(7);
		F(Flag_P) <= not (DI_Reg(0) xor DI_Reg(1) xor DI_Reg(2) xor DI_Reg(3) xor
			DI_Reg(4) xor DI_Reg(5) xor DI_Reg(6) xor DI_Reg(7));
end if;

if TState = 1 and Auto_Wait_t1 = '0' then
		DO <= BusB;
		if I_RLD = '1' then
			DO(3 downto 0) <= BusA(3 downto 0);
			DO(7 downto 4) <= BusB(3 downto 0);
		end if;
		if I_RRD = '1' then
			DO(3 downto 0) <= BusB(7 downto 4);
			DO(7 downto 4) <= BusA(3 downto 0);
		end if;
end if;

if T_Res = '1' then
		Read_To_Reg_r(3 downto 0) <= Set_BusA_To;
		Read_To_Reg_r(4) <= Read_To_Reg;
		if Read_To_Acc = '1' then
			Read_To_Reg_r(3 downto 0) <= "0111";
			Read_To_Reg_r(4) <= '1';
		end if;
end if;

if TState = 1 and I_BT = '1' then
		F(Flag_X) <= ALU_Q(3);
		F(Flag_Y) <= ALU_Q(1);
		F(Flag_H) <= '0';
		F(Flag_N) <= '0';
end if;
if I_BC = '1' or I_BT = '1' then
		F(Flag_P) <= IncDecZ;
end if;

if (TState = 1 and Save_ALU_r = '0' and Auto_Wait_t1 = '0') or
		(Save_ALU_r = '1' and ALU_OP_r /= "0111") then
		case Read_To_Reg_r is
		when "10111" =>
			ACC <= Save_Mux;
		when "10110" =>
			DO <= Save_Mux;
		when "11000" =>
```

```
                                        SP(7 downto 0) <= unsigned(Save_Mux);
                                    when "11001" =>
                                        SP(15 downto 8) <= unsigned(Save_Mux);
                                    when "11011" =>
                                        F <= Save_Mux;
                                    when others =>
                                    end case;
                                end if;

                        end if;

                        end if;

                end process;

-------------------------------------------------------------------------
--
-- BC('), DE('), HL('), IX and IY
--
-------------------------------------------------------------------------
                process (CLK_n)
                begin
                        if CLK_n'event and CLK_n = '1' then
                                if ClkEn = '1' then
                                    -- Bus A / Write
                                    RegAddrA_r <= Alternate & Set_BusA_To(2 downto 1);
                                    if XY_Ind = '0' and XY_State /= "00" and Set_BusA_To(2 downto 1) = "10" then
                                        RegAddrA_r <= XY_State(1) & "11";
                                    end if;

                                    -- Bus B
                                    RegAddrB_r <= Alternate & Set_BusB_To(2 downto 1);
                                    if XY_Ind = '0' and XY_State /= "00" and Set_BusB_To(2 downto 1) = "10" then
                                        RegAddrB_r <= XY_State(1) & "11";
                                    end if;

                                    -- Address from register
                                    RegAddrC <= Alternate & Set_Addr_To(1 downto 0);
                                    -- Jump (HL), LD SP,HL
                                    if (JumpXY = '1' or LDSPHL = '1') then
                                        RegAddrC <= Alternate & "10";
                                    end if;
                                    if ((JumpXY = '1' or LDSPHL = '1') and XY_State /= "00") or (MCycle = "110") then
                                        RegAddrC <= XY_State(1) & "11";
                                    end if;

                                    if I_DJNZ = '1' and Save_ALU_r = '1' and Mode < 2 then
                                        IncDecZ <= F_Out(Flag_Z);
                                    end if;
                                    if (TState = 2 or (TState = 3 and MCycle = "001")) and IncDec_16(2 downto 0) = "100" then
                                        if ID16 = 0 then
                                            IncDecZ <= '0';
                                        else
                                            IncDecZ <= '1';
                                        end if;
                                    end if;

                                    RegBusA_r <= RegBusA;
                                end if;
                        end if;
                end process;

                RegAddrA <=
                                    -- 16 bit increment/decrement
                                    Alternate & IncDec_16(1 downto 0) when (TState = 2 or
                                        (TState = 3 and MCycle = "001" and IncDec_16(2) = '1')) and XY_State = "00" else
                                    XY_State(1) & "11" when (TState = 2 or
                                        (TState = 3 and MCycle = "001" and IncDec_16(2) = '1')) and IncDec_16(1 downto 0) = "10" else
                                    -- EX HL,DL
                                    Alternate & "10" when ExchangeDH = '1' and TState = 3 else
```

```
                        Alternate & "01" when ExchangeDH = '1' and TState = 4 else
                        -- Bus A / Write
                        RegAddrA_r;

RegAddrB <=
                        -- EX HL,DL
                        Alternate & "01" when ExchangeDH = '1' and TState = 3 else
                        -- Bus B
                        RegAddrB_r;

ID16 <= signed(RegBusA) - 1 when IncDec_16(3) = '1' else
                        signed(RegBusA) + 1;

process (Save_ALU_r, Auto_Wait_t1, ALU_OP_r, Read_To_Reg_r,
                        ExchangeDH, IncDec_16, MCycle, TState, Wait_n)
begin
        RegWEH <= '0';
        RegWEL <= '0';
        if (TState = 1 and Save_ALU_r = '0' and Auto_Wait_t1 = '0') or
                        (Save_ALU_r = '1' and ALU_OP_r /= "0111") then
                        case Read_To_Reg_r is
                        when "10000" | "10001" | "10010" | "10011" | "10100" | "10101" =>
                                RegWEH <= not Read_To_Reg_r(0);
                                RegWEL <= Read_To_Reg_r(0);
                        when others =>
                        end case;
        end if;

        if ExchangeDH = '1' and (TState = 3 or TState = 4) then
                        RegWEH <= '1';
                        RegWEL <= '1';
        end if;

        if IncDec_16(2) = '1' and ((TState = 2 and Wait_n = '1' and MCycle /= "001") or (TState = 3 and MCycle = "001")) then
                        case IncDec_16(1 downto 0) is
                        when "00" | "01" | "10" =>
                                RegWEH <= '1';
                                RegWEL <= '1';
                        when others =>
                        end case;
        end if;
end process;

process (Save_Mux, RegBusB, RegBusA_r, ID16,
                        ExchangeDH, IncDec_16, MCycle, TState, Wait_n)
begin
        RegDIH <= Save_Mux;
        RegDIL <= Save_Mux;

        if ExchangeDH = '1' and TState = 3 then
                        RegDIH <= RegBusB(15 downto 8);
                        RegDIL <= RegBusB(7 downto 0);
        end if;
        if ExchangeDH = '1' and TState = 4 then
                        RegDIH <= RegBusA_r(15 downto 8);
                        RegDIL <= RegBusA_r(7 downto 0);
        end if;

        if IncDec_16(2) = '1' and ((TState = 2 and MCycle /= "001") or (TState = 3 and MCycle = "001")) then
                        RegDIH <= std_logic_vector(ID16(15 downto 8));
                        RegDIL <= std_logic_vector(ID16(7 downto 0));
        end if;
end process;

Regs : T80_Reg
        port map(
                        Clk => CLK_n,
                        CEN => ClkEn,
                        WEH => RegWEH,
                        WEL => RegWEL,
```

```
                                AddrA => RegAddrA,
                                AddrB => RegAddrB,
                                AddrC => RegAddrC,
                                DIH => RegDIH,
                                DIL => RegDIL,
                                DOAH => RegBusA(15 downto 8),
                                DOAL => RegBusA(7 downto 0),
                                DOBH => RegBusB(15 downto 8),
                                DOBL => RegBusB(7 downto 0),
                                DOCH => RegBusC(15 downto 8),
                                DOCL => RegBusC(7 downto 0));

-------------------------------------------------------------------------
--
-- Buses
--
-------------------------------------------------------------------------
        process (CLK_n)
        begin
                if CLK_n'event and CLK_n = '1' then
                        if ClkEn = '1' then
                        case Set_BusB_To is
                        when "0111" =>
                                BusB <= ACC;
                        when "0000" | "0001" | "0010" | "0011" | "0100" | "0101" =>
                                if Set_BusB_To(0) = '1' then
                                        BusB <= RegBusB(7 downto 0);
                                else
                                        BusB <= RegBusB(15 downto 8);
                                end if;
                        when "0110" =>
                                BusB <= DI_Reg;
                        when "1000" =>
                                BusB <= std_logic_vector(SP(7 downto 0));
                        when "1001" =>
                                BusB <= std_logic_vector(SP(15 downto 8));
                        when "1010" =>
                                BusB <= "00000001";
                        when "1011" =>
                                BusB <= F;
                        when "1100" =>
                                BusB <= std_logic_vector(PC(7 downto 0));
                        when "1101" =>
                                BusB <= std_logic_vector(PC(15 downto 8));
                        when "1110" =>
                                BusB <= "00000000";
                        when others =>
                                BusB <= "--------";
                        end case;

                        case Set_BusA_To is
                        when "0111" =>
                                BusA <= ACC;
                        when "0000" | "0001" | "0010" | "0011" | "0100" | "0101" =>
                                if Set_BusA_To(0) = '1' then
                                        BusA <= RegBusA(7 downto 0);
                                else
                                        BusA <= RegBusA(15 downto 8);
                                end if;
                        when "0110" =>
                                BusA <= DI_Reg;
                        when "1000" =>
                                BusA <= std_logic_vector(SP(7 downto 0));
                        when "1001" =>
                                BusA <= std_logic_vector(SP(15 downto 8));
                        when "1010" =>
                                BusA <= "00000000";
                        when others =>
                                BusB <= "--------";
                        end case;
```

```
                        end if;
                end if;
        end process;

-------------------------------------------------------------------------
--
-- Generate external control signals
--
-------------------------------------------------------------------------
        process (RESET_n,CLK_n)
        begin
                if RESET_n = '0' then
                        RFSH_n <= '1';
                elsif CLK_n'event and CLK_n = '1' then
                        if CEN = '1' then
                        if MCycle = "001" and ((TState = 2  and Wait_n = '1') or TState = 3) then
                                RFSH_n <= '0';
                        else
                                RFSH_n <= '1';
                        end if;
                        end if;
                end if;
        end process;

        MC <= std_logic_vector(MCycle);
        TS <= std_logic_vector(TState);
        DI_Reg <= DI;
        HALT_n <= not Halt_FF;
        BUSAK_n <= not BusAck;
        IntCycle_n <= not IntCycle;
        IntE <= IntE_FF1;
        IORQ <= IORQ_i;
        Stop <= I_DJNZ;

-------------------------------------------------------------------------
--
-- Syncronise inputs
--
-------------------------------------------------------------------------
        process (RESET_n, CLK_n)
                variable OldNMI_n : std_logic;
        begin
                if RESET_n = '0' then
                        BusReq_s <= '0';
                        INT_s <= '0';
                        NMI_s <= '0';
                        OldNMI_n := '0';
                elsif CLK_n'event and CLK_n = '1' then
                        if CEN = '1' then
                        BusReq_s <= not BUSRQ_n;
                        INT_s <= not INT_n;
                        if NMICycle = '1' then
                                NMI_s <= '0';
                        elsif NMI_n = '0' and OldNMI_n = '1' then
                                NMI_s <= '1';
                        end if;
                        OldNMI_n := NMI_n;
                        end if;
                end if;
        end process;

-------------------------------------------------------------------------
--
-- Main state machine
--
-------------------------------------------------------------------------
        process (RESET_n, CLK_n)
        begin
                if RESET_n = '0' then
                        MCycle <= "001";
```

```
		TState <= "000";
		Pre_XY_F_M <= "000";
		Halt_FF <= '0';
		BusAck <= '0';
		NMICycle <= '0';
		IntCycle <= '0';
		IntE_FF1 <= '0';
		IntE_FF2 <= '0';
		No_BTR <= '0';
		Auto_Wait_t1 <= '0';
		Auto_Wait_t2 <= '0';
		M1_n <= '1';
	elsif CLK_n'event and CLK_n = '1' then
		if CEN = '1' then
		if T_Res = '1' then
			Auto_Wait_t1 <= '0';
		else
			Auto_Wait_t1 <= Auto_Wait or IORQ_i;
		end if;
		Auto_Wait_t2 <= Auto_Wait_t1;
		No_BTR <= (I_BT and (not IR(4) or not F(Flag_P))) or
				(I_BC and (not IR(4) or F(Flag_Z) or not F(Flag_P))) or
				(I_BTR and (not IR(4) or F(Flag_Z)));
		if TState = 2 then
			if SetEI = '1' then
				IntE_FF1 <= '1';
				IntE_FF2 <= '1';
			end if;
			if I_RETN = '1' then
				IntE_FF1 <= IntE_FF2;
			end if;
		end if;
		if TState = 3 then
			if SetDI = '1' then
				IntE_FF1 <= '0';
				IntE_FF2 <= '0';
			end if;
		end if;
		if IntCycle = '1' or NMICycle = '1' then
			Halt_FF <= '0';
		end if;
		if MCycle = "001" and TState = 2 and Wait_n = '1' then
			M1_n <= '1';
		end if;
		if BusReq_s = '1' and BusAck = '1' then
		else
			BusAck <= '0';
			if TState = 2 and Wait_n = '0' then
			elsif T_Res = '1' then
				if Halt = '1' then
					Halt_FF <= '1';
				end if;
				if BusReq_s = '1' then
					BusAck <= '1';
				else
					TState <= "001";
					if NextIs_XY_Fetch = '1' then
						MCycle <= "110";
						Pre_XY_F_M <= MCycle;
						if IR = "00110110" and Mode = 0 then
							Pre_XY_F_M <= "010";
						end if;
					elsif (MCycle = "111") or
						(MCycle = "110" and Mode = 1 and ISet /= "01") then
						MCycle <= std_logic_vector(unsigned(Pre_XY_F_M) + 1);
					elsif (MCycle = MCycles) or
						No_BTR = '1' or
						(MCycle = "010" and I_DJNZ = '1' and IncDecZ = '1') then
						M1_n <= '0';
						MCycle <= "001";
```

```
                                                IntCycle <= '0';
                                                NMICycle <= '0';
                                                if NMI_s = '1' and Prefix = "00" then
                                                        NMICycle <= '1';
                                                        IntE_FF1 <= '0';
                                                elsif (IntE_FF1 = '1' and INT_s = '1') and Prefix = "00" and SetEI = '0' then
                                                        IntCycle <= '1';
                                                        IntE_FF1 <= '0';
                                                        IntE_FF2 <= '0';
                                                end if;
                                        else
                                                MCycle <= std_logic_vector(unsigned(MCycle) + 1);
                                        end if;
                                end if;
                        else
                                if (Auto_Wait = '1' and Auto_Wait_t2 = '0') nor
                                        (IOWait = 1 and IORQ_i = '1' and Auto_Wait_t1 = '0') then
                                        TState <= TState + 1;
                                end if;
                        end if;
                end if;
                if TState = 0 then
                        M1_n <= '0';
                end if;
                end if;
        end if;
    end process;

    process (IntCycle, NMICycle, MCycle)
    begin
        Auto_Wait <= '0';
        if IntCycle = '1' or NMICycle = '1' then
                if MCycle = "001" then
                        Auto_Wait <= '1';
                end if;
        end if;
    end process;

end;
```

## 8.2 T80_MCODE.vhd

```
--
-- Z80 compatible microprocessor core
--
-- Version : 0242
--
-- Copyright (c) 2001-2002 Daniel Wallner (jesus@opencores.org)
--
-- All rights reserved
--
-- Redistribution and use in source and synthezised forms, with or without
-- modification, are permitted provided that the following conditions are met:
--
-- Redistributions of source code must retain the above copyright notice,
-- this list of conditions and the following disclaimer.
--
-- Redistributions in synthesized form must reproduce the above copyright
-- notice, this list of conditions and the following disclaimer in the
-- documentation and/or other materials provided with the distribution.
--
-- Neither the name of the author nor the names of other contributors may
-- be used to endorse or promote products derived from this software without
-- specific prior written permission.
--
-- THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
-- AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO,
-- THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
-- PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE
-- LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
-- CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
-- SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
-- INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
-- CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
-- ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
-- POSSIBILITY OF SUCH DAMAGE.
--
-- Please report bugs to the author, but before you do so, please
-- make sure that this is not a derivative work and that
-- you have the latest version of this file.
--
-- The latest version of this file can be found at:
--      http://www.opencores.org/cvsweb.shtml/t80/
--
-- Limitations :
--
-- File history :
--
--      0208 : First complete release
--
--      0211 : Fixed IM 1
--
--      0214 : Fixed mostly flags, only the block instructions now fail the zex regression test
--
--      0235 : Added IM 2 fix by Mike Johnson
--
--      0238 : Added NoRead signal
--
--      0238b: Fixed instruction timing for POP and DJNZ
--
--      0240 : Added (IX/IY+d) states, removed op-codes from mode 2 and added all remaining mode 3 op-codes
--
--      0242 : Fixed I/O instruction timing, cleanup
--

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;
```

```
entity T80_MCode is
	generic(
		Mode : integer := 0;
		Flag_C : integer := 0;
		Flag_N : integer := 1;
		Flag_P : integer := 2;
		Flag_X : integer := 3;
		Flag_H : integer := 4;
		Flag_Y : integer := 5;
		Flag_Z : integer := 6;
		Flag_S : integer := 7
	);
	port(
		IR			: in std_logic_vector(7 downto 0);
		ISet			: in std_logic_vector(1 downto 0);
		MCycle			: in std_logic_vector(2 downto 0);
		F			: in std_logic_vector(7 downto 0);
		NMICycle		: in std_logic;
		IntCycle		: in std_logic;
		MCycles			: out std_logic_vector(2 downto 0);
		TStates			: out std_logic_vector(2 downto 0);
		Prefix			: out std_logic_vector(1 downto 0); -- None,BC,ED,DD/FD
		Inc_PC			: out std_logic;
		Inc_WZ			: out std_logic;
		IncDec_16		: out std_logic_vector(3 downto 0); -- BC,DE,HL,SP   0 is inc
		Read_To_Reg		: out std_logic;
		Read_To_Acc		: out std_logic;
		Set_BusA_To		: out std_logic_vector(3 downto 0); -- B,C,D,E,H,L,DI/DB,A,SP(L),SP(M),0,F
		Set_BusB_To		: out std_logic_vector(3 downto 0); -- B,C,D,E,H,L,DI,A,SP(L),SP(M),1,F,PC(L),PC(M),0
		ALU_Op			: out std_logic_vector(3 downto 0);
				-- ADD, ADC, SUB, SBC, AND, XOR, OR, CP, ROT, BIT, SET, RES, DAA, RLD, RRD, None
		Save_ALU		: out std_logic;
		PreserveC		: out std_logic;
		Arith16			: out std_logic;
		Set_Addr_To		: out std_logic_vector(2 downto 0); -- aNone,aXY,aIOA,aSP,aBC,aDE,aZI
		IORQ			: out std_logic;
		Jump			: out std_logic;
		JumpE			: out std_logic;
		JumpXY			: out std_logic;
		Call			: out std_logic;
		RstP			: out std_logic;
		LDZ			: out std_logic;
		LDW			: out std_logic;
		LDSPHL			: out std_logic;
		Special_LD		: out std_logic_vector(2 downto 0); -- A,I;A,R;I,A;R,A;None
		ExchangeDH		: out std_logic;
		ExchangeRp		: out std_logic;
		ExchangeAF		: out std_logic;
		ExchangeRS		: out std_logic;
		I_DJNZ			: out std_logic;
		I_CPL			: out std_logic;
		I_CCF			: out std_logic;
		I_SCF			: out std_logic;
		I_RETN			: out std_logic;
		I_BT			: out std_logic;
		I_BC			: out std_logic;
		I_BTR			: out std_logic;
		I_RLD			: out std_logic;
		I_RRD			: out std_logic;
		I_INRC			: out std_logic;
		SetDI			: out std_logic;
		SetEI			: out std_logic;
		IMode			: out std_logic_vector(1 downto 0);
		Halt			: out std_logic;
		NoRead			: out std_logic;
		Write_80		: out std_logic
	);
end T80_MCode;
```

```
architecture rtl of T80_MCode is

	constant aNone	: std_logic_vector(2 downto 0) := "111";
	constant aBC	: std_logic_vector(2 downto 0) := "000";
	constant aDE	: std_logic_vector(2 downto 0) := "001";
	constant aXY	: std_logic_vector(2 downto 0) := "010";
	constant aIOA	: std_logic_vector(2 downto 0) := "100";
	constant aSP	: std_logic_vector(2 downto 0) := "101";
	constant aZI	: std_logic_vector(2 downto 0) := "110";
--	constant aNone	: std_logic_vector(2 downto 0) := "000";
--	constant aXY	: std_logic_vector(2 downto 0) := "001";
--	constant aIOA	: std_logic_vector(2 downto 0) := "010";
--	constant aSP	: std_logic_vector(2 downto 0) := "011";
--	constant aBC	: std_logic_vector(2 downto 0) := "100";
--	constant aDE	: std_logic_vector(2 downto 0) := "101";
--	constant aZI	: std_logic_vector(2 downto 0) := "110";

	function is_cc_true(
		F : std_logic_vector(7 downto 0);
		cc : bit_vector(2 downto 0)
		) return boolean is
	begin
		if Mode = 3 then
			case cc is
			when "000" => return F(7) = '0'; -- NZ
			when "001" => return F(7) = '1'; -- Z
			when "010" => return F(4) = '0'; -- NC
			when "011" => return F(4) = '1'; -- C
			when "100" => return false;
			when "101" => return false;
			when "110" => return false;
			when "111" => return false;
			end case;
		else
			case cc is
			when "000" => return F(6) = '0'; -- NZ
			when "001" => return F(6) = '1'; -- Z
			when "010" => return F(0) = '0'; -- NC
			when "011" => return F(0) = '1'; -- C
			when "100" => return F(2) = '0'; -- PO
			when "101" => return F(2) = '1'; -- PE
			when "110" => return F(7) = '0'; -- P
			when "111" => return F(7) = '1'; -- M
			end case;
		end if;
	end;

begin

	process (IR, ISet, MCycle, F, NMICycle, IntCycle)
		variable DDD : std_logic_vector(2 downto 0);
		variable SSS : std_logic_vector(2 downto 0);
		variable DPair : std_logic_vector(1 downto 0);
		variable IRB : bit_vector(7 downto 0);
	begin
		DDD := IR(5 downto 3);
		SSS := IR(2 downto 0);
		DPair := IR(5 downto 4);
		IRB := to_bitvector(IR);

		MCycles <= "001";
		if MCycle = "001" then
			TStates <= "100";
		else
			TStates <= "011";
		end if;
		Prefix <= "00";
		Inc_PC <= '0';
		Inc_WZ <= '0';
		IncDec_16 <= "0000";
```

```
                        Read_To_Acc <= '0';
                        Read_To_Reg <= '0';
                        Set_BusB_To <= "0000";
                        Set_BusA_To <= "0000";
                        ALU_Op <= "0" & IR(5 downto 3);
                        Save_ALU <= '0';
                        PreserveC <= '0';
                        Arith16 <= '0';
                        IORQ <= '0';
                        Set_Addr_To <= aNone;
                        Jump <= '0';
                        JumpE <= '0';
                        JumpXY <= '0';
                        Call <= '0';
                        RstP <= '0';
                        LDZ <= '0';
                        LDW <= '0';
                        LDSPHL <= '0';
                        Special_LD <= "000";
                        ExchangeDH <= '0';
                        ExchangeRp <= '0';
                        ExchangeAF <= '0';
                        ExchangeRS <= '0';
                        I_DJNZ <= '0';
                        I_CPL <= '0';
                        I_CCF <= '0';
                        I_SCF <= '0';
                        I_RETN <= '0';
                        I_BT <= '0';
                        I_BC <= '0';
                        I_BTR <= '0';
                        I_RLD <= '0';
                        I_RRD <= '0';
                        I_INRC <= '0';
                        SetDI <= '0';
                        SetEI <= '0';
                        IMode <= "11";
                        Halt <= '0';
                        NoRead <= '0';
                        Write_80 <= '0';

                        case ISet is
                        when "00" =>

-------------------------------------------------------------------------------
--
--              Unprefixed instructions
--
-------------------------------------------------------------------------------

                        case IRB is
-- 8 BIT LOAD GROUP
                        when "01000000"|"01000001"|"01000010"|"01000011"|"01000100"|"01000101"|"01000111"
                                |"01001000"|"01001001"|"01001010"|"01001011"|"01001100"|"01001101"|"01001111"
                                |"01010000"|"01010001"|"01010010"|"01010011"|"01010100"|"01010101"|"01010111"
                                |"01011000"|"01011001"|"01011010"|"01011011"|"01011100"|"01011101"|"01011111"
                                |"01100000"|"01100001"|"01100010"|"01100011"|"01100100"|"01100101"|"01100111"
                                |"01101000"|"01101001"|"01101010"|"01101011"|"01101100"|"01101101"|"01101111"
                                |"01111000"|"01111001"|"01111010"|"01111011"|"01111100"|"01111101"|"01111111" =>
                                -- LD r,r'
                                Set_BusB_To(2 downto 0) <= SSS;
                                ExchangeRp <= '1';
                                Set_BusA_To(2 downto 0) <= DDD;
                                Read_To_Reg <= '1';
                        when "00000110"|"00001110"|"00010110"|"00011110"|"00100110"|"00101110"|"00111110" =>
                                -- LD r,n
                                MCycles <= "010";
                                case to_integer(unsigned(MCycle)) is
                                when 2 =>
                                        Inc_PC <= '1';
```

```
                        Set_BusA_To(2 downto 0) <= DDD;
                        Read_To_Reg <= '1';
                when others => null;
                end case;
when "01000110"|"01001110"|"01010110"|"01011110"|"01100110"|"01101110"|"01111110" =>
                -- LD r,(HL)
                MCycles <= "010";
                case to_integer(unsigned(MCycle)) is
                when 1 =>
                        Set_Addr_To <= aXY;
                when 2 =>
                        Set_BusA_To(2 downto 0) <= DDD;
                        Read_To_Reg <= '1';
                when others => null;
                end case;
when "01110000"|"01110001"|"01110010"|"01110011"|"01110100"|"01110101"|"01110111" =>
                -- LD (HL),r
                MCycles <= "010";
                case to_integer(unsigned(MCycle)) is
                when 1 =>
                        Set_Addr_To <= aXY;
                        Set_BusB_To(2 downto 0) <= SSS;
                        Set_BusB_To(3) <= '0';
                when 2 =>
                        Write_80 <= '1';
                when others => null;
                end case;
when "00110110" =>
                -- LD (HL),n
                MCycles <= "011";
                case to_integer(unsigned(MCycle)) is
                when 2 =>
                        Inc_PC <= '1';
                        Set_Addr_To <= aXY;
                        Set_BusB_To(2 downto 0) <= SSS;
                        Set_BusB_To(3) <= '0';
                when 3 =>
                        Write_80 <= '1';
                when others => null;
                end case;
when "00001010" =>
                -- LD A,(BC)
                MCycles <= "010";
                case to_integer(unsigned(MCycle)) is
                when 1 =>
                        Set_Addr_To <= aBC;
                when 2 =>
                        Read_To_Acc <= '1';
                when others => null;
                end case;
when "00011010" =>
                -- LD A,(DE)
                MCycles <= "010";
                case to_integer(unsigned(MCycle)) is
                when 1 =>
                        Set_Addr_To <= aDE;
                when 2 =>
                        Read_To_Acc <= '1';
                when others => null;
                end case;
when "00111010" =>
                if Mode = 3 then
                        -- LDD A,(HL)
                        MCycles <= "010";
                        case to_integer(unsigned(MCycle)) is
                        when 1 =>
                                Set_Addr_To <= aXY;
                        when 2 =>
                                Read_To_Acc <= '1';
                                IncDec_16 <= "1110";
```

```
                        when others => null;
                        end case;
                else
                        -- LD A,(nn)
                        MCycles <= "100";
                        case to_integer(unsigned(MCycle)) is
                        when 2 =>
                                Inc_PC <= '1';
                                LDZ <= '1';
                        when 3 =>
                                Set_Addr_To <= aZI;
                                Inc_PC <= '1';
                        when 4 =>
                                Read_To_Acc <= '1';
                        when others => null;
                        end case;
                end if;
        when "00000010" =>
                -- LD (BC),A
                MCycles <= "010";
                case to_integer(unsigned(MCycle)) is
                when 1 =>
                        Set_Addr_To <= aBC;
                        Set_BusB_To <= "0111";
                when 2 =>
                        Write_80 <= '1';
                when others => null;
                end case;
        when "00010010" =>
                -- LD (DE),A
                MCycles <= "010";
                case to_integer(unsigned(MCycle)) is
                when 1 =>
                        Set_Addr_To <= aDE;
                        Set_BusB_To <= "0111";
                when 2 =>
                        Write_80 <= '1';
                when others => null;
                end case;
        when "00110010" =>
                if Mode = 3 then
                        -- LDD (HL),A
                        MCycles <= "010";
                        case to_integer(unsigned(MCycle)) is
                        when 1 =>
                                Set_Addr_To <= aXY;
                                Set_BusB_To <= "0111";
                        when 2 =>
                                Write_80 <= '1';
                                IncDec_16 <= "1110";
                        when others => null;
                        end case;
                else
                        -- LD (nn),A
                        MCycles <= "100";
                        case to_integer(unsigned(MCycle)) is
                        when 2 =>
                                Inc_PC <= '1';
                                LDZ <= '1';
                        when 3 =>
                                Set_Addr_To <= aZI;
                                Inc_PC <= '1';
                                Set_BusB_To <= "0111";
                        when 4 =>
                                Write_80 <= '1';
                        when others => null;
                        end case;
                end if;

-- 16 BIT LOAD GROUP
```

```
when "00000001"|"00010001"|"00100001"|"00110001" =>
		-- LD dd,nn
		MCycles <= "011";
		case to_integer(unsigned(MCycle)) is
		when 2 =>
			Inc_PC <= '1';
			Read_To_Reg <= '1';
			if DPAIR = "11" then
				Set_BusA_To(3 downto 0) <= "1000";
			else
				Set_BusA_To(2 downto 1) <= DPAIR;
				Set_BusA_To(0) <= '1';
			end if;
		when 3 =>
			Inc_PC <= '1';
			Read_To_Reg <= '1';
			if DPAIR = "11" then
				Set_BusA_To(3 downto 0) <= "1001";
			else
				Set_BusA_To(2 downto 1) <= DPAIR;
				Set_BusA_To(0) <= '0';
			end if;
		when others => null;
		end case;
when "00101010" =>
		if Mode = 3 then
			-- LDI A,(HL)
			MCycles <= "010";
			case to_integer(unsigned(MCycle)) is
			when 1 =>
				Set_Addr_To <= aXY;
			when 2 =>
				Read_To_Acc <= '1';
				IncDec_16 <= "0110";
			when others => null;
			end case;
		else
			-- LD HL,(nn)
			MCycles <= "101";
			case to_integer(unsigned(MCycle)) is
			when 2 =>
				Inc_PC <= '1';
				LDZ <= '1';
			when 3 =>
				Set_Addr_To <= aZI;
				Inc_PC <= '1';
				LDW <= '1';
			when 4 =>
				Set_BusA_To(2 downto 0) <= "101"; -- L
				Read_To_Reg <= '1';
				Inc_WZ <= '1';
				Set_Addr_To <= aZI;
			when 5 =>
				Set_BusA_To(2 downto 0) <= "100"; -- H
				Read_To_Reg <= '1';
			when others => null;
			end case;
		end if;
when "00100010" =>
		if Mode = 3 then
			-- LDI (HL),A
			MCycles <= "010";
			case to_integer(unsigned(MCycle)) is
			when 1 =>
				Set_Addr_To <= aXY;
				Set_BusB_To <= "0111";
			when 2 =>
				Write_80 <= '1';
				IncDec_16 <= "0110";
			when others => null;
```

```
                end case;
        else
                -- LD (nn),HL
                MCycles <= "101";
                case to_integer(unsigned(MCycle)) is
                when 2 =>
                        Inc_PC <= '1';
                        LDZ <= '1';
                when 3 =>
                        Set_Addr_To <= aZI;
                        Inc_PC <= '1';
                        LDW <= '1';
                        Set_BusB_To <= "0101"; -- L
                when 4 =>
                        Inc_WZ <= '1';
                        Set_Addr_To <= aZI;
                        Write_80 <= '1';
                        Set_BusB_To <= "0100"; -- H
                when 5 =>
                        Write_80 <= '1';
                when others => null;
                end case;
        end if;
when "11111001" =>
        -- LD SP,HL
        TStates <= "110";
        LDSPHL <= '1';
when "11000101"|"11010101"|"11100101"|"11110101" =>
        -- PUSH qq
        MCycles <= "011";
        case to_integer(unsigned(MCycle)) is
        when 1 =>
                TStates <= "101";
                IncDec_16 <= "1111";
                Set_Addr_TO <= aSP;
                if DPAIR = "11" then
                        Set_BusB_To <= "0111";
                else
                        Set_BusB_To(2 downto 1) <= DPAIR;
                        Set_BusB_To(0) <= '0';
                        Set_BusB_To(3) <= '0';
                end if;
        when 2 =>
                IncDec_16 <= "1111";
                Set_Addr_To <= aSP;
                if DPAIR = "11" then
                        Set_BusB_To <= "1011";
                else
                        Set_BusB_To(2 downto 1) <= DPAIR;
                        Set_BusB_To(0) <= '1';
                        Set_BusB_To(3) <= '0';
                end if;
                Write_80 <= '1';
        when 3 =>
                Write_80 <= '1';
        when others => null;
        end case;
when "11000001"|"11010001"|"11100001"|"11110001" =>
        -- POP qq
        MCycles <= "011";
        case to_integer(unsigned(MCycle)) is
        when 1 =>
                Set_Addr_To <= aSP;
        when 2 =>
                IncDec_16 <= "0111";
                Set_Addr_To <= aSP;
                Read_To_Reg <= '1';
                if DPAIR = "11" then
                        Set_BusA_To(3 downto 0) <= "1011";
                else
```

```
                                        Set_BusA_To(2 downto 1) <= DPAIR;
                                        Set_BusA_To(0) <= '1';
                                end if;
                        when 3 =>
                                IncDec_16 <= "0111";
                                Read_To_Reg <= '1';
                                if DPAIR = "11" then
                                        Set_BusA_To(3 downto 0) <= "0111";
                                else
                                        Set_BusA_To(2 downto 1) <= DPAIR;
                                        Set_BusA_To(0) <= '0';
                                end if;
                        when others => null;
                        end case;

-- EXCHANGE, BLOCK TRANSFER AND SEARCH GROUP
                when "11101011" =>
                        if Mode /= 3 then
                                -- EX DE,HL
                                ExchangeDH <= '1';
                        end if;
                when "00001000" =>
                        if Mode = 3 then
                                -- LD (nn),SP
                                MCycles <= "101";
                                case to_integer(unsigned(MCycle)) is
                                when 2 =>
                                        Inc_PC <= '1';
                                        LDZ <= '1';
                                when 3 =>
                                        Set_Addr_To <= aZI;
                                        Inc_PC <= '1';
                                        LDW <= '1';
                                        Set_BusB_To <= "1000";
                                when 4 =>
                                        Inc_WZ <= '1';
                                        Set_Addr_To <= aZI;
                                        Write_80 <= '1';
                                        Set_BusB_To <= "1001";
                                when 5 =>
                                        Write_80 <= '1';
                                when others => null;
                                end case;
                        elsif Mode < 2 then
                                -- EX AF,AF'
                                ExchangeAF <= '1';
                        end if;
                when "11011001" =>
                        if Mode = 3 then
                                -- RETI
                                MCycles <= "011";
                                case to_integer(unsigned(MCycle)) is
                                when 1 =>
                                        Set_Addr_TO <= aSP;
                                when 2 =>
                                        IncDec_16 <= "0111";
                                        Set_Addr_To <= aSP;
                                        LDZ <= '1';
                                when 3 =>
                                        Jump <= '1';
                                        IncDec_16 <= "0111";
                                        I_RETN <= '1';
                                        SetEI <= '1';
                                when others => null;
                                end case;
                        elsif Mode < 2 then
                                -- EXX
                                ExchangeRS <= '1';
                        end if;
                when "11100011" =>
```

```
                        if Mode /= 3 then
                                -- EX (SP),HL
                                MCycles <= "101";
                                case to_integer(unsigned(MCycle)) is
                                when 1 =>
                                        Set_Addr_To <= aSP;
                                when 2 =>
                                        Read_To_Reg <= '1';
                                        Set_BusA_To <= "0101";
                                        Set_BusB_To <= "0101";
                                        Set_Addr_To <= aSP;
                                when 3 =>
                                        IncDec_16 <= "0111";
                                        Set_Addr_To <= aSP;
                                        TStates <= "100";
                                        Write_80 <= '1';
                                when 4 =>
                                        Read_To_Reg <= '1';
                                        Set_BusA_To <= "0100";
                                        Set_BusB_To <= "0100";
                                        Set_Addr_To <= aSP;
                                when 5 =>
                                        IncDec_16 <= "1111";
                                        TStates <= "101";
                                        Write_80 <= '1';
                                when others => null;
                                end case;
                        end if;

-- 8 BIT ARITHMETIC AND LOGICAL GROUP
                when "10000000"|"10000001"|"10000010"|"10000011"|"10000100"|"10000101"|"10000111"
                        |"10001000"|"10001001"|"10001010"|"10001011"|"10001100"|"10001101"|"10001111"
                        |"10010000"|"10010001"|"10010010"|"10010011"|"10010100"|"10010101"|"10010111"
                        |"10011000"|"10011001"|"10011010"|"10011011"|"10011100"|"10011101"|"10011111"
                        |"10100000"|"10100001"|"10100010"|"10100011"|"10100100"|"10100101"|"10100111"
                        |"10101000"|"10101001"|"10101010"|"10101011"|"10101100"|"10101101"|"10101111"
                        |"10110000"|"10110001"|"10110010"|"10110011"|"10110100"|"10110101"|"10110111"
                        |"10111000"|"10111001"|"10111010"|"10111011"|"10111100"|"10111101"|"10111111" =>
                        -- ADD A,r
                        -- ADC A,r
                        -- SUB A,r
                        -- SBC A,r
                        -- AND A,r
                        -- OR A,r
                        -- XOR A,r
                        -- CP A,r
                        Set_BusB_To(2 downto 0) <= SSS;
                        Set_BusA_To(2 downto 0) <= "111";
                        Read_To_Reg <= '1';
                        Save_ALU <= '1';
                when "10000110"|"10001110"|"10010110"|"10011110"|"10100110"|"10101110"|"10110110"|"10111110" =>
                        -- ADD A,(HL)
                        -- ADC A,(HL)
                        -- SUB A,(HL)
                        -- SBC A,(HL)
                        -- AND A,(HL)
                        -- OR A,(HL)
                        -- XOR A,(HL)
                        -- CP A,(HL)
                        MCycles <= "010";
                        case to_integer(unsigned(MCycle)) is
                        when 1 =>
                                Set_Addr_To <= aXY;
                        when 2 =>
                                Read_To_Reg <= '1';
                                Save_ALU <= '1';
                                Set_BusB_To(2 downto 0) <= SSS;
                                Set_BusA_To(2 downto 0) <= "111";
                        when others => null;
                        end case;
```

```
when "11000110"|"11001110"|"11010110"|"11011110"|"11100110"|"11101110"|"11110110"|"11111110" =>
		-- ADD A,n
		-- ADC A,n
		-- SUB A,n
		-- SBC A,n
		-- AND A,n
		-- OR A,n
		-- XOR A,n
		-- CP A,n
		MCycles <= "010";
		if MCycle = "010" then
			Inc_PC <= '1';
			Read_To_Reg <= '1';
			Save_ALU <= '1';
			Set_BusB_To(2 downto 0) <= SSS;
			Set_BusA_To(2 downto 0) <= "111";
		end if;
when "00000100"|"00001100"|"00010100"|"00011100"|"00100100"|"00101100"|"00111100" =>
		-- INC r
		Set_BusB_To <= "1010";
		Set_BusA_To(2 downto 0) <= DDD;
		Read_To_Reg <= '1';
		Save_ALU <= '1';
		PreserveC <= '1';
		ALU_Op <= "0000";
when "00110100" =>
		-- INC (HL)
		MCycles <= "011";
		case to_integer(unsigned(MCycle)) is
		when 1 =>
			Set_Addr_To <= aXY;
		when 2 =>
			TStates <= "100";
			Set_Addr_To <= aXY;
			Read_To_Reg <= '1';
			Save_ALU <= '1';
			PreserveC <= '1';
			ALU_Op <= "0000";
			Set_BusB_To <= "1010";
			Set_BusA_To(2 downto 0) <= DDD;
		when 3 =>
			Write_80 <= '1';
		when others => null;
		end case;
when "00000101"|"00001101"|"00010101"|"00011101"|"00100101"|"00101101"|"00111101" =>
		-- DEC r
		Set_BusB_To <= "1010";
		Set_BusA_To(2 downto 0) <= DDD;
		Read_To_Reg <= '1';
		Save_ALU <= '1';
		PreserveC <= '1';
		ALU_Op <= "0010";
when "00110101" =>
		-- DEC (HL)
		MCycles <= "011";
		case to_integer(unsigned(MCycle)) is
		when 1 =>
			Set_Addr_To <= aXY;
		when 2 =>
			TStates <= "100";
			Set_Addr_To <= aXY;
			ALU_Op <= "0010";
			Read_To_Reg <= '1';
			Save_ALU <= '1';
			PreserveC <= '1';
			Set_BusB_To <= "1010";
			Set_BusA_To(2 downto 0) <= DDD;
		when 3 =>
			Write_80 <= '1';
		when others => null;
```

```
                end case;

-- GENERAL PURPOSE ARITHMETIC AND CPU CONTROL GROUPS
            when "00100111" =>
                -- DAA
                Set_BusA_To(2 downto 0) <= "111";
                Read_To_Reg <= '1';
                ALU_Op <= "1100";
                Save_ALU <= '1';
            when "00101111" =>
                -- CPL
                I_CPL <= '1';
            when "00111111" =>
                -- CCF
                I_CCF <= '1';
            when "00110111" =>
                -- SCF
                I_SCF <= '1';
            when "00000000" =>
                if NMICycle = '1' then
                    -- NMI
                    MCycles <= "011";
                    case to_integer(unsigned(MCycle)) is
                    when 1 =>
                        TStates <= "101";
                        IncDec_16 <= "1111";
                        Set_Addr_To <= aSP;
                        Set_BusB_To <= "1101";
                    when 2 =>
                        TStates <= "100";
                        Write_80 <= '1';
                        IncDec_16 <= "1111";
                        Set_Addr_To <= aSP;
                        Set_BusB_To <= "1100";
                    when 3 =>
                        TStates <= "100";
                        Write_80 <= '1';
                    when others => null;
                    end case;
                elsif IntCycle = '1' then
                    -- INT (IM 2)
                    MCycles <= "101";
                    case to_integer(unsigned(MCycle)) is
                    when 1 =>
                        LDZ <= '1';
                        TStates <= "101";
                        IncDec_16 <= "1111";
                        Set_Addr_To <= aSP;
                        Set_BusB_To <= "1101";
                    when 2 =>
                        TStates <= "100";
                        Write_80 <= '1';
                        IncDec_16 <= "1111";
                        Set_Addr_To <= aSP;
                        Set_BusB_To <= "1100";
                    when 3 =>
                        TStates <= "100";
                        Write_80 <= '1';
                    when 4 =>
                        Inc_PC <= '1';
                        LDZ <= '1';
                    when 5 =>
                        Jump <= '1';
                    when others => null;
                    end case;
                else
                    -- NOP
                end if;
            when "01110110" =>
                -- HALT
```

```
                        Halt <= '1';
                when "11110011" =>
                        -- DI
                        SetDI <= '1';
                when "11111011" =>
                        -- EI
                        SetEI <= '1';

-- 16 BIT ARITHMETIC GROUP
                when "00001001"|"00011001"|"00101001"|"00111001" =>
                        -- ADD HL,ss
                        MCycles <= "011";
                        case to_integer(unsigned(MCycle)) is
                        when 2 =>
                                NoRead <= '1';
                                ALU_Op <= "0000";
                                Read_To_Reg <= '1';
                                Save_ALU <= '1';
                                Set_BusA_To(2 downto 0) <= "101";
                                case to_integer(unsigned(IR(5 downto 4))) is
                                when 0|1|2 =>
                                        Set_BusB_To(2 downto 1) <= IR(5 downto 4);
                                        Set_BusB_To(0) <= '1';
                                when others =>
                                        Set_BusB_To <= "1000";
                                end case;
                                TStates <= "100";
                                Arith16 <= '1';
                        when 3 =>
                                NoRead <= '1';
                                Read_To_Reg <= '1';
                                Save_ALU <= '1';
                                ALU_Op <= "0001";
                                Set_BusA_To(2 downto 0) <= "100";
                                case to_integer(unsigned(IR(5 downto 4))) is
                                when 0|1|2 =>
                                        Set_BusB_To(2 downto 1) <= IR(5 downto 4);
                                when others =>
                                        Set_BusB_To <= "1001";
                                end case;
                                Arith16 <= '1';
                        when others =>
                        end case;
                when "00000011"|"00010011"|"00100011"|"00110011" =>
                        -- INC ss
                        TStates <= "110";
                        IncDec_16(3 downto 2) <= "01";
                        IncDec_16(1 downto 0) <= DPair;
                when "00001011"|"00011011"|"00101011"|"00111011" =>
                        -- DEC ss
                        TStates <= "110";
                        IncDec_16(3 downto 2) <= "11";
                        IncDec_16(1 downto 0) <= DPair;

-- ROTATE AND SHIFT GROUP
                when "00000111"
                        -- RLCA
                        |"00010111"
                        -- RLA
                        |"00001111"
                        -- RRCA
                        |"00011111" =>
                        -- RRA
                        Set_BusA_To(2 downto 0) <= "111";
                        ALU_Op <= "1000";
                        Read_To_Reg <= '1';
                        Save_ALU <= '1';

-- JUMP GROUP
                when "11000011" =>
```

```
                -- JP nn
                MCycles <= "011";
                case to_integer(unsigned(MCycle)) is
                when 2 =>
                        Inc_PC <= '1';
                        LDZ <= '1';
                when 3 =>
                        Inc_PC <= '1';
                        Jump <= '1';
                when others => null;
                end case;
        when "11000010"|"11001010"|"11010010"|"11011010"|"11100010"|"11101010"|"11110010"|"11111010" =>
                if IR(5) = '1' and Mode = 3 then
                        case IRB(4 downto 3) is
                        when "00" =>
                                -- LD ($FF00+C),A
                                MCycles <= "010";
                                case to_integer(unsigned(MCycle)) is
                                when 1 =>
                                        Set_Addr_To <= aBC;
                                        Set_BusB_To             <= "0111";
                                when 2 =>
                                        Write_80 <= '1';
                                        IORQ <= '1';
                                when others =>
                                end case;
                        when "01" =>
                                -- LD (nn),A
                                MCycles <= "100";
                                case to_integer(unsigned(MCycle)) is
                                when 2 =>
                                        Inc_PC <= '1';
                                        LDZ <= '1';
                                when 3 =>
                                        Set_Addr_To <= aZI;
                                        Inc_PC <= '1';
                                        Set_BusB_To <= "0111";
                                when 4 =>
                                        Write_80 <= '1';
                                when others => null;
                                end case;
                        when "10" =>
                                -- LD A,($FF00+C)
                                MCycles <= "010";
                                case to_integer(unsigned(MCycle)) is
                                when 1 =>
                                        Set_Addr_To <= aBC;
                                when 2 =>
                                        Read_To_Acc <= '1';
                                        IORQ <= '1';
                                when others =>
                                end case;
                        when "11" =>
                                -- LD A,(nn)
                                MCycles <= "100";
                                case to_integer(unsigned(MCycle)) is
                                when 2 =>
                                        Inc_PC <= '1';
                                        LDZ <= '1';
                                when 3 =>
                                        Set_Addr_To <= aZI;
                                        Inc_PC <= '1';
                                when 4 =>
                                        Read_To_Acc <= '1';
                                when others => null;
                                end case;
                        end case;
                else
                        -- JP cc,nn
                        MCycles <= "011";
```

```
                        case to_integer(unsigned(MCycle)) is
                        when 2 =>
                                Inc_PC <= '1';
                                LDZ <= '1';
                        when 3 =>
                                Inc_PC <= '1';
                                if is_cc_true(F, to_bitvector(IR(5 downto 3))) then
                                        Jump <= '1';
                                end if;
                        when others => null;
                        end case;
                end if;
when "00011000" =>
                if Mode /= 2 then
                        -- JR e
                        MCycles <= "011";
                        case to_integer(unsigned(MCycle)) is
                        when 2 =>
                                Inc_PC <= '1';
                        when 3 =>
                                NoRead <= '1';
                                JumpE <= '1';
                                TStates <= "101";
                        when others => null;
                        end case;
                end if;
when "00111000" =>
                if Mode /= 2 then
                        -- JR C,e
                        MCycles <= "011";
                        case to_integer(unsigned(MCycle)) is
                        when 2 =>
                                Inc_PC <= '1';
                                if F(Flag_C) = '0' then
                                        MCycles <= "010";
                                end if;
                        when 3 =>
                                NoRead <= '1';
                                JumpE <= '1';
                                TStates <= "101";
                        when others => null;
                        end case;
                end if;
when "00110000" =>
                if Mode /= 2 then
                        -- JR NC,e
                        MCycles <= "011";
                        case to_integer(unsigned(MCycle)) is
                        when 2 =>
                                Inc_PC <= '1';
                                if F(Flag_C) = '1' then
                                        MCycles <= "010";
                                end if;
                        when 3 =>
                                NoRead <= '1';
                                JumpE <= '1';
                                TStates <= "101";
                        when others => null;
                        end case;
                end if;
when "00101000" =>
                if Mode /= 2 then
                        -- JR Z,e
                        MCycles <= "011";
                        case to_integer(unsigned(MCycle)) is
                        when 2 =>
                                Inc_PC <= '1';
                                if F(Flag_Z) = '0' then
                                        MCycles <= "010";
                                end if;
```

```
                        when 3 =>
                                NoRead <= '1';
                                JumpE <= '1';
                                TStates <= "101";
                        when others => null;
                        end case;
                end if;
        when "00100000" =>
                if Mode /= 2 then
                        -- JR NZ,e
                        MCycles <= "011";
                        case to_integer(unsigned(MCycle)) is
                        when 2 =>
                                Inc_PC <= '1';
                                if F(Flag_Z) = '1' then
                                        MCycles <= "010";
                                end if;
                        when 3 =>
                                NoRead <= '1';
                                JumpE <= '1';
                                TStates <= "101";
                        when others => null;
                        end case;
                end if;
        when "11101001" =>
                -- JP (HL)
                JumpXY <= '1';
        when "00010000" =>
                if Mode = 3 then
                        I_DJNZ <= '1';
                elsif Mode < 2 then
                        -- DJNZ,e
                        MCycles <= "011";
                        case to_integer(unsigned(MCycle)) is
                        when 1 =>
                                TStates <= "101";
                                I_DJNZ <= '1';
                                Set_BusB_To <= "1010";
                                Set_BusA_To(2 downto 0) <= "000";
                                Read_To_Reg <= '1';
                                Save_ALU <= '1';
                                ALU_Op <= "0010";
                        when 2 =>
                                I_DJNZ <= '1';
                                Inc_PC <= '1';
                        when 3 =>
                                NoRead <= '1';
                                JumpE <= '1';
                                TStates <= "101";
                        when others => null;
                        end case;
                end if;

-- CALL AND RETURN GROUP
        when "11001101" =>
                -- CALL nn
                MCycles <= "101";
                case to_integer(unsigned(MCycle)) is
                when 2 =>
                        Inc_PC <= '1';
                        LDZ <= '1';
                when 3 =>
                        IncDec_16 <= "1111";
                        Inc_PC <= '1';
                        TStates <= "100";
                        Set_Addr_To <= aSP;
                        LDW <= '1';
                        Set_BusB_To <= "1101";
                when 4 =>
                        Write_80 <= '1';
```

```
                        IncDec_16 <= "1111";
                        Set_Addr_To <= aSP;
                        Set_BusB_To <= "1100";
                when 5 =>
                        Write_80 <= '1';
                        Call <= '1';
                when others => null;
                end case;
        when "11000100"|"11001100"|"11010100"|"11011100"|"11100100"|"11101100"|"11110100"|"11111100" =>
                if IR(5) = '0' or Mode /= 3 then
                        -- CALL cc,nn
                        MCycles <= "101";
                        case to_integer(unsigned(MCycle)) is
                        when 2 =>
                                Inc_PC <= '1';
                                LDZ <= '1';
                        when 3 =>
                                Inc_PC <= '1';
                                LDW <= '1';
                                if is_cc_true(F, to_bitvector(IR(5 downto 3))) then
                                        IncDec_16 <= "1111";
                                        Set_Addr_TO <= aSP;
                                        TStates <= "100";
                                        Set_BusB_To <= "1101";
                                else
                                        MCycles <= "011";
                                end if;
                        when 4 =>
                                Write_80 <= '1';
                                IncDec_16 <= "1111";
                                Set_Addr_To <= aSP;
                                Set_BusB_To <= "1100";
                        when 5 =>
                                Write_80 <= '1';
                                Call <= '1';
                        when others => null;
                        end case;
                end if;
        when "11001001" =>
                -- RET
                MCycles <= "011";
                case to_integer(unsigned(MCycle)) is
                when 1 =>
                        TStates <= "101";
                        Set_Addr_TO <= aSP;
                when 2 =>
                        IncDec_16 <= "0111";
                        Set_Addr_To <= aSP;
                        LDZ <= '1';
                when 3 =>
                        Jump <= '1';
                        IncDec_16 <= "0111";
                when others => null;
                end case;
        when "11000000"|"11001000"|"11010000"|"11011000"|"11100000"|"11101000"|"11110000"|"11111000" =>
                if IR(5) = '1' and Mode = 3 then
                        case IRB(4 downto 3) is
                        when "00" =>
                                -- LD ($FF00+nn),A
                                MCycles <= "011";
                                case to_integer(unsigned(MCycle)) is
                                when 2 =>
                                        Inc_PC <= '1';
                                        Set_Addr_To <= aIOA;
                                        Set_BusB_To     <= "0111";
                                when 3 =>
                                        Write_80 <= '1';
                                when others => null;
                                end case;
                        when "01" =>
```

```
                -- ADD SP,n
                MCycles <= "011";
                case to_integer(unsigned(MCycle)) is
                when 2 =>
                        ALU_Op <= "0000";
                        Inc_PC <= '1';
                        Read_To_Reg <= '1';
                        Save_ALU <= '1';
                        Set_BusA_To <= "1000";
                        Set_BusB_To <= "0110";
                when 3 =>
                        NoRead <= '1';
                        Read_To_Reg <= '1';
                        Save_ALU <= '1';
                        ALU_Op <= "0001";
                        Set_BusA_To <= "1001";
                        Set_BusB_To <= "1110";          -- Incorrect unsigned !!!!!!!!!!!!!!!!!!!!!
                when others =>
                end case;
        when "10" =>
                -- LD A,($FF00+nn)
                MCycles <= "011";
                case to_integer(unsigned(MCycle)) is
                when 2 =>
                        Inc_PC <= '1';
                        Set_Addr_To <= aIOA;
                when 3 =>
                        Read_To_Acc <= '1';
                when others => null;
                end case;
        when "11" =>
                -- LD HL,SP+n          -- Not correct !!!!!!!!!!!!!!!!!!!
                MCycles <= "101";
                case to_integer(unsigned(MCycle)) is
                when 2 =>
                        Inc_PC <= '1';
                        LDZ <= '1';
                when 3 =>
                        Set_Addr_To <= aZI;
                        Inc_PC <= '1';
                        LDW <= '1';
                when 4 =>
                        Set_BusA_To(2 downto 0) <= "101"; -- L
                        Read_To_Reg <= '1';
                        Inc_WZ <= '1';
                        Set_Addr_To <= aZI;
                when 5 =>
                        Set_BusA_To(2 downto 0) <= "100"; -- H
                        Read_To_Reg <= '1';
                when others => null;
                end case;
        end case;
else
                -- RET cc
                MCycles <= "011";
                case to_integer(unsigned(MCycle)) is
                when 1 =>
                        if is_cc_true(F, to_bitvector(IR(5 downto 3))) then
                                Set_Addr_TO <= aSP;
                        else
                                MCycles <= "001";
                        end if;
                        TStates <= "101";
                when 2 =>
                        IncDec_16 <= "0111";
                        Set_Addr_To <= aSP;
                        LDZ <= '1';
                when 3 =>
                        Jump <= '1';
                        IncDec_16 <= "0111";
```

```
                        when others => null;
                        end case;
                end if;
        when "11000111"|"11001111"|"11010111"|"11011111"|"11100111"|"11101111"|"11110111"|"11111111" =>
                -- RST p
                MCycles <= "011";
                case to_integer(unsigned(MCycle)) is
                when 1 =>
                        TStates <= "101";
                        IncDec_16 <= "1111";
                        Set_Addr_To <= aSP;
                        Set_BusB_To <= "1101";
                when 2 =>
                        Write_80 <= '1';
                        IncDec_16 <= "1111";
                        Set_Addr_To <= aSP;
                        Set_BusB_To <= "1100";
                when 3 =>
                        Write_80 <= '1';
                        RstP <= '1';
                when others => null;
                end case;

-- INPUT AND OUTPUT GROUP
        when "11011011" =>
                if Mode /= 3 then
                        -- IN A,(n)
                        MCycles <= "011";
                        case to_integer(unsigned(MCycle)) is
                        when 2 =>
                                Inc_PC <= '1';
                                Set_Addr_To <= aIOA;
                        when 3 =>
                                Read_To_Acc <= '1';
                                IORQ <= '1';
                        when others => null;
                        end case;
                end if;
        when "11010011" =>
                if Mode /= 3 then
                        -- OUT (n),A
                        MCycles <= "011";
                        case to_integer(unsigned(MCycle)) is
                        when 2 =>
                                Inc_PC <= '1';
                                Set_Addr_To <= aIOA;
                                Set_BusB_To             <= "0111";
                        when 3 =>
                                Write_80 <= '1';
                                IORQ <= '1';
                        when others => null;
                        end case;
                end if;

-------------------------------------------------------------------------
-------------------------------------------------------------------------
-- MULTIBYTE INSTRUCTIONS
-------------------------------------------------------------------------
-------------------------------------------------------------------------

        when "11001011" =>
                if Mode /= 2 then
                        Prefix <= "01";
                end if;

        when "11101101" =>
                if Mode < 2 then
                        Prefix <= "10";
                end if;
```

```
                when "11011101"|"11111101" =>
                        if Mode < 2 then
                                Prefix <= "11";
                        end if;

                end case;

                when "01" =>

-----------------------------------------------------------------------------
--
--              CB prefixed instructions
--
-----------------------------------------------------------------------------

                        Set_BusA_To(2 downto 0) <= IR(2 downto 0);
                        Set_BusB_To(2 downto 0) <= IR(2 downto 0);

                        case IRB is
                        when "00000000"|"00000001"|"00000010"|"00000011"|"00000100"|"00000101"|"00000111"
                                |"00010000"|"00010001"|"00010010"|"00010011"|"00010100"|"00010101"|"00010111"
                                |"00001000"|"00001001"|"00001010"|"00001011"|"00001100"|"00001101"|"00001111"
                                |"00011000"|"00011001"|"00011010"|"00011011"|"00011100"|"00011101"|"00011111"
                                |"00100000"|"00100001"|"00100010"|"00100011"|"00100100"|"00100101"|"00100111"
                                |"00101000"|"00101001"|"00101010"|"00101011"|"00101100"|"00101101"|"00101111"
                                |"00110000"|"00110001"|"00110010"|"00110011"|"00110100"|"00110101"|"00110111"
                                |"00111000"|"00111001"|"00111010"|"00111011"|"00111100"|"00111101"|"00111111" =>
                                -- RLC r
                                -- RL r
                                -- RRC r
                                -- RR r
                                -- SLA r
                                -- SRA r
                                -- SRL r
                                -- SLL r (Undocumented) / SWAP r
                                if MCycle = "001" then
                                        ALU_Op <= "1000";
                                        Read_To_Reg <= '1';
                                        Save_ALU <= '1';
                                end if;
                        when "00000110"|"00010110"|"00001110"|"00011110"|"00101110"|"00111110"|"00100110"|"00110110" =>
                                -- RLC (HL)
                                -- RL (HL)
                                -- RRC (HL)
                                -- RR (HL)
                                -- SRA (HL)
                                -- SRL (HL)
                                -- SLA (HL)
                                -- SLL (HL) (Undocumented) / SWAP (HL)
                                MCycles <= "011";
                                case to_integer(unsigned(MCycle)) is
                                when 1 | 7 =>
                                        Set_Addr_To <= aXY;
                                when 2 =>
                                        ALU_Op <= "1000";
                                        Read_To_Reg <= '1';
                                        Save_ALU <= '1';
                                        Set_Addr_To <= aXY;
                                        TStates <= "100";
                                when 3 =>
                                        Write_80 <= '1';
                                when others =>
                                end case;
                        when "01000000"|"01000001"|"01000010"|"01000011"|"01000100"|"01000101"|"01000111"
                                |"01001000"|"01001001"|"01001010"|"01001011"|"01001100"|"01001101"|"01001111"
                                |"01010000"|"01010001"|"01010010"|"01010011"|"01010100"|"01010101"|"01010111"
                                |"01011000"|"01011001"|"01011010"|"01011011"|"01011100"|"01011101"|"01011111"
                                |"01100000"|"01100001"|"01100010"|"01100011"|"01100100"|"01100101"|"01100111"
                                |"01101000"|"01101001"|"01101010"|"01101011"|"01101100"|"01101101"|"01101111"
                                |"01110000"|"01110001"|"01110010"|"01110011"|"01110100"|"01110101"|"01110111"
```

```
		|"01111000"|"01111001"|"01111010"|"01111011"|"01111100"|"01111101"|"01111111" =>
		-- BIT b,r
		if MCycle = "001" then
			Set_BusB_To(2 downto 0) <= IR(2 downto 0);
			ALU_Op <= "1001";
		end if;
when "01000110"|"01001110"|"01010110"|"01011110"|"01100110"|"01101110"|"01110110"|"01111110" =>
		-- BIT b,(HL)
		MCycles <= "010";
		case to_integer(unsigned(MCycle)) is
		when 1 | 7 =>
			Set_Addr_To <= aXY;
		when 2 =>
			ALU_Op <= "1001";
			TStates <= "100";
		when others =>
		end case;
when "11000000"|"11000001"|"11000010"|"11000011"|"11000100"|"11000101"|"11000111"
		|"11001000"|"11001001"|"11001010"|"11001011"|"11001100"|"11001101"|"11001111"
		|"11010000"|"11010001"|"11010010"|"11010011"|"11010100"|"11010101"|"11010111"
		|"11011000"|"11011001"|"11011010"|"11011011"|"11011100"|"11011101"|"11011111"
		|"11100000"|"11100001"|"11100010"|"11100011"|"11100100"|"11100101"|"11100111"
		|"11101000"|"11101001"|"11101010"|"11101011"|"11101100"|"11101101"|"11101111"
		|"11110000"|"11110001"|"11110010"|"11110011"|"11110100"|"11110101"|"11110111"
		|"11111000"|"11111001"|"11111010"|"11111011"|"11111100"|"11111101"|"11111111" =>
		-- SET b,r
		if MCycle = "001" then
			ALU_Op <= "1010";
			Read_To_Reg <= '1';
			Save_ALU <= '1';
		end if;
when "11000110"|"11001110"|"11010110"|"11011110"|"11100110"|"11101110"|"11110110"|"11111110" =>
		-- SET b,(HL)
		MCycles <= "011";
		case to_integer(unsigned(MCycle)) is
		when 1 | 7 =>
			Set_Addr_To <= aXY;
		when 2 =>
			ALU_Op <= "1010";
			Read_To_Reg <= '1';
			Save_ALU <= '1';
			Set_Addr_To <= aXY;
			TStates <= "100";
		when 3 =>
			Write_80 <= '1';
		when others =>
		end case;
when "10000000"|"10000001"|"10000010"|"10000011"|"10000100"|"10000101"|"10000111"
		|"10001000"|"10001001"|"10001010"|"10001011"|"10001100"|"10001101"|"10001111"
		|"10010000"|"10010001"|"10010010"|"10010011"|"10010100"|"10010101"|"10010111"
		|"10011000"|"10011001"|"10011010"|"10011011"|"10011100"|"10011101"|"10011111"
		|"10100000"|"10100001"|"10100010"|"10100011"|"10100100"|"10100101"|"10100111"
		|"10101000"|"10101001"|"10101010"|"10101011"|"10101100"|"10101101"|"10101111"
		|"10110000"|"10110001"|"10110010"|"10110011"|"10110100"|"10110101"|"10110111"
		|"10111000"|"10111001"|"10111010"|"10111011"|"10111100"|"10111101"|"10111111" =>
		-- RES b,r
		if MCycle = "001" then
			ALU_Op <= "1011";
			Read_To_Reg <= '1';
			Save_ALU <= '1';
		end if;
when "10000110"|"10001110"|"10010110"|"10011110"|"10100110"|"10101110"|"10110110"|"10111110" =>
		-- RES b,(HL)
		MCycles <= "011";
		case to_integer(unsigned(MCycle)) is
		when 1 | 7 =>
			Set_Addr_To <= aXY;
		when 2 =>
			ALU_Op <= "1011";
			Read_To_Reg <= '1';
```

```
                                        Save_ALU <= '1';
                                        Set_Addr_To <= aXY;
                                        TStates <= "100";
                                when 3 =>
                                        Write_80 <= '1';
                                when others =>
                                end case;
                        end case;

                when others =>

-----------------------------------------------------------------------------
--
--      ED prefixed instructions
--
-----------------------------------------------------------------------------

                        case IRB is
                        when "00000000"|"00000001"|"00000010"|"00000011"|"00000100"|"00000101"|"00000110"|"00000111"
                                |"00001000"|"00001001"|"00001010"|"00001011"|"00001100"|"00001101"|"00001110"|"00001111"
                                |"00010000"|"00010001"|"00010010"|"00010011"|"00010100"|"00010101"|"00010110"|"00010111"
                                |"00011000"|"00011001"|"00011010"|"00011011"|"00011100"|"00011101"|"00011110"|"00011111"
                                |"00100000"|"00100001"|"00100010"|"00100011"|"00100100"|"00100101"|"00100110"|"00100111"
                                |"00101000"|"00101001"|"00101010"|"00101011"|"00101100"|"00101101"|"00101110"|"00101111"
                                |"00110000"|"00110001"|"00110010"|"00110011"|"00110100"|"00110101"|"00110110"|"00110111"
                                |"00111000"|"00111001"|"00111010"|"00111011"|"00111100"|"00111101"|"00111110"|"00111111"


                                |"10000000"|"10000001"|"10000010"|"10000011"|"10000100"|"10000101"|"10000110"|"10000111"
                                |"10001000"|"10001001"|"10001010"|"10001011"|"10001100"|"10001101"|"10001110"|"10001111"
                                |"10010000"|"10010001"|"10010010"|"10010011"|"10010100"|"10010101"|"10010110"|"10010111"
                                |"10011000"|"10011001"|"10011010"|"10011011"|"10011100"|"10011101"|"10011110"|"10011111"
                                |                       "10100100"|"10100101"|"10100110"|"10100111"
                                |                       "10101100"|"10101101"|"10101110"|"10101111"
                                |                       "10110100"|"10110101"|"10110110"|"10110111"
                                |                       "10111100"|"10111101"|"10111110"|"10111111"
                                |"11000000"|"11000001"|"11000010"|"11000011"|"11000100"|"11000101"|"11000110"|"11000111"
                                |"11001000"|"11001001"|"11001010"|"11001011"|"11001100"|"11001101"|"11001110"|"11001111"
                                |"11010000"|"11010001"|"11010010"|"11010011"|"11010100"|"11010101"|"11010110"|"11010111"
                                |"11011000"|"11011001"|"11011010"|"11011011"|"11011100"|"11011101"|"11011110"|"11011111"
                                |"11100000"|"11100001"|"11100010"|"11100011"|"11100100"|"11100101"|"11100110"|"11100111"
                                |"11101000"|"11101001"|"11101010"|"11101011"|"11101100"|"11101101"|"11101110"|"11101111"
                                |"11110000"|"11110001"|"11110010"|"11110011"|"11110100"|"11110101"|"11110110"|"11110111"
                                |"11111000"|"11111001"|"11111010"|"11111011"|"11111100"|"11111101"|"11111110"|"11111111" =>
                                null; -- NOP, undocumented
                        when "01111110"|"01111111" =>
                                -- NOP, undocumented
                                null;
-- 8 BIT LOAD GROUP
                        when "01010111" =>
                                -- LD A,I
                                Special_LD <= "100";
                                TStates <= "101";
                        when "01011111" =>
                                -- LD A,R
                                Special_LD <= "101";
                                TStates <= "101";
                        when "01000111" =>
                                -- LD I,A
                                Special_LD <= "110";
                                TStates <= "101";
                        when "01001111" =>
                                -- LD R,A
                                Special_LD <= "111";
                                TStates <= "101";
-- 16 BIT LOAD GROUP
                        when "01001011"|"01011011"|"01101011"|"01111011" =>
                                -- LD dd,(nn)
                                MCycles <= "101";
                                case to_integer(unsigned(MCycle)) is
```

```
		when 2 =>
			Inc_PC <= '1';
			LDZ <= '1';
		when 3 =>
			Set_Addr_To <= aZI;
			Inc_PC <= '1';
			LDW <= '1';
		when 4 =>
			Read_To_Reg <= '1';
			if IR(5 downto 4) = "11" then
				Set_BusA_To <= "1000";
			else
				Set_BusA_To(2 downto 1) <= IR(5 downto 4);
				Set_BusA_To(0) <= '1';
			end if;
			Inc_WZ <= '1';
			Set_Addr_To <= aZI;
		when 5 =>
			Read_To_Reg <= '1';
			if IR(5 downto 4) = "11" then
				Set_BusA_To <= "1001";
			else
				Set_BusA_To(2 downto 1) <= IR(5 downto 4);
				Set_BusA_To(0) <= '0';
			end if;
		when others => null;
		end case;
	when "01000011"|"01010011"|"01100011"|"01110011" =>
		-- LD (nn),dd
		MCycles <= "101";
		case to_integer(unsigned(MCycle)) is
		when 2 =>
			Inc_PC <= '1';
			LDZ <= '1';
		when 3 =>
			Set_Addr_To <= aZI;
			Inc_PC <= '1';
			LDW <= '1';
			if IR(5 downto 4) = "11" then
				Set_BusB_To <= "1000";
			else
				Set_BusB_To(2 downto 1) <= IR(5 downto 4);
				Set_BusB_To(0) <= '1';
				Set_BusB_To(3) <= '0';
			end if;
		when 4 =>
			Inc_WZ <= '1';
			Set_Addr_To <= aZI;
			Write_80 <= '1';
			if IR(5 downto 4) = "11" then
				Set_BusB_To <= "1001";
			else
				Set_BusB_To(2 downto 1) <= IR(5 downto 4);
				Set_BusB_To(0) <= '0';
				Set_BusB_To(3) <= '0';
			end if;
		when 5 =>
			Write_80 <= '1';
		when others => null;
		end case;
	when "10100000" | "10101000" | "10110000" | "10111000" =>
		-- LDI, LDD, LDIR, LDDR
		MCycles <= "100";
		case to_integer(unsigned(MCycle)) is
		when 1 =>
			Set_Addr_To <= aXY;
			IncDec_16 <= "1100"; -- BC
		when 2 =>
			Set_BusB_To <= "0110";
			Set_BusA_To(2 downto 0) <= "111";
```

```
                        ALU_Op <= "0000";
                        Set_Addr_To <= aDE;
                        if IR(3) = '0' then
                                IncDec_16 <= "0110"; -- IX
                        else
                                IncDec_16 <= "1110";
                        end if;
                when 3 =>
                        I_BT <= '1';
                        TStates <= "101";
                        Write_80 <= '1';
                        if IR(3) = '0' then
                                IncDec_16 <= "0101"; -- DE
                        else
                                IncDec_16 <= "1101";
                        end if;
                when 4 =>
                        NoRead <= '1';
                        TStates <= "101";
                when others => null;
                end case;
        when "10100001" | "10101001" | "10110001" | "10111001" =>
                -- CPI, CPD, CPIR, CPDR
                MCycles <= "100";
                case to_integer(unsigned(MCycle)) is
                when 1 =>
                        Set_Addr_To <= aXY;
                        IncDec_16 <= "1100"; -- BC
                when 2 =>
                        Set_BusB_To <= "0110";
                        Set_BusA_To(2 downto 0) <= "111";
                        ALU_Op <= "0111";
                        Save_ALU <= '1';
                        PreserveC <= '1';
                        if IR(3) = '0' then
                                IncDec_16 <= "0110";
                        else
                                IncDec_16 <= "1110";
                        end if;
                when 3 =>
                        NoRead <= '1';
                        I_BC <= '1';
                        TStates <= "101";
                when 4 =>
                        NoRead <= '1';
                        TStates <= "101";
                when others => null;
                end case;
        when "01000100"|"01001100"|"01010100"|"01011100"|"01100100"|"01101100"|"01110100"|"01111100" =>
                -- NEG
                Alu_OP <= "0010";
                Set_BusB_To <= "0111";
                Set_BusA_To <= "1010";
                Read_To_Acc <= '1';
                Save_ALU <= '1';
        when "01000110"|"01001110"|"01100110"|"01101110" =>
                -- IM 0
                IMode <= "00";
        when "01010110"|"01110110" =>
                -- IM 1
                IMode <= "01";
        when "01011110"|"01110111" =>
                -- IM 2
                IMode <= "10";
-- 16 bit arithmetic
        when "01001010"|"01011010"|"01101010"|"01111010" =>
                -- ADC HL,ss
                MCycles <= "011";
                case to_integer(unsigned(MCycle)) is
                when 2 =>
```

```
                        NoRead <= '1';
                        ALU_Op <= "0001";
                        Read_To_Reg <= '1';
                        Save_ALU <= '1';
                        Set_BusA_To(2 downto 0) <= "101";
                        case to_integer(unsigned(IR(5 downto 4))) is
                        when 0|1|2 =>
                                Set_BusB_To(2 downto 1) <= IR(5 downto 4);
                        Set_BusB_To(0) <= '1';
                                when others =>
                                Set_BusB_To <= "1000";
                        end case;
                        TStates <= "100";
                when 3 =>
                        NoRead <= '1';
                        Read_To_Reg <= '1';
                        Save_ALU <= '1';
                        ALU_Op <= "0001";
                        Set_BusA_To(2 downto 0) <= "100";
                        case to_integer(unsigned(IR(5 downto 4))) is
                        when 0|1|2 =>
                                Set_BusB_To(2 downto 1) <= IR(5 downto 4);
                                Set_BusB_To(0) <= '0';
                        when others =>
                                Set_BusB_To <= "1001";
                        end case;
                when others =>
                end case;
        when "01000010"|"01010010"|"01100010"|"01110010" =>
                -- SBC HL,ss
                MCycles <= "011";
                case to_integer(unsigned(MCycle)) is
                when 2 =>
                        NoRead <= '1';
                        ALU_Op <= "0011";
                        Read_To_Reg <= '1';
                        Save_ALU <= '1';
                        Set_BusA_To(2 downto 0) <= "101";
                        case to_integer(unsigned(IR(5 downto 4))) is
                        when 0|1|2 =>
                                Set_BusB_To(2 downto 1) <= IR(5 downto 4);
                                Set_BusB_To(0) <= '1';
                        when others =>
                                Set_BusB_To <= "1000";
                        end case;
                        TStates <= "100";
                when 3 =>
                        NoRead <= '1';
                        ALU_Op <= "0011";
                        Read_To_Reg <= '1';
                        Save_ALU <= '1';
                        Set_BusA_To(2 downto 0) <= "100";
                        case to_integer(unsigned(IR(5 downto 4))) is
                        when 0|1|2 =>
                                Set_BusB_To(2 downto 1) <= IR(5 downto 4);
                        when others =>
                                        Set_BusB_To <= "1001";
                        end case;
                when others =>
                end case;
        when "01101111" =>
                -- RLD
                MCycles <= "100";
                case to_integer(unsigned(MCycle)) is
                when 2 =>
                        NoRead <= '1';
                        Set_Addr_To <= aXY;
                when 3 =>
                        Read_To_Reg <= '1';
                        Set_BusB_To(2 downto 0) <= "110";
```

```
                        Set_BusA_To(2 downto 0) <= "111";
                        ALU_Op <= "1101";
                        TStates <= "100";
                        Set_Addr_To <= aXY;
                        Save_ALU <= '1';
                when 4 =>
                        I_RLD <= '1';
                        Write_80<= '1';
                when others =>
                end case;
        when "01100111" =>
                -- RRD
                MCycles <= "100";
                case to_integer(unsigned(MCycle)) is
                when 2 =>
                        Set_Addr_To <= aXY;
                when 3 =>
                        Read_To_Reg <= '1';
                        Set_BusB_To(2 downto 0) <= "110";
                        Set_BusA_To(2 downto 0) <= "111";
                        ALU_Op <= "1110";
                        TStates <= "100";
                        Set_Addr_To <= aXY;
                        Save_ALU <= '1';
                when 4 =>
                        I_RRD <= '1';
                        Write_80 <= '1';
                when others =>
                end case;
        when "01000101"|"01001101"|"01010101"|"01011101"|"01100101"|"01101101"|"01110101"|"01111101" =>
                -- RETI, RETN
                MCycles <= "011";
                case to_integer(unsigned(MCycle)) is
                when 1 =>
                        Set_Addr_TO <= aSP;
                when 2 =>
                        IncDec_16 <= "0111";
                        Set_Addr_To <= aSP;
                        LDZ <= '1';
                when 3 =>
                        Jump <= '1';
                        IncDec_16 <= "0111";
                        I_RETN <= '1';
                when others => null;
                end case;
        when "01000000"|"01001000"|"01010000"|"01011000"|"01100000"|"01101000"|"01110000"|"01111000" =>
                -- IN r,(C)
                MCycles <= "010";
                case to_integer(unsigned(MCycle)) is
                when 1 =>
                        Set_Addr_To <= aBC;
                when 2 =>
                        IORQ <= '1';
                        if IR(5 downto 3) /= "110" then
                                Read_To_Reg <= '1';
                                Set_BusA_To(2 downto 0) <= IR(5 downto 3);
                        end if;
                        I_INRC <= '1';
                when others =>
                end case;
        when "01000001"|"01001001"|"01010001"|"01011001"|"01100001"|"01101001"|"01110001"|"01111001" =>
                -- OUT (C),r
                -- OUT (C),0
                MCycles <= "010";
                case to_integer(unsigned(MCycle)) is
                when 1 =>
                        Set_Addr_To <= aBC;
                        Set_BusB_To(2 downto 0)         <= IR(5 downto 3);
                        if IR(5 downto 3) = "110" then
                                Set_BusB_To(3) <= '1';
```

```
                        end if;
                    when 2 =>
                        Write_80 <= '1';
                        IORQ <= '1';
                    when others =>
                    end case;
                when "10100010" | "10101010" | "10110010" | "10111010" =>
                    -- INI, IND, INIR, INDR
                    MCycles <= "100";
                    case to_integer(unsigned(MCycle)) is
                    when 1 =>
                        Set_Addr_To <= aBC;
                        Set_BusB_To <= "1010";
                        Set_BusA_To <= "0000";
                        Read_To_Reg <= '1';
                        Save_ALU <= '1';
                        ALU_Op <= "0010";
                    when 2 =>
                        IORQ <= '1';
                        Set_BusB_To <= "0110";
                        Set_Addr_To <= aXY;
                    when 3 =>
                        if IR(3) = '0' then
                            IncDec_16 <= "0010";
                        else
                            IncDec_16 <= "1010";
                        end if;
                        TStates <= "100";
                        Write_80 <= '1';
                        I_BTR <= '1';
                    when 4 =>
                        NoRead <= '1';
                        TStates <= "101";
                    when others => null;
                    end case;
                when "10100011" | "10101011" | "10110011" | "10111011" =>
                    -- OUTI, OUTD, OTIR, OTDR
                    MCycles <= "100";
                    case to_integer(unsigned(MCycle)) is
                    when 1 =>
                        TStates <= "101";
                        Set_Addr_To <= aXY;
                        Set_BusB_To <= "1010";
                        Set_BusA_To <= "0000";
                        Read_To_Reg <= '1';
                        Save_ALU <= '1';
                        ALU_Op <= "0010";
                    when 2 =>
                        Set_BusB_To <= "0110";
                        Set_Addr_To <= aBC;
                    when 3 =>
                        if IR(3) = '0' then
                            IncDec_16 <= "0010";
                        else
                            IncDec_16 <= "1010";
                        end if;
                        IORQ <= '1';
                        Write_80 <= '1';
                        I_BTR <= '1';
                    when 4 =>
                        NoRead <= '1';
                        TStates <= "101";
                    when others => null;
                    end case;
                end case;

        end case;

        if Mode = 1 then
            if MCycle = "001" then
```

```
--                                      TStates <= "100";
                                        else
                                                TStates <= "011";
                                        end if;
                                end if;

                                if Mode = 3 then
                                        if MCycle = "001" then
--                                              TStates <= "100";
                                        else
                                                TStates <= "100";
                                        end if;
                                end if;

                                if Mode < 2 then
                                        if MCycle = "110" then
                                                Inc_PC <= '1';
                                                if Mode = 1 then
                                                        Set_Addr_To <= aXY;
                                                        TStates <= "100";
                                                        Set_BusB_To(2 downto 0) <= SSS;
                                                        Set_BusB_To(3) <= '0';
                                                end if;
                                                if IRB = "00110110" or IRB = "11001011" then
                                                        Set_Addr_To <= aNone;
                                                end if;
                                        end if;
                                        if MCycle = "111" then
                                                if Mode = 0 then
                                                        TStates <= "101";
                                                end if;
                                                if ISet /= "01" then
                                                        Set_Addr_To <= aXY;
                                                end if;
                                                Set_BusB_To(2 downto 0) <= SSS;
                                                Set_BusB_To(3) <= '0';
                                                if IRB = "00110110" or ISet = "01" then
                                                        -- LD (HL),n
                                                        Inc_PC <= '1';
                                                else
                                                        NoRead <= '1';
                                                end if;
                                        end if;
                                end if;

        end process;

end;
```

## 8.3 T80_ALU.vhd

```
--
-- Z80 compatible microprocessor core
--
-- Version : 0247
--
-- Copyright (c) 2001-2002 Daniel Wallner (jesus@opencores.org)
--
-- All rights reserved
--
-- Redistribution and use in source and synthezised forms, with or without
-- modification, are permitted provided that the following conditions are met:
--
-- Redistributions of source code must retain the above copyright notice,
-- this list of conditions and the following disclaimer.
--
-- Redistributions in synthesized form must reproduce the above copyright
-- notice, this list of conditions and the following disclaimer in the
-- documentation and/or other materials provided with the distribution.
--
-- Neither the name of the author nor the names of other contributors may
-- be used to endorse or promote products derived from this software without
-- specific prior written permission.
--
-- THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
-- AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO,
-- THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
-- PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE
-- LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
-- CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
-- SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
-- INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
-- CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
-- ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
-- POSSIBILITY OF SUCH DAMAGE.
--
-- Please report bugs to the author, but before you do so, please
-- make sure that this is not a derivative work and that
-- you have the latest version of this file.
--
-- The latest version of this file can be found at:
--      http://www.opencores.org/cvsweb.shtml/t80/
--
-- Limitations :
--
-- File history :
--
--      0214 : Fixed mostly flags, only the block instructions now fail the zex regression test
--
--      0238 : Fixed zero flag for 16 bit SBC and ADC
--
--      0240 : Added GB operations
--
--      0242 : Cleanup
--
--      0247 : Cleanup

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity T80_ALU is
        generic(
                Mode : integer := 0;
                Flag_C : integer := 0;
                Flag_N : integer := 1;
                Flag_P : integer := 2;
                Flag_X : integer := 3;
                Flag_H : integer := 4;
```

```
		Flag_Y : integer := 5;
		Flag_Z : integer := 6;
		Flag_S : integer := 7
	);
	port(
		Arith16		: in std_logic;
		Z16			: in std_logic;
		ALU_Op		: in std_logic_vector(3 downto 0);
		IR			: in std_logic_vector(5 downto 0);
		ISet		: in std_logic_vector(1 downto 0);
		BusA		: in std_logic_vector(7 downto 0);
		BusB		: in std_logic_vector(7 downto 0);
		F_In		: in std_logic_vector(7 downto 0);
		Q			: out std_logic_vector(7 downto 0);
		F_Out		: out std_logic_vector(7 downto 0)
	);
end T80_ALU;

architecture rtl of T80_ALU is
	procedure AddSub(A : std_logic_vector;
			B : std_logic_vector;
			Sub : std_logic;
			Carry_In : std_logic;
			signal Res : out std_logic_vector;
			signal Carry : out std_logic) is
		variable B_i		: unsigned(A'length - 1 downto 0);
		variable Res_i		: unsigned(A'length + 1 downto 0);
	begin
		if Sub = '1' then
			B_i := not unsigned(B);
		else
			B_i := unsigned(B);
		end if;
		Res_i := unsigned("0" & A & Carry_In) + unsigned("0" & B_i & "1");
		Carry <= Res_i(A'length + 1);
		Res <= std_logic_vector(Res_i(A'length downto 1));
	end;

	-- AddSub variables (temporary signals)
	signal	UseCarry		: std_logic;
	signal	Carry7_v		: std_logic;
	signal	Overflow_v		: std_logic;
	signal	HalfCarry_v		: std_logic;
	signal	Carry_v			: std_logic;
	signal	Q_v				: std_logic_vector(7 downto 0);

	signal	BitMask			: std_logic_vector(7 downto 0);

begin
	with IR(5 downto 3) select BitMask <= "00000001" when "000",
			"00000010" when "001",
			"00000100" when "010",
			"00001000" when "011",
			"00010000" when "100",
			"00100000" when "101",
			"01000000" when "110",
			"10000000" when others;

	UseCarry <= not ALU_Op(2) and ALU_Op(0);
	AddSub(BusA(3 downto 0), BusB(3 downto 0), ALU_Op(1), ALU_Op(1) xor (UseCarry and F_In(Flag_C)), Q_v(3 downto 0), HalfCarry_v);
	AddSub(BusA(6 downto 4), BusB(6 downto 4), ALU_Op(1), HalfCarry_v, Q_v(6 downto 4), Carry7_v);
	AddSub(BusA(7 downto 7), BusB(7 downto 7), ALU_Op(1), Carry7_v, Q_v(7 downto 7), Carry_v);
	OverFlow_v <= Carry_v xor Carry7_v;

	process (Arith16, ALU_OP, F_In, BusA, BusB, IR, Q_v, Carry_v, HalfCarry_v, OverFlow_v, BitMask, ISet, Z16)
		variable Q_t : std_logic_vector(7 downto 0);
		variable DAA_Q : unsigned(8 downto 0);
	begin
		Q_t := "--------";
		F_Out <= F_In;
```

```
DAA_Q := "---------";
case ALU_Op is
when "0000" | "0001" |  "0010" | "0011" | "0100" | "0101" | "0110" | "0111" =>
	F_Out(Flag_N) <= '0';
	F_Out(Flag_C) <= '0';
	case ALU_OP(2 downto 0) is
	when "000" | "001" => -- ADD, ADC
		Q_t := Q_v;
		F_Out(Flag_C) <= Carry_v;
		F_Out(Flag_H) <= HalfCarry_v;
		F_Out(Flag_P) <= OverFlow_v;
	when "010" | "011" | "111" => -- SUB, SBC, CP
		Q_t := Q_v;
		F_Out(Flag_N) <= '1';
		F_Out(Flag_C) <= not Carry_v;
		F_Out(Flag_H) <= not HalfCarry_v;
		F_Out(Flag_P) <= OverFlow_v;
	when "100" => -- AND
		Q_t(7 downto 0) := BusA and BusB;
		F_Out(Flag_H) <= '1';
	when "101" => -- XOR
		Q_t(7 downto 0) := BusA xor BusB;
		F_Out(Flag_H) <= '0';
	when others => -- OR "110"
		Q_t(7 downto 0) := BusA or BusB;
		F_Out(Flag_H) <= '0';
	end case;
	if ALU_Op(2 downto 0) = "111" then -- CP
		F_Out(Flag_X) <= BusB(3);
		F_Out(Flag_Y) <= BusB(5);
	else
		F_Out(Flag_X) <= Q_t(3);
		F_Out(Flag_Y) <= Q_t(5);
	end if;
	if Q_t(7 downto 0) = "00000000" then
		F_Out(Flag_Z) <= '1';
		if Z16 = '1' then
			F_Out(Flag_Z) <= F_In(Flag_Z);		-- 16 bit ADC,SBC
		end if;
	else
		F_Out(Flag_Z) <= '0';
	end if;
	F_Out(Flag_S) <= Q_t(7);
	case ALU_Op(2 downto 0) is
	when "000" | "001" | "010" | "011" | "111" => -- ADD, ADC, SUB, SBC, CP
	when others =>
		F_Out(Flag_P) <= not (Q_t(0) xor Q_t(1) xor Q_t(2) xor Q_t(3) xor
			Q_t(4) xor Q_t(5) xor Q_t(6) xor Q_t(7));
	end case;
	if Arith16 = '1' then
		F_Out(Flag_S) <= F_In(Flag_S);
		F_Out(Flag_Z) <= F_In(Flag_Z);
		F_Out(Flag_P) <= F_In(Flag_P);
	end if;
when "1100" =>
	-- DAA
	F_Out(Flag_H) <= F_In(Flag_H);
	F_Out(Flag_C) <= F_In(Flag_C);
	DAA_Q(7 downto 0) := unsigned(BusA);
	DAA_Q(8) := '0';
	if F_In(Flag_N) = '0' then
		-- After addition
		-- Alow > 9 or H = 1
		if DAA_Q(3 downto 0) > 9 or F_In(Flag_H) = '1' then
			if (DAA_Q(3 downto 0) > 9) then
				F_Out(Flag_H) <= '1';
			else
				F_Out(Flag_H) <= '0';
			end if;
			DAA_Q := DAA_Q + 6;
```

```
                        end if;
                        -- new Ahigh > 9 or C = 1
                        if DAA_Q(8 downto 4) > 9 or F_In(Flag_C) = '1' then
                                DAA_Q := DAA_Q + 96; -- 0x60
                        end if;
                else
                        -- After subtraction
                        if DAA_Q(3 downto 0) > 9 or F_In(Flag_H) = '1' then
                                if DAA_Q(3 downto 0) > 5 then
                                        F_Out(Flag_H) <= '0';
                                end if;
                                DAA_Q(7 downto 0) := DAA_Q(7 downto 0) - 6;
                        end if;
                        if unsigned(BusA) > 153 or F_In(Flag_C) = '1' then
                                DAA_Q := DAA_Q - 352; -- 0x160
                        end if;
                end if;
                F_Out(Flag_X) <= DAA_Q(3);
                F_Out(Flag_Y) <= DAA_Q(5);
                F_Out(Flag_C) <= F_In(Flag_C) or DAA_Q(8);
                Q_t := std_logic_vector(DAA_Q(7 downto 0));
                if DAA_Q(7 downto 0) = "00000000" then
                        F_Out(Flag_Z) <= '1';
                else
                        F_Out(Flag_Z) <= '0';
                end if;
                F_Out(Flag_S) <= DAA_Q(7);
                F_Out(Flag_P) <= not (DAA_Q(0) xor DAA_Q(1) xor DAA_Q(2) xor DAA_Q(3) xor
                        DAA_Q(4) xor DAA_Q(5) xor DAA_Q(6) xor DAA_Q(7));
when "1101" | "1110" =>
                -- RLD, RRD
                Q_t(7 downto 4) := BusA(7 downto 4);
                if ALU_Op(0) = '1' then
                        Q_t(3 downto 0) := BusB(7 downto 4);
                else
                        Q_t(3 downto 0) := BusB(3 downto 0);
                end if;
                F_Out(Flag_H) <= '0';
                F_Out(Flag_N) <= '0';
                F_Out(Flag_X) <= Q_t(3);
                F_Out(Flag_Y) <= Q_t(5);
                if Q_t(7 downto 0) = "00000000" then
                        F_Out(Flag_Z) <= '1';
                else
                        F_Out(Flag_Z) <= '0';
                end if;
                F_Out(Flag_S) <= Q_t(7);
                F_Out(Flag_P) <= not (Q_t(0) xor Q_t(1) xor Q_t(2) xor Q_t(3) xor
                        Q_t(4) xor Q_t(5) xor Q_t(6) xor Q_t(7));
when "1001" =>
                -- BIT
                Q_t(7 downto 0) := BusB and BitMask;
                F_Out(Flag_S) <= Q_t(7);
                if Q_t(7 downto 0) = "00000000" then
                        F_Out(Flag_Z) <= '1';
                        F_Out(Flag_P) <= '1';
                else
                        F_Out(Flag_Z) <= '0';
                        F_Out(Flag_P) <= '0';
                end if;
                F_Out(Flag_H) <= '1';
                F_Out(Flag_N) <= '0';
                F_Out(Flag_X) <= '0';
                F_Out(Flag_Y) <= '0';
                if IR(2 downto 0) /= "110" then
                        F_Out(Flag_X) <= BusB(3);
                        F_Out(Flag_Y) <= BusB(5);
                end if;
when "1010" =>
                -- SET
```

```
                Q_t(7 downto 0) := BusB or BitMask;
            when "1011" =>
                -- RES
                Q_t(7 downto 0) := BusB and not BitMask;
            when "1000" =>
                -- ROT
                case IR(5 downto 3) is
                when "000" => -- RLC
                    Q_t(7 downto 1) := BusA(6 downto 0);
                    Q_t(0) := BusA(7);
                    F_Out(Flag_C) <= BusA(7);
                when "010" => -- RL
                    Q_t(7 downto 1) := BusA(6 downto 0);
                    Q_t(0) := F_In(Flag_C);
                    F_Out(Flag_C) <= BusA(7);
                when "001" => -- RRC
                    Q_t(6 downto 0) := BusA(7 downto 1);
                    Q_t(7) := BusA(0);
                    F_Out(Flag_C) <= BusA(0);
                when "011" => -- RR
                    Q_t(6 downto 0) := BusA(7 downto 1);
                    Q_t(7) := F_In(Flag_C);
                    F_Out(Flag_C) <= BusA(0);
                when "100" => -- SLA
                    Q_t(7 downto 1) := BusA(6 downto 0);
                    Q_t(0) := '0';
                    F_Out(Flag_C) <= BusA(7);
                when "110" => -- SLL (Undocumented) / SWAP
                    if Mode = 3 then
                        Q_t(7 downto 4) := BusA(3 downto 0);
                        Q_t(3 downto 0) := BusA(7 downto 4);
                        F_Out(Flag_C) <= '0';
                    else
                        Q_t(7 downto 1) := BusA(6 downto 0);
                        Q_t(0) := '1';
                        F_Out(Flag_C) <= BusA(7);
                    end if;
                when "101" => -- SRA
                    Q_t(6 downto 0) := BusA(7 downto 1);
                    Q_t(7) := BusA(7);
                    F_Out(Flag_C) <= BusA(0);
                when others => -- SRL
                    Q_t(6 downto 0) := BusA(7 downto 1);
                    Q_t(7) := '0';
                    F_Out(Flag_C) <= BusA(0);
                end case;
                F_Out(Flag_H) <= '0';
                F_Out(Flag_N) <= '0';
                F_Out(Flag_X) <= Q_t(3);
                F_Out(Flag_Y) <= Q_t(5);
                F_Out(Flag_S) <= Q_t(7);
                if Q_t(7 downto 0) = "00000000" then
                    F_Out(Flag_Z) <= '1';
                else
                    F_Out(Flag_Z) <= '0';
                end if;
                F_Out(Flag_P) <= not (Q_t(0) xor Q_t(1) xor Q_t(2) xor Q_t(3) xor
                    Q_t(4) xor Q_t(5) xor Q_t(6) xor Q_t(7));
                if ISet = "00" then
                    F_Out(Flag_P) <= F_In(Flag_P);
                    F_Out(Flag_S) <= F_In(Flag_S);
                    F_Out(Flag_Z) <= F_In(Flag_Z);
                end if;
            when others =>
                null;
            end case;
            Q <= Q_t;
        end process;

end;
```

## 8.4 T80_REG.vhd

```
--
-- T80 Registers, technology independent
--
-- Version : 0244
--
-- Copyright (c) 2002 Daniel Wallner (jesus@opencores.org)
--
-- All rights reserved
--
-- Redistribution and use in source and synthezised forms, with or without
-- modification, are permitted provided that the following conditions are met:
--
-- Redistributions of source code must retain the above copyright notice,
-- this list of conditions and the following disclaimer.
--
-- Redistributions in synthesized form must reproduce the above copyright
-- notice, this list of conditions and the following disclaimer in the
-- documentation and/or other materials provided with the distribution.
--
-- Neither the name of the author nor the names of other contributors may
-- be used to endorse or promote products derived from this software without
-- specific prior written permission.
--
-- THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
-- AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO,
-- THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
-- PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE
-- LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
-- CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
-- SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
-- INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
-- CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
-- ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
-- POSSIBILITY OF SUCH DAMAGE.
--
-- Please report bugs to the author, but before you do so, please
-- make sure that this is not a derivative work and that
-- you have the latest version of this file.
--
-- The latest version of this file can be found at:
--      http://www.opencores.org/cvsweb.shtml/t51/
--
-- Limitations :
--
-- File history :
--
--      0242 : Initial release
--
--      0244 : Changed to single register file
--

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity T80_Reg is
        port(
                Clk             : in std_logic;
                CEN             : in std_logic;
                WEH             : in std_logic;
                WEL             : in std_logic;
                AddrA           : in std_logic_vector(2 downto 0);
                AddrB           : in std_logic_vector(2 downto 0);
                AddrC           : in std_logic_vector(2 downto 0);
                DIH             : in std_logic_vector(7 downto 0);
                DIL             : in std_logic_vector(7 downto 0);
                DOAH            : out std_logic_vector(7 downto 0);
```

```
                DOAL            : out std_logic_vector(7 downto 0);
                DOBH            : out std_logic_vector(7 downto 0);
                DOBL            : out std_logic_vector(7 downto 0);
                DOCH            : out std_logic_vector(7 downto 0);
                DOCL            : out std_logic_vector(7 downto 0)
        );
end T80_Reg;

architecture rtl of T80_Reg is

        type Register_Image is array (natural range <>) of std_logic_vector(7 downto 0);
        signal  RegsH   : Register_Image(0 to 7);
        signal  RegsL   : Register_Image(0 to 7);

begin

        process (Clk)
        begin
                if Clk'event and Clk = '1' then
                        if CEN = '1' then
                                if WEH = '1' then
                                        RegsH(to_integer(unsigned(AddrA))) <= DIH;
                                end if;
                                if WEL = '1' then
                                        RegsL(to_integer(unsigned(AddrA))) <= DIL;
                                end if;
                        end if;
                end if;
        end process;

        DOAH <= RegsH(to_integer(unsigned(AddrA)));
        DOAL <= RegsL(to_integer(unsigned(AddrA)));
        DOBH <= RegsH(to_integer(unsigned(AddrB)));
        DOBL <= RegsL(to_integer(unsigned(AddrB)));
        DOCH <= RegsH(to_integer(unsigned(AddrC)));
        DOCL <= RegsL(to_integer(unsigned(AddrC)));

end;
```

## 8.5 T80_PACK.vhd

```
--
-- Z80 compatible microprocessor core
--
-- Version : 0242
--
-- Copyright (c) 2001-2002 Daniel Wallner (jesus@opencores.org)
--
-- All rights reserved
--
-- Redistribution and use in source and synthezised forms, with or without
-- modification, are permitted provided that the following conditions are met:
--
-- Redistributions of source code must retain the above copyright notice,
-- this list of conditions and the following disclaimer.
--
-- Redistributions in synthesized form must reproduce the above copyright
-- notice, this list of conditions and the following disclaimer in the
-- documentation and/or other materials provided with the distribution.
--
-- Neither the name of the author nor the names of other contributors may
-- be used to endorse or promote products derived from this software without
-- specific prior written permission.
--
-- THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
-- AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO,
-- THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
-- PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE
-- LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
-- CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
-- SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
-- INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
-- CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
-- ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
-- POSSIBILITY OF SUCH DAMAGE.
--
-- Please report bugs to the author, but before you do so, please
-- make sure that this is not a derivative work and that
-- you have the latest version of this file.
--
-- The latest version of this file can be found at:
--      http://www.opencores.org/cvsweb.shtml/t80/
--
-- Limitations :
--
-- File history :
--

library IEEE;
use IEEE.std_logic_1164.all;

package T80_Pack is

        component T80
        generic(
                Mode : integer := 0;    -- 0 => Z80, 1 => Fast Z80, 2 => 8080, 3 => GB
                IOWait : integer := 0; -- 1 => Single cycle I/O, 1 => Std I/O cycle
                Flag_C : integer := 0;
                Flag_N : integer := 1;
                Flag_P : integer := 2;
                Flag_X : integer := 3;
                Flag_H : integer := 4;
                Flag_Y : integer := 5;
                Flag_Z : integer := 6;
                Flag_S : integer := 7
        );
        port(
                RESET_n         : in std_logic;
                CLK_n           : in std_logic;
```

```
		CEN				: in std_logic;
		WAIT_n		: in std_logic;
		INT_n		: in std_logic;
		NMI_n		: in std_logic;
		BUSRQ_n		: in std_logic;
		M1_n		: out std_logic;
		IORQ		: out std_logic;
		NoRead		: out std_logic;
		Write_80	: out std_logic;
		RFSH_n		: out std_logic;
		HALT_n		: out std_logic;
		BUSAK_n		: out std_logic;
		A				: out std_logic_vector(15 downto 0);
		DInst		: in std_logic_vector(7 downto 0);
		DI				: in std_logic_vector(7 downto 0);
		DO				: out std_logic_vector(7 downto 0);
		MC				: out std_logic_vector(2 downto 0);
		TS				: out std_logic_vector(2 downto 0);
		IntCycle_n	: out std_logic;
		IntE		: out std_logic;
		Stop		: out std_logic
);
end component;

component T80_Reg
port(
		Clk				: in std_logic;
		CEN				: in std_logic;
		WEH				: in std_logic;
		WEL				: in std_logic;
		AddrA		: in std_logic_vector(2 downto 0);
		AddrB		: in std_logic_vector(2 downto 0);
		AddrC		: in std_logic_vector(2 downto 0);
		DIH				: in std_logic_vector(7 downto 0);
		DIL				: in std_logic_vector(7 downto 0);
		DOAH		: out std_logic_vector(7 downto 0);
		DOAL		: out std_logic_vector(7 downto 0);
		DOBH		: out std_logic_vector(7 downto 0);
		DOBL		: out std_logic_vector(7 downto 0);
		DOCH		: out std_logic_vector(7 downto 0);
		DOCL		: out std_logic_vector(7 downto 0)
);
end component;

component T80_MCode
generic(
		Mode : integer := 0;
		Flag_C : integer := 0;
		Flag_N : integer := 1;
		Flag_P : integer := 2;
		Flag_X : integer := 3;
		Flag_H : integer := 4;
		Flag_Y : integer := 5;
		Flag_Z : integer := 6;
		Flag_S : integer := 7
);
port(
		IR					: in std_logic_vector(7 downto 0);
		ISet				: in std_logic_vector(1 downto 0);
		MCycle				: in std_logic_vector(2 downto 0);
		F					: in std_logic_vector(7 downto 0);
		NMICycle		: in std_logic;
		IntCycle		: in std_logic;
		MCycles				: out std_logic_vector(2 downto 0);
		TStates				: out std_logic_vector(2 downto 0);
		Prefix				: out std_logic_vector(1 downto 0); -- None,BC,ED,DD/FD
		Inc_PC				: out std_logic;
		Inc_WZ				: out std_logic;
		IncDec_16		: out std_logic_vector(3 downto 0); -- BC,DE,HL,SP   0 is inc
		Read_To_Reg			: out std_logic;
```

```
		Read_To_Acc		: out std_logic;
		Set_BusA_To		: out std_logic_vector(3 downto 0); -- B,C,D,E,H,L,DI/DB,A,SP(L),SP(M),0,F
		Set_BusB_To		: out std_logic_vector(3 downto 0); -- B,C,D,E,H,L,DI,A,SP(L),SP(M),1,F,PC(L),PC(M),0
		ALU_Op			: out std_logic_vector(3 downto 0);
				-- ADD, ADC, SUB, SBC, AND, XOR, OR, CP, ROT, BIT, SET, RES, DAA, RLD, RRD, None
		Save_ALU		: out std_logic;
		PreserveC		: out std_logic;
		Arith16			: out std_logic;
		Set_Addr_To		: out std_logic_vector(2 downto 0); -- aNone,aXY,aIOA,aSP,aBC,aDE,aZI
		IORQ			: out std_logic;
		Jump			: out std_logic;
		JumpE			: out std_logic;
		JumpXY			: out std_logic;
		Call			: out std_logic;
		RstP			: out std_logic;
		LDZ				: out std_logic;
		LDW				: out std_logic;
		LDSPHL			: out std_logic;
		Special_LD		: out std_logic_vector(2 downto 0); -- A,I;A,R;I,A;R,A;None
		ExchangeDH		: out std_logic;
		ExchangeRp		: out std_logic;
		ExchangeAF		: out std_logic;
		ExchangeRS		: out std_logic;
		I_DJNZ			: out std_logic;
		I_CPL			: out std_logic;
		I_CCF			: out std_logic;
		I_SCF			: out std_logic;
		I_RETN			: out std_logic;
		I_BT			: out std_logic;
		I_BC			: out std_logic;
		I_BTR			: out std_logic;
		I_RLD			: out std_logic;
		I_RRD			: out std_logic;
		I_INRC			: out std_logic;
		SetDI			: out std_logic;
		SetEI			: out std_logic;
		IMode			: out std_logic_vector(1 downto 0);
		Halt			: out std_logic;
		NoRead			: out std_logic;
		Write_80		: out std_logic
	);
	end component;

	component T80_ALU
	generic(
		Mode : integer := 0;
		Flag_C : integer := 0;
		Flag_N : integer := 1;
		Flag_P : integer := 2;
		Flag_X : integer := 3;
		Flag_H : integer := 4;
		Flag_Y : integer := 5;
		Flag_Z : integer := 6;
		Flag_S : integer := 7
	);
	port(
		Arith16		: in std_logic;
		Z16			: in std_logic;
		ALU_Op		: in std_logic_vector(3 downto 0);
		IR			: in std_logic_vector(5 downto 0);
		ISet		: in std_logic_vector(1 downto 0);
		BusA		: in std_logic_vector(7 downto 0);
		BusB		: in std_logic_vector(7 downto 0);
		F_In		: in std_logic_vector(7 downto 0);
		Q			: out std_logic_vector(7 downto 0);
		F_Out		: out std_logic_vector(7 downto 0)
	);
	end component;

end;
```

## 8.6 T80se.vhd

```
--
-- Z80 compatible microprocessor core, synchronous top level with clock enable
-- Different timing than the original z80
-- Inputs needs to be synchronous and outputs may glitch
--
-- Version : 0242
--
-- Copyright (c) 2001-2002 Daniel Wallner (jesus@opencores.org)
--
-- All rights reserved
--
-- Redistribution and use in source and synthezised forms, with or without
-- modification, are permitted provided that the following conditions are met:
--
-- Redistributions of source code must retain the above copyright notice,
-- this list of conditions and the following disclaimer.
--
-- Redistributions in synthesized form must reproduce the above copyright
-- notice, this list of conditions and the following disclaimer in the
-- documentation and/or other materials provided with the distribution.
--
-- Neither the name of the author nor the names of other contributors may
-- be used to endorse or promote products derived from this software without
-- specific prior written permission.
--
-- THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
-- AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO,
-- THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
-- PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE
-- LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
-- CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
-- SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
-- INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
-- CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
-- ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
-- POSSIBILITY OF SUCH DAMAGE.
--
-- Please report bugs to the author, but before you do so, please
-- make sure that this is not a derivative work and that
-- you have the latest version of this file.
--
-- The latest version of this file can be found at:
--      http://www.opencores.org/cvsweb.shtml/t80/
--
-- Limitations :
--
-- File history :
--
--      0235 : First release
--
--      0236 : Added T2Write generic
--
--      0237 : Fixed T2Write with wait state
--
--      0238 : Updated for T80 interface change
--
--      0240 : Updated for T80 interface change
--
--      0242 : Updated for T80 interface change
--

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;
use work.T80_Pack.all;

entity T80se is
        generic(
```

```
		Mode : integer := 0;   -- 0 => Z80, 1 => Fast Z80, 2 => 8080, 3 => GB
		T2Write : integer := 0;		-- 0 => WR_n active in T3, /=0 => WR_n active in T2
		IOWait : integer := 1  -- 0 => Single cycle I/O, 1 => Std I/O cycle
	);
	port(
		RESET_n		: in std_logic;
		CLK_n		: in std_logic;
		CLKEN		: in std_logic;
		WAIT_n		: in std_logic;
		INT_n		: in std_logic;
		NMI_n		: in std_logic;
		BUSRQ_n		: in std_logic;
		M1_n		: out std_logic;
		MREQ_n		: out std_logic;
		IORQ_n		: out std_logic;
		RD_n		: out std_logic;
		WR_n		: out std_logic;
		RFSH_n		: out std_logic;
		HALT_n		: out std_logic;
		BUSAK_n		: out std_logic;
		A			: out std_logic_vector(15 downto 0);
		DI			: in std_logic_vector(7 downto 0);
		DO			: out std_logic_vector(7 downto 0)
	);
end T80se;

architecture rtl of T80se is

	signal IntCycle_n	: std_logic;
	signal NoRead		: std_logic;
	signal Write_80		: std_logic;
	signal IORQ			: std_logic;
	signal DI_Reg		: std_logic_vector(7 downto 0);
	signal MCycle		: std_logic_vector(2 downto 0);
	signal TState		: std_logic_vector(2 downto 0);

begin

	u0 : T80
		generic map(
			Mode => Mode,
			IOWait => IOWait)
		port map(
			CEN => CLKEN,
			M1_n => M1_n,
			IORQ => IORQ,
			NoRead => NoRead,
			Write_80 => Write_80,
			RFSH_n => RFSH_n,
			HALT_n => HALT_n,
			WAIT_n => Wait_n,
			INT_n => INT_n,
			NMI_n => NMI_n,
			RESET_n => RESET_n,
			BUSRQ_n => BUSRQ_n,
			BUSAK_n => BUSAK_n,
			CLK_n => CLK_n,
			A => A,
			DInst => DI,
			DI => DI_Reg,
			DO => DO,
			MC => MCycle,
			TS => TState,
			IntCycle_n => IntCycle_n);

	process (RESET_n, CLK_n)
	begin
		if RESET_n = '0' then
			RD_n <= '1';
			WR_n <= '1';
```

```
                IORQ_n <= '1';
                MREQ_n <= '1';
                DI_Reg <= "00000000";
            elsif CLK_n'event and CLK_n = '1' then
                if CLKEN = '1' then
                    RD_n <= '1';
                    WR_n <= '1';
                    IORQ_n <= '1';
                    MREQ_n <= '1';
                    if MCycle = "001" then
                        if TState = "001" or (TState = "010" and Wait_n = '0') then
                            RD_n <= not IntCycle_n;
                            MREQ_n <= not IntCycle_n;
                            IORQ_n <= IntCycle_n;
                        end if;
                        if TState = "011" then
                            MREQ_n <= '0';
                        end if;
                    else
                        if (TState = "001" or (TState = "010" and Wait_n = '0')) and NoRead = '0' and Write_80 = '0' then
                            RD_n <= '0';
                            IORQ_n <= not IORQ;
                            MREQ_n <= IORQ;
                        end if;
                        if T2Write = 0 then
                            if TState = "010" and Write_80 = '1' then
                                WR_n <= '0';
                                IORQ_n <= not IORQ;
                                MREQ_n <= IORQ;
                            end if;
                        else
                            if (TState = "001" or (TState = "010" and Wait_n = '0')) and Write_80 = '1' then
                                WR_n <= '0';
                                IORQ_n <= not IORQ;
                                MREQ_n <= IORQ;
                            end if;
                        end if;
                    end if;
                    if TState = "010" and Wait_n = '1' then
                        DI_Reg <= DI;
                    end if;
                end if;
            end if;
        end process;

end;
```

## 8.7 Decodificador 7 segmentos

```
LIBRARY IEEE;
USE  IEEE.STD_LOGIC_1164.all;
USE  IEEE.STD_LOGIC_UNSIGNED.all;

entity decoder_7seg is
        port
        (
                NUMBER          : in   std_logic_vector(3 downto 0);
                HEX_DISP        : out  std_logic_vector(6 downto 0)
        );
end decoder_7seg;

architecture rtl of decoder_7seg is
begin
process(NUMBER)
begin
        case NUMBER is
                --0 to 9
                when "0000" => HEX_DISP <= "1000000";
                when "0001" => HEX_DISP <= "1111001";
                when "0010" => HEX_DISP <= "0100100";
                when "0011" => HEX_DISP <= "0110000";
                when "0100" => HEX_DISP <= "0011001";
                when "0101" => HEX_DISP <= "0010010";
                when "0110" => HEX_DISP <= "0000010";
                when "0111" => HEX_DISP <= "1111000";
                when "1000" => HEX_DISP <= "0000000";
                when "1001" => HEX_DISP <= "0011000";
                -- A to F
                when "1010" => HEX_DISP <= "0001000";
                when "1011" => HEX_DISP <= "0000011";
                when "1100" => HEX_DISP <= "1000110";
                when "1101" => HEX_DISP <= "0100001";
                when "1110" => HEX_DISP <= "0000110";
                when "1111" => HEX_DISP <= "0001110";
                when others => HEX_DISP <= "1111111";
        end case;
end process;
end rtl;
```

## 8.8 Memoria RAM de 2K

```
--
-- Inferrable Synchronous SRAM for XST synthesis
--
-- Version : 0220
--
-- Copyright (c) 2002 Daniel Wallner (jesus@opencores.org)
--
-- All rights reserved
--
-- Redistribution and use in source and synthezised forms, with or without
-- modification, are permitted provided that the following conditions are met:
--
-- Redistributions of source code must retain the above copyright notice,
-- this list of conditions and the following disclaimer.
--
-- Redistributions in synthesized form must reproduce the above copyright
-- notice, this list of conditions and the following disclaimer in the
-- documentation and/or other materials provided with the distribution.
--
-- Neither the name of the author nor the names of other contributors may
-- be used to endorse or promote products derived from this software without
-- specific prior written permission.
--
-- THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
-- AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO,
-- THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
-- PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE
-- LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
-- CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
-- SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
-- INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
-- CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
-- ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
-- POSSIBILITY OF SUCH DAMAGE.
--
-- Please report bugs to the author, but before you do so, please
-- make sure that this is not a derivative work and that
-- you have the latest version of this file.
--
-- The latest version of this file can be found at:
--      http://www.opencores.org/cvsweb.shtml/t51/
--
-- Limitations :
--
-- File history :
--      0208 : Initial release
--      0218 : Fixed data out at write
--      0220 : Added support for XST

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity SSRAM is
        generic(
                AddrWidth       : integer := 11;
                DataWidth: integer := 8
        );
        port(
                Clk                     : in std_logic;
                CE_n            : in std_logic;
                WE_n            : in std_logic;
                A                       : in std_logic_vector(AddrWidth - 1 downto 0);
                DIn                     : in std_logic_vector(DataWidth - 1 downto 0);
```

```
		DOut		: out std_logic_vector(DataWidth - 1 downto 0)
	);
end SSRAM;

architecture behaviour of SSRAM is

	type Memory_Image is array (natural range <>) of std_logic_vector(DataWidth - 1 downto 0);
	signal	RAM		: Memory_Image(0 to 2 ** AddrWidth - 1);
	signal	A_r		: std_logic_vector(AddrWidth - 1 downto 0);

begin

	process (Clk)
	begin
		if Clk'event and Clk = '1' then
			if (CE_n nor WE_n) = '1' then
				RAM(to_integer(unsigned(A))) <= DIn;
			end if;
			A_r <= A;
		end if;
	end process;

	DOut <= RAM(to_integer(unsigned(A_r)))
-- pragma translate_off
			when not is_x(A_r) else (others => '-')
-- pragma translate_on
	;
end;
```

## 8.9 cabecera_00.asm

```
	org	0x0000


cero	equ	0x00

p0	equ	0x00
p1	equ	0x01
p2	equ	0x02
p3	equ	0x03
p4	equ	0x04
p5	equ	0x05
p6	equ	0x06
p7	equ	0x07
p8	equ	0x08
p9	equ	0x09


	ld	hl,0xfff0
	ld	sp,hl


	ld	a,0x00
	ld	b,0x00
	ld	c,0x00
	ld	d,0x00
	ld	e,0x00
	ld	h,0x00
	ld	l,0x00
	call	seg
	jp	final
```

```
seg:
	out	(p7),a
	ld	a,b
	out	(p0),a
	ld	a,c
	out	(p1),a
	ld	a,d
	out	(p2),a
	ld	a,e
	out	(p3),a
	ld	a,h
	out	(p4),a
	ld	a,l
	out	(p5),a

	ld	a,0x00
	out	(p6),a
	ld	a,0x00
	out	(p8),a
	ret
final:
	nop
	nop
	nop
	nop
	nop
	nop
	nop
	nop
	nop
	nop
	nop
```