Jocelyn miranda y lino ontano

Escuela superior politécnica del litoral

Facultad de Ingeniería en Electricidad y Computación

sistema de gestión de base de datos en la tarjeta de10 standard  
MANUAL TÉCNICO

Diseño de sistemas digitales

4 de septiembre de 2019

# OBJETIVOS

## General

Realizar consultas desde la FPGA hacia el servidor de base de datos implementado en la parte del Hard Processor System de la tarjeta de desarrollo DE10 Standard.

## Específicos

* Diseñar la arquitectura digital del sistema.
* Realizar la conexión entre las componentes a utilizar en el archivo Qsys.
* Configurar el servidor de base de datos en el HPS.
* Establecer el direccionamiento en el sistema operativo Linux.
* Crear archivos ejecutables para realizar las consultas a la base de datos.

# DESCRIPCIÓN DEL PROYECTO

El sistema constará de diseñar e implementar un sistema que realice monitoreo de consultas a una base de datos. El servidor de la base de datos se la implementará en el Hard-Processor System (HPS) de la FPGA DE-10 Standard, la cuál será manejada vía Putty SSH conectada a una red local. La FPGA será la encargada de realizar las consultas al HPS el cual responderá y la FPGA mostrará el resultado por pantalla por medio de su puerto VGA. El proyecto se lo desarrollará en tres etapas: La primera etapa constará de montar el servidor de la base de datos en la HPS y poder acceder a ella vía Putty SSH, la imagen del sistema operativo será descargada desde la página de Terasic 1 y donde se levantará el servicio de base de datos MySQL. La segunda etapa será poder mostrar al servidor por pantalla utilizando la FPGA, donde el procesador NIOS manejará las respuestas del servidor para ser mostradas y procesadas. Y la última etapa será de poder, desde la FPGA, solicitar consultas al HPS y recibir las mismas respuestas, es decir una comunicación bidireccional.

.

# DISEÑO DE LA ARQUITECTURA

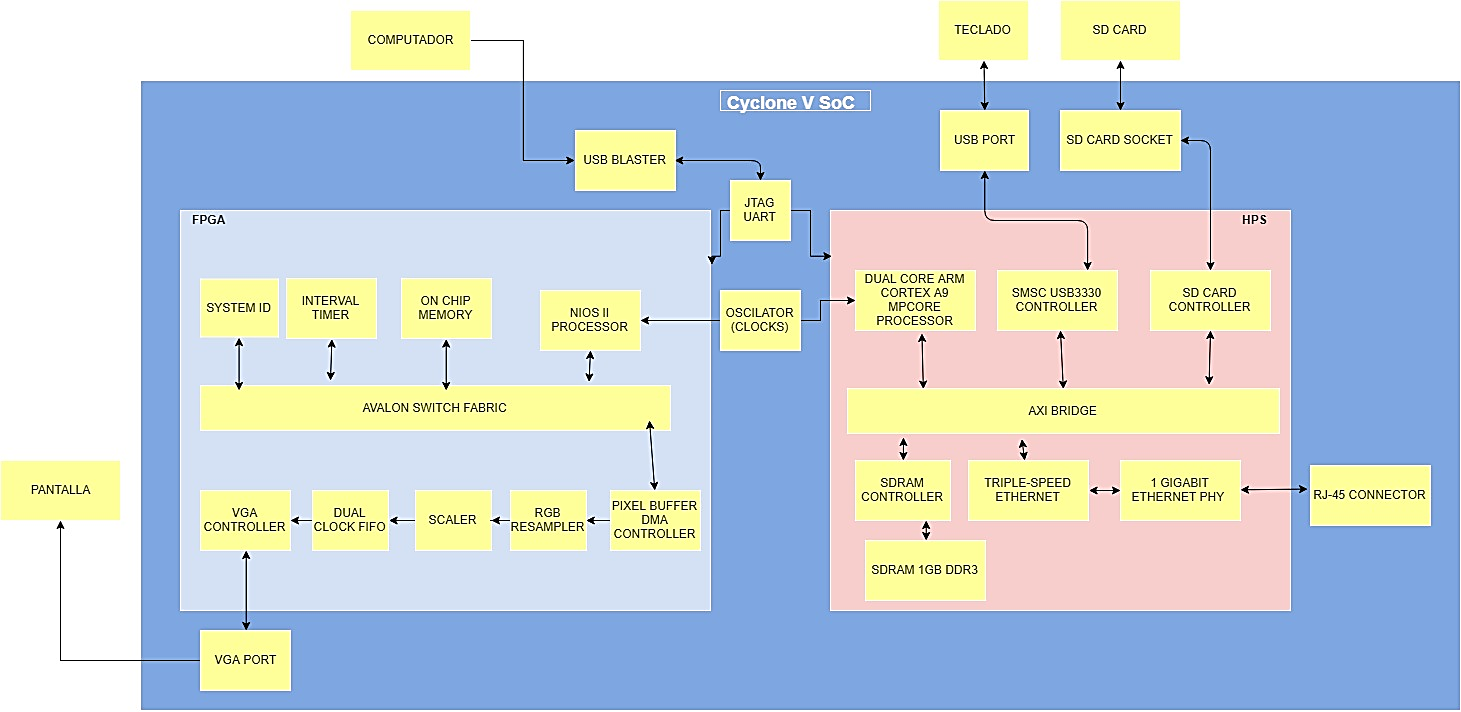


Ilustración - Arquitectura

## Descripción de cada bloque

**AVALON:** Permite la comunicación entre los diferentes bloques presentes en el sistema.

**SDRAM CONTROLLER:** Permite al procesador almacenar datos para su manipulación.

**SDRAM 1GB DDR3:** Dada la cantidad de peticiones que el servidor tendrá, se consideró este bloque ya que ejecuta instrucciones con mayor rapidez, esto permite que mejore la tasa de transferencia de datos.

**SD CARD CONTROLLER:** Recibe las señales dadas por el convertidor analógico/digital las cuales son procesadas en el interior del Core. El Core genera la resolución para la pantalla. Guardará los datos que reciba de los sensores en una parte de la memoria.

**SD CARD SOCKET:** Puerto para el intercambio del flujo de información relacionado con el servidor Linux.

**SD CARD:** Tarjeta Micro SD booteable con el sistema Linux.

**TRIPLE- SPEED ETHERNET:** Contiene múltiples velocidades semejantes a PHY.

**1GB ETHERNET PHY:** Interfaz Ethernet por la cual se conectará el dispositivo a un enrutador para su manejo por PUTTY.

**RJ-45 CONNECTOR:** Para la conexión a la red.

**DUAL CORE ARM CORTEX A9 MPCORE PROCESSOR:** Se encargará de la parte del servidor de base de datos.

**JTAG UART:** Permite la programación del procesador y la creación del sistema embebido en la tarjeta de desarrollo.

**USB BLASTER:** Puerto USB de la tarjeta FPGA.

**VGA CONTROLLER:** Sincroniza la salida de las tramas para correcta reproducción a través de una pantalla conectada por VGA.

**VGA PORT:** Para la conexión de la pantalla con la tarjeta de video

**NIOS II PROCESSOR:** Se encarga de preparar las tramas para la salida por medio del protocolo VGA.

**ON CHIP MEMORY:** Va de la mano con el Procesador NIOS II. El núcleo de memoria FIFO en el chip es un componente configurable utilizado para almacenar datos y proporcionar control de flujo en un sistema SOPC Builder. El FIFO puede operar con un solo reloj o con relojes separados para los puertos de entrada y salida. El FIFO en chip. El núcleo de memoria no es compatible con ráfaga de lectura o escritura.

**INTERNAL TIMER:** Controla los intervalos de tiempo de todo el sistema.

**SYSTEM ID:** Se encarga de verificar que un programa ejecutable sea compilado con la imagen del Hardware actual configurado en la tarjeta de desarrollo FPGA.

# DESARROLLO

## Instalación del Sistema operativo Linux en una MicroSD

1. Descargar la imagen Linux LXDE Desktop de la tarjeta DE10 Standard desde la página de Terasic[[1]](#footnote-1).

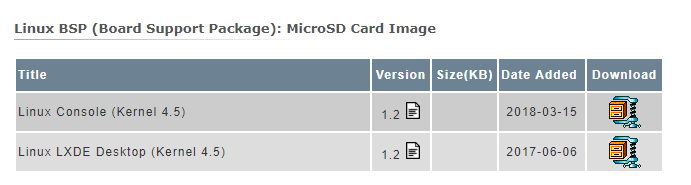


Ilustración - Terasic - Linux

1. Descargue e instale el software *Win32 Disk Imager*.
2. Conecte la *MICROSD* al ordenador.
3. Abra el software *Win32 Disk Imager*. Seleccione el archivo de la imagen descargada y el dispositivo donde deseamos escribir. Clic en *Write.*

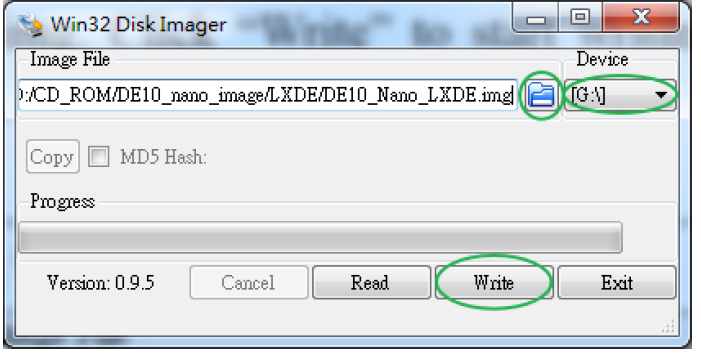


Ilustración - Grabar la imagen en una MICROSD

## Configurar arquitectura

1. Abrir el archivo DE10\_Standard\_FB.qpf alojado en la carpeta *SocFPGA/ControlPanel/Quartus* de los archivos del CD.
2. Utilizamos el proyecto por defecto que tiene configurado la tarjeta DE10 Standard “ControlPanel”.

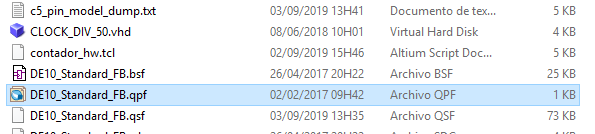


Ilustración - Archivo qpf Control Panel

1. Creamos un archivo VHDL para el contador de segundos, File -> New -> VHDL File.

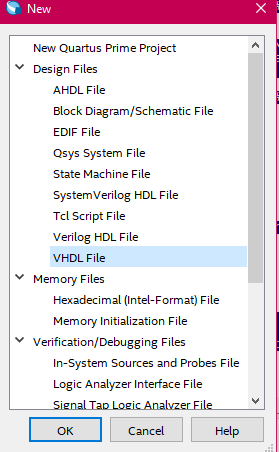


Ilustración - Crear archivo VHDL

1. Copiamos el siguiente código en el mismo.

########################################################################-- Contador.vhd

-- This component describes a simple counter UP-DOWN with an Avalon-MM slave interface. The Input Signals can be written to registers 0 and 1, and the outputs of the counter can be read back from registers 2 and 3.

#######################################################################

library ieee;

use ieee.std\_logic\_1164.all;

use ieee.std\_logic\_UNSIGNED.all;

use ieee.std\_logic\_Arith.all;

ENTITY bloque IS

PORT(

-- Avalon Clock interface

clk1,clk2 : in std\_logic; --clk1 for Avalon and clk2 for Counter

-- Avalon Reset interface

reset : in std\_logic;

-- Avalon-MM Slave interface

address : in std\_logic\_vector(1 downto 0);

read : in std\_logic;

write : in std\_logic;

readdata : out std\_logic\_vector(31 downto 0);

writedata : in std\_logic\_vector(31 downto 0);

counter\_conduit: out std\_logic\_vector(9 downto 0)

);

END bloque;

ARCHITECTURE sol OF bloque IS

--Address for different registers

constant REG\_INPUT\_1\_OFST : std\_logic\_vector(1 downto 0) := "00";--Address for Cargar, habilcnt and descendente

constant REG\_INPUT\_2\_OFST : std\_logic\_vector(1 downto 0) := "01";--Address for dato\_ent

constant REG\_OUTPUT\_1\_OFST : std\_logic\_vector(1 downto 0) := "10";--Address for Q

constant REG\_OUTPUT\_2\_OFST : std\_logic\_vector(1 downto 0) := "11";--Address for ct\_term

signal reg\_input\_1 : unsigned(writedata'range); --creating register 1 for Inputs -> Cargar, habilcnt and descendente

signal reg\_input\_2 : unsigned(writedata'range); --creating register 2 for input -> dato\_ent

signal reg\_output\_1 : std\_logic\_vector(31 downto 0); --creating register 3 for Output -> Q

signal reg\_output\_2 : unsigned(readdata'range); --creating register 4 for Output -> ct\_term

SIGNAL cargar, habilcnt, descendente, ct\_term : STD\_LOGIC; -- Input Signals for the Counter Process

SIGNAL dato\_ent,Q : STD\_LOGIC\_VECTOR (9 downto 0); -- Input and Output Signals for the Counter Process

SIGNAL conteo: STD\_LOGIC\_VECTOR(9 downto 0); -- define a 4 bits Bus

BEGIN

-- Avalon-MM slave write

process(clk1, reset)

begin

if reset = '1' then

reg\_input\_1 <= (others => '0');

reg\_input\_2 <= (others => '0');

elsif rising\_edge(clk1) then

if write = '1' then

case address is

when REG\_INPUT\_1\_OFST => reg\_input\_1 <= unsigned(writedata);

when REG\_INPUT\_2\_OFST => reg\_input\_2 <= unsigned(writedata);

-- RESULT register is read-only

when REG\_OUTPUT\_1\_OFST => null;

when REG\_OUTPUT\_2\_OFST => null;

-- Remaining addresses in register map are unused.

when others => null;

end case;

end if;

end if;

end process;

-- Avalon-MM slave read

process(clk1, reset)

begin

reg\_output\_1(9 downto 0)<=Q;reg\_output\_2(0)<=ct\_term;

if rising\_edge(clk1) then

if read = '1' then

case address is

when REG\_INPUT\_1\_OFST => readdata <= std\_logic\_vector(reg\_input\_1); --assign readdata<=Cargar&habilcnt&descendente

when REG\_INPUT\_2\_OFST => readdata <= std\_logic\_vector(reg\_input\_2); --assign readdata<=data\_ent

when REG\_OUTPUT\_1\_OFST => readdata <= reg\_output\_1; --assign readdata<=Q

when REG\_OUTPUT\_2\_OFST => readdata <= std\_logic\_vector(reg\_output\_2);--assign readdata<=ct\_term

-- Remaining addresses in register map are unmapped => return 0.

when others => readdata <= (others => '0');

end case;

end if;

end if;

end process;

--Counter Process

PROCESS(clk2,reset,descendente)

BEGIN

cargar<=reg\_input\_1(2);habilcnt<=reg\_input\_1(1);descendente<=reg\_input\_1(2);

dato\_ent<=conv\_std\_logic\_vector(reg\_input\_2,10);

if reset='1' then conteo<="0000000000"; -- borrar asÃ­ncrona

elsif (clk2'event and clk2='1') then -- flanco ascendente?

if cargar='1' then conteo<=dato\_ent; --carga en paralelo

elsif habilcnt='1' then -- habilitado?

if descendente='0' then conteo<=conteo+1; --incremento

else conteo<=conteo-1; --decremento

end if;

end if;

end if;

if (((conteo="0000000000" and descendente='1')) OR ((conteo="1111111111" and descendente='0'))) AND habilcnt='1'

then ct\_term<='1';

else ct\_term<='0';

end if;

q<=conteo; --transfer the content from register to output

END PROCESS;

counter\_conduit<=Q; --transfer the content from register Q to output Conduit

END sol;

1. Creamos un archivo VHDL para el divisor de frecuencia, File -> New -> VHDL File (Ilustración 3).
2. Copiamos el siguiente código.

LIBRARY IEEE;

USE IEEE.STD\_LOGIC\_1164.all;

USE IEEE.STD\_LOGIC\_ARITH.all;

USE IEEE.STD\_LOGIC\_UNSIGNED.all;

ENTITY CLOCK\_DIV\_50 IS

PORT

( CLOCK\_50MHz :IN STD\_LOGIC;

CLOCK\_1MHz :OUT STD\_LOGIC;

CLOCK\_100KHz :OUT STD\_LOGIC;

CLOCK\_10KHz :OUT STD\_LOGIC;

CLOCK\_1KHz :OUT STD\_LOGIC;

CLOCK\_100Hz :OUT STD\_LOGIC;

CLOCK\_10Hz :OUT STD\_LOGIC;

CLOCK\_1Hz :OUT STD\_LOGIC);

END CLOCK\_DIV\_50;

ARCHITECTURE a OF CLOCK\_DIV\_50 IS

SIGNAL count\_1Mhz: STD\_LOGIC\_VECTOR(5 DOWNTO 0);

SIGNAL count\_100Khz, count\_10Khz, count\_1Khz: STD\_LOGIC\_VECTOR(2 DOWNTO 0);

SIGNAL count\_100hz, count\_10hz, count\_1hz: STD\_LOGIC\_VECTOR(2 DOWNTO 0);

SIGNAL clock\_1Mhz\_int, clock\_100Khz\_int, clock\_10Khz\_int, clock\_1Khz\_int: STD\_LOGIC;

SIGNAL clock\_100hz\_int, clock\_10hz\_int, clock\_1hz\_int: STD\_LOGIC;

BEGIN

PROCESS

BEGIN

-- Divide by 50

WAIT UNTIL clock\_50Mhz'EVENT and clock\_50Mhz = '1'; -- 24 Mhz

IF count\_1Mhz < 49 THEN

count\_1Mhz <= count\_1Mhz + 1;

ELSE

count\_1Mhz <= "000000";

END IF;

IF count\_1Mhz < 4 THEN

clock\_1Mhz\_int <= '0';

ELSE

clock\_1Mhz\_int <= '1';

END IF;

-- Ripple clocks are used in this code to save prescalar hardware

-- Sync all clock prescalar outputs back to master clock signal

clock\_1Mhz <= clock\_1Mhz\_int;

clock\_100Khz <= clock\_100Khz\_int;

clock\_10Khz <= clock\_10Khz\_int;

clock\_1Khz <= clock\_1Khz\_int;

clock\_100hz <= clock\_100hz\_int;

clock\_10hz <= clock\_10hz\_int;

clock\_1hz <= clock\_1hz\_int;

END PROCESS;

-- Divide by 10

PROCESS

BEGIN

WAIT UNTIL clock\_1Mhz\_int'EVENT and clock\_1Mhz\_int = '1';

IF count\_100Khz /= 4 THEN

count\_100Khz <= count\_100Khz + 1;

ELSE

count\_100Khz <= "000";

clock\_100Khz\_int <= NOT clock\_100Khz\_int;

END IF;

END PROCESS;

-- Divide by 10

PROCESS

BEGIN

WAIT UNTIL clock\_100Khz\_int'EVENT and clock\_100Khz\_int = '1';

IF count\_10Khz /= 4 THEN

count\_10Khz <= count\_10Khz + 1;

ELSE

count\_10Khz <= "000";

clock\_10Khz\_int <= NOT clock\_10Khz\_int;

END IF;

END PROCESS;

-- Divide by 10

PROCESS

BEGIN

WAIT UNTIL clock\_10Khz\_int'EVENT and clock\_10Khz\_int = '1';

IF count\_1Khz /= 4 THEN

count\_1Khz <= count\_1Khz + 1;

ELSE

count\_1Khz <= "000";

clock\_1Khz\_int <= NOT clock\_1Khz\_int;

END IF;

END PROCESS;

-- Divide by 10

PROCESS

BEGIN

WAIT UNTIL clock\_1Khz\_int'EVENT and clock\_1Khz\_int = '1';

IF count\_100hz /= 4 THEN

count\_100hz <= count\_100hz + 1;

ELSE

count\_100hz <= "000";

clock\_100hz\_int <= NOT clock\_100hz\_int;

END IF;

END PROCESS;

-- Divide by 10

PROCESS

BEGIN

WAIT UNTIL clock\_100hz\_int'EVENT and clock\_100hz\_int = '1';

IF count\_10hz /= 4 THEN

count\_10hz <= count\_10hz + 1;

ELSE

count\_10hz <= "000";

clock\_10hz\_int <= NOT clock\_10hz\_int;

END IF;

END PROCESS;

-- Divide by 10

PROCESS

BEGIN

WAIT UNTIL clock\_10hz\_int'EVENT and clock\_10hz\_int = '1';

IF count\_1hz /= 4 THEN

count\_1hz <= count\_1hz + 1;

ELSE

count\_1hz <= "000";

clock\_1hz\_int <= NOT clock\_1hz\_int;

END IF;

END PROCESS;

END a;

1. Damos clic derecho en el archivo VHDL del contador y seleccionamos la opción *Set as Top-Level Entity.*

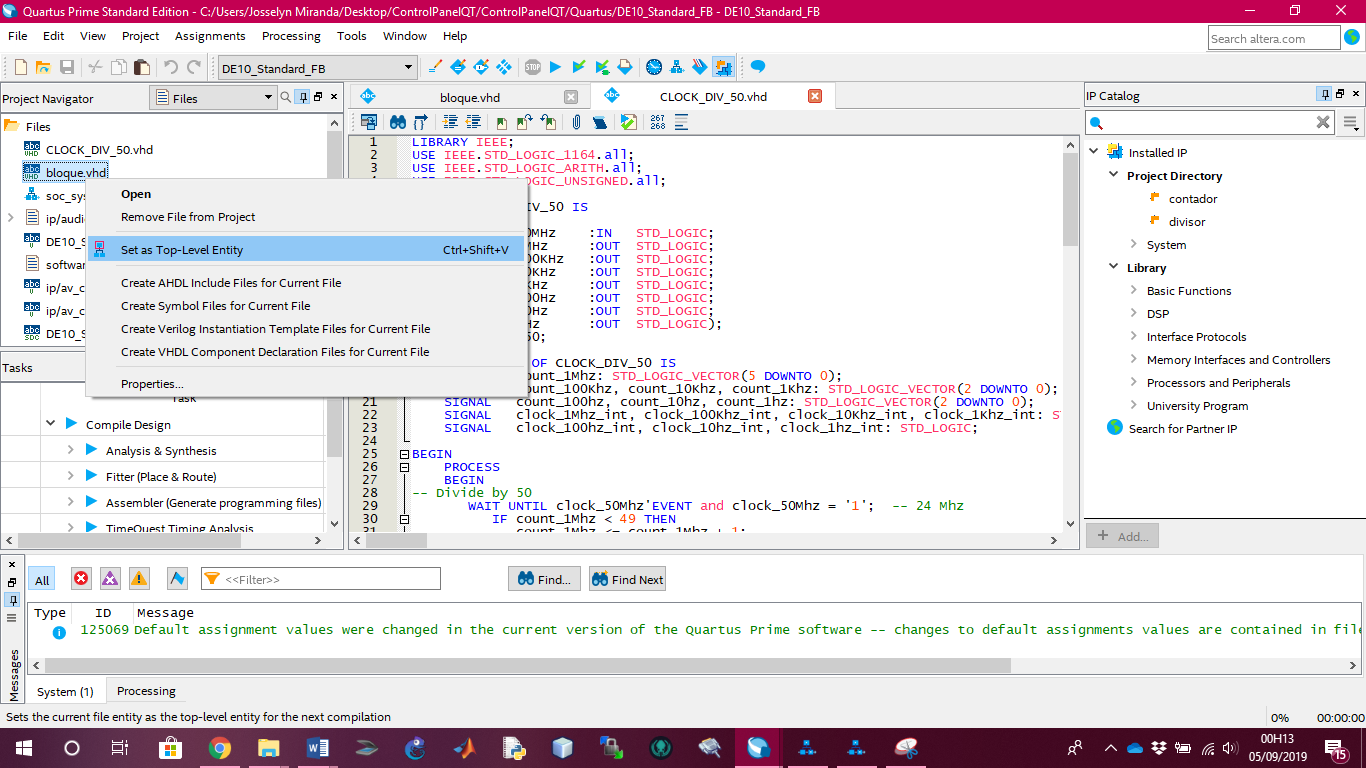


Ilustración - Archivo de alta prioridad

1. Damos clic en  para empezar la compilación del archivo contador.
2. Repetimos el paso 6 y 7 pero esta vez para el archivo VHDL del divisor de frecuencia.
3. Luego de la compilación exitosa (100%). Seleccionamos *Files* en el *Project Navigator.* Damos doble clic en el archivo *soc\_system.qsys* donde construiremos nuestra arquitectura.

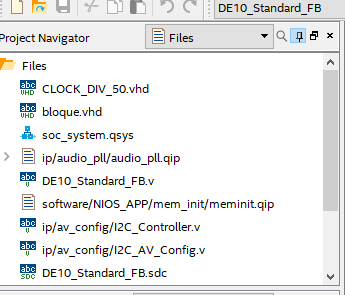


Ilustración - Proyject navigator

1. Dentro de la ventana Qsys observamos todos los componentes utilizados por el proyecto base Control Panel. Sin embargo, nuestra arquitectura no utiliza todo ello, por lo que procedemos a eliminar los siguientes componentes: vga\_controller, TERASIC\_ALSA, tv\_decoder, vga\_stream y el resto de componentes relacionados a VGA.
2. Creamos las componentes del *CONTADOR* y *CLOCK50*. Doble clic en *New Component* del *catálogo IP.*

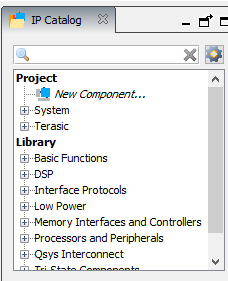


Ilustración - Crear componentes

1. Seleccionamos *Add File*  y escogemos el archivo del bloque *contador.vhd.* Luego haga clic en A*nalyze Synthesis Files*. Clic en ok.

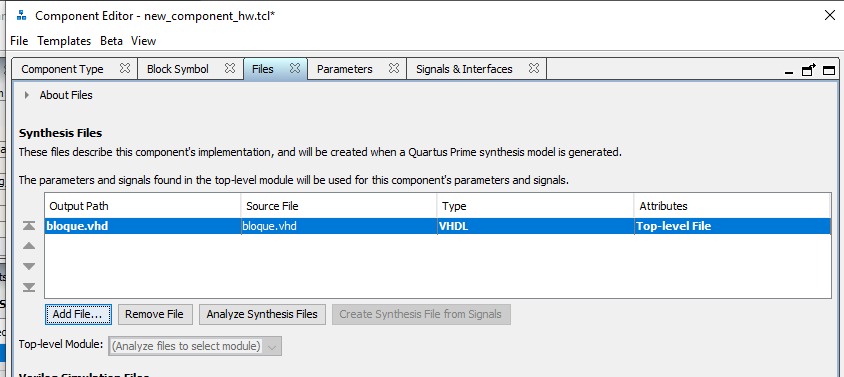
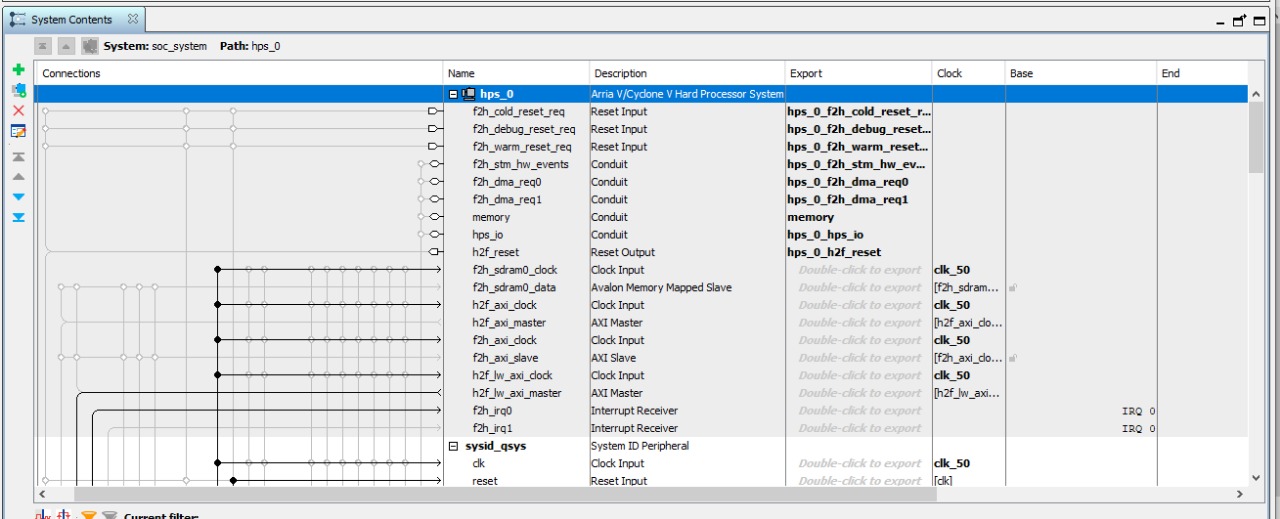


Ilustración - Add file vhd

1. Repita los pasos anteriores para crear el componente del divisor de frecuencia usando el archivo VHDL respectivo.
2. Una vez agregados todos los bloques faltantes a Qsys empezar a realizar la conexión entre ellos tal como indica la Ilustración 8 - Conexiones Qsys.



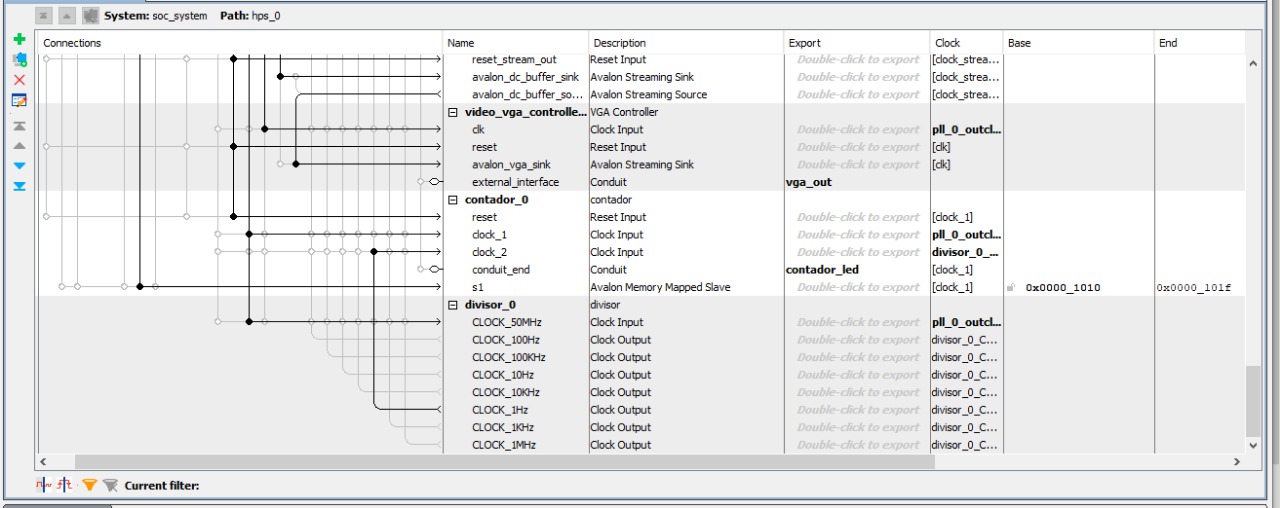
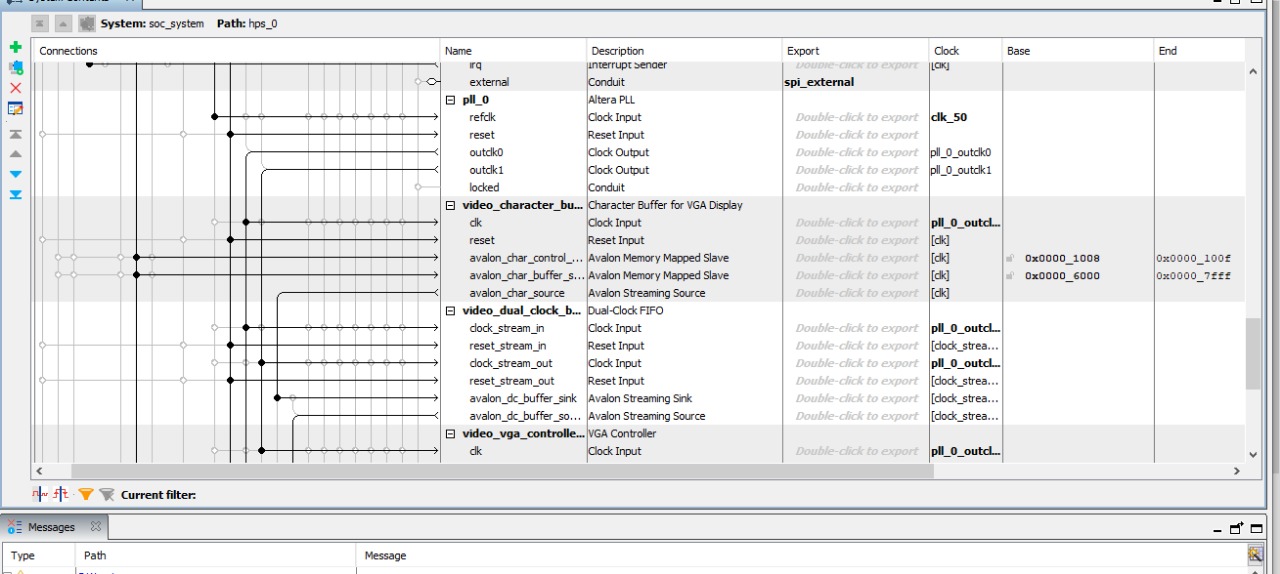
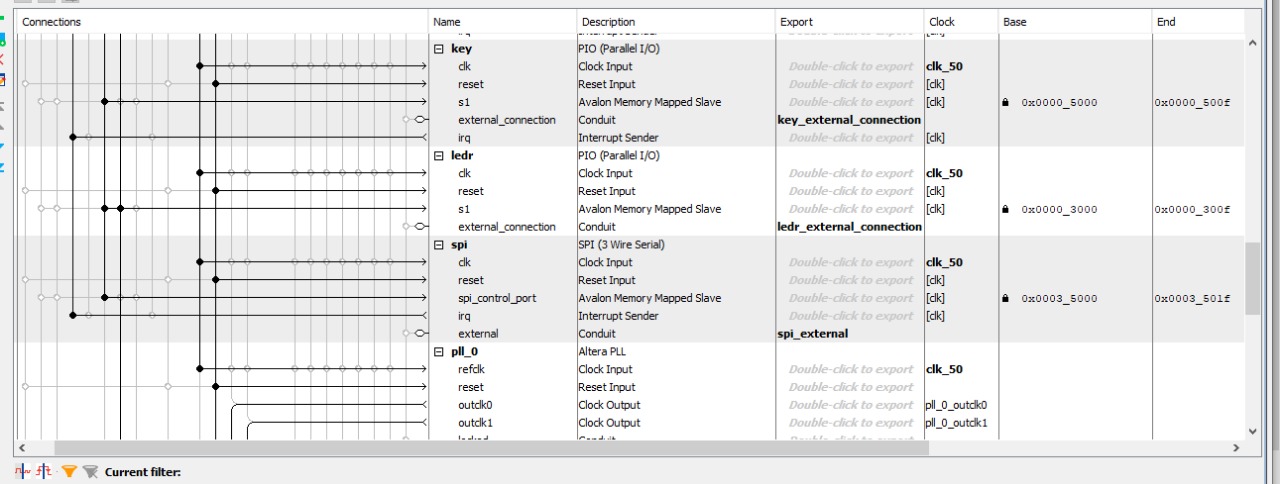
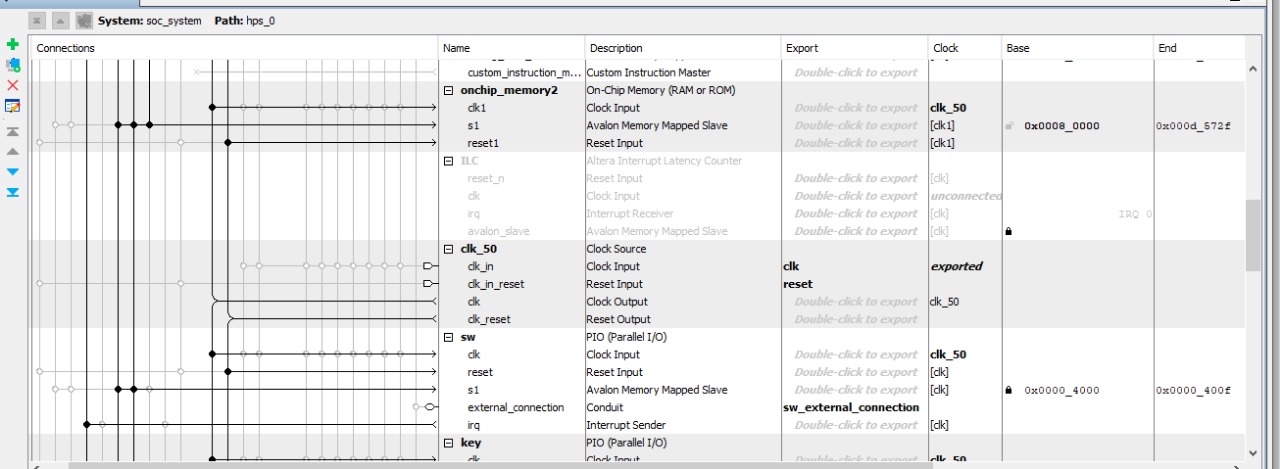
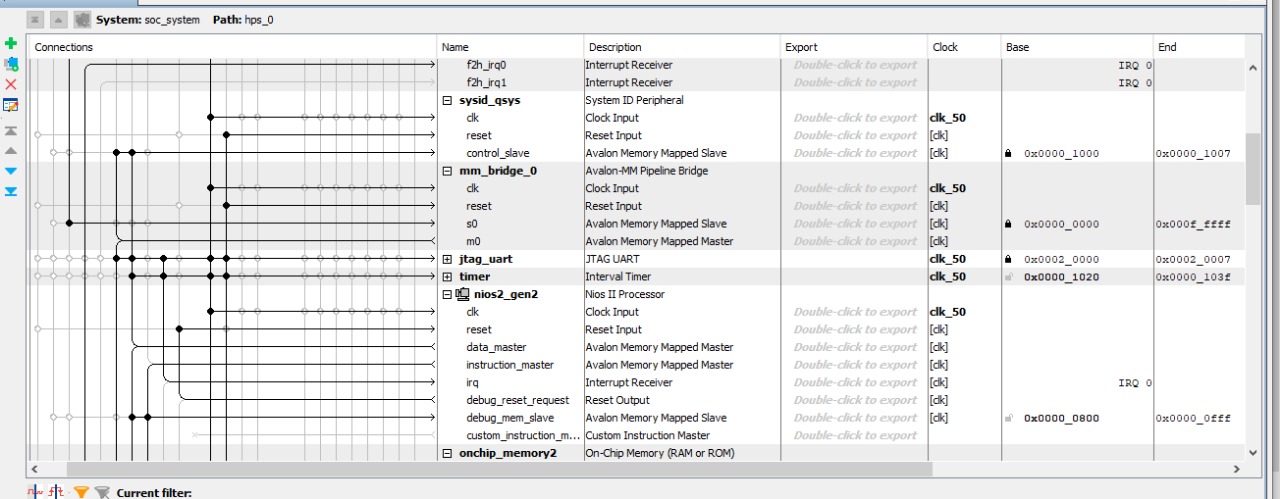


Ilustración - Conexiones Qsys

1. Después de realizar las conexiones entre componentes asignamos Base Address desde la pestaña System – Assign Base Address. Como indica la Ilustración 7.

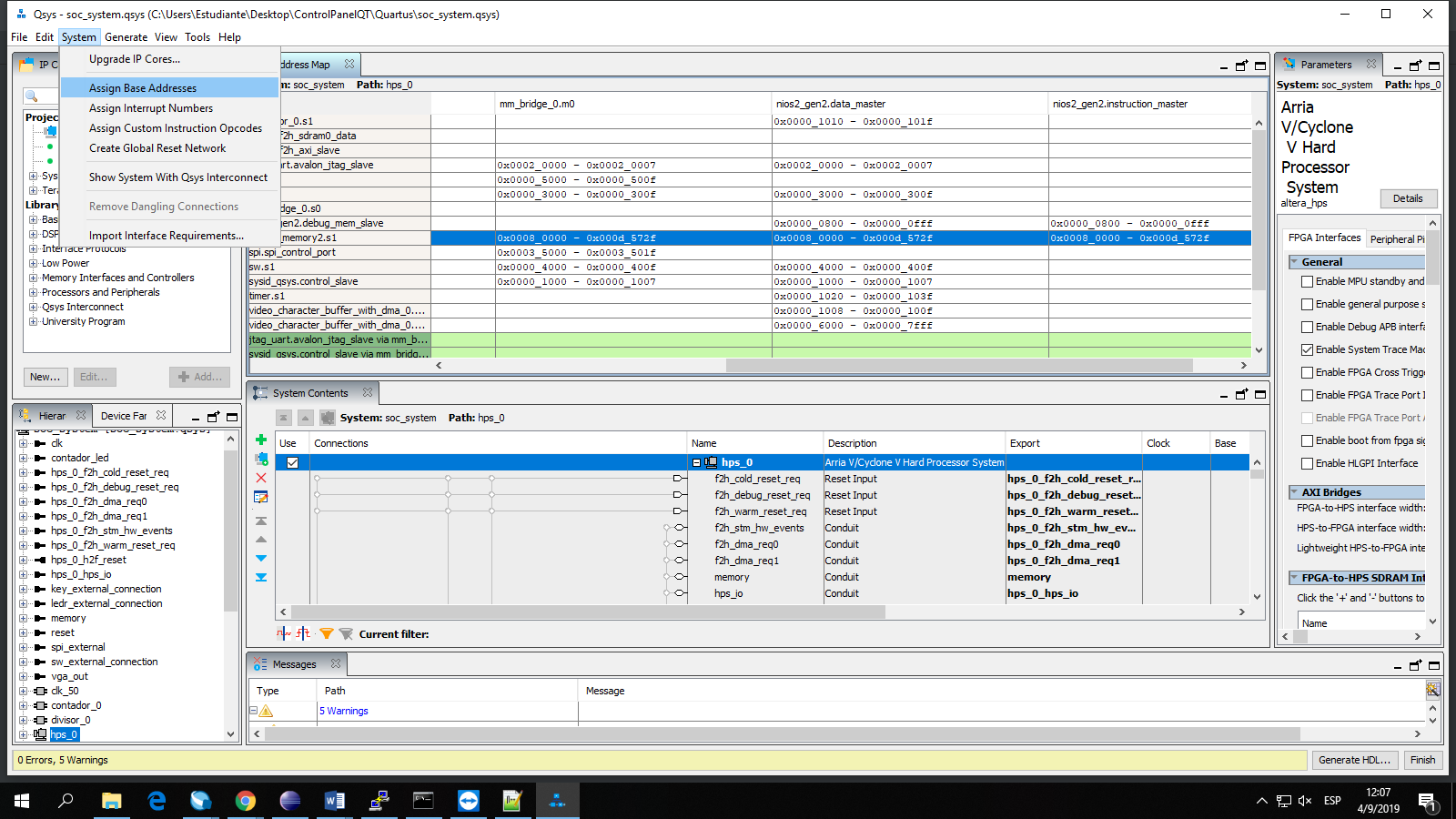


Ilustración - Asignar direcciones

1. Presionar el botón *Generar HDL* para actualizar el *soc\_system.qsys* con los cambios realizados.
2. Ver la instanciación del componente de *soc\_system u0*. Seleccionar la opción *Generate -> Show Instantiation Template.*

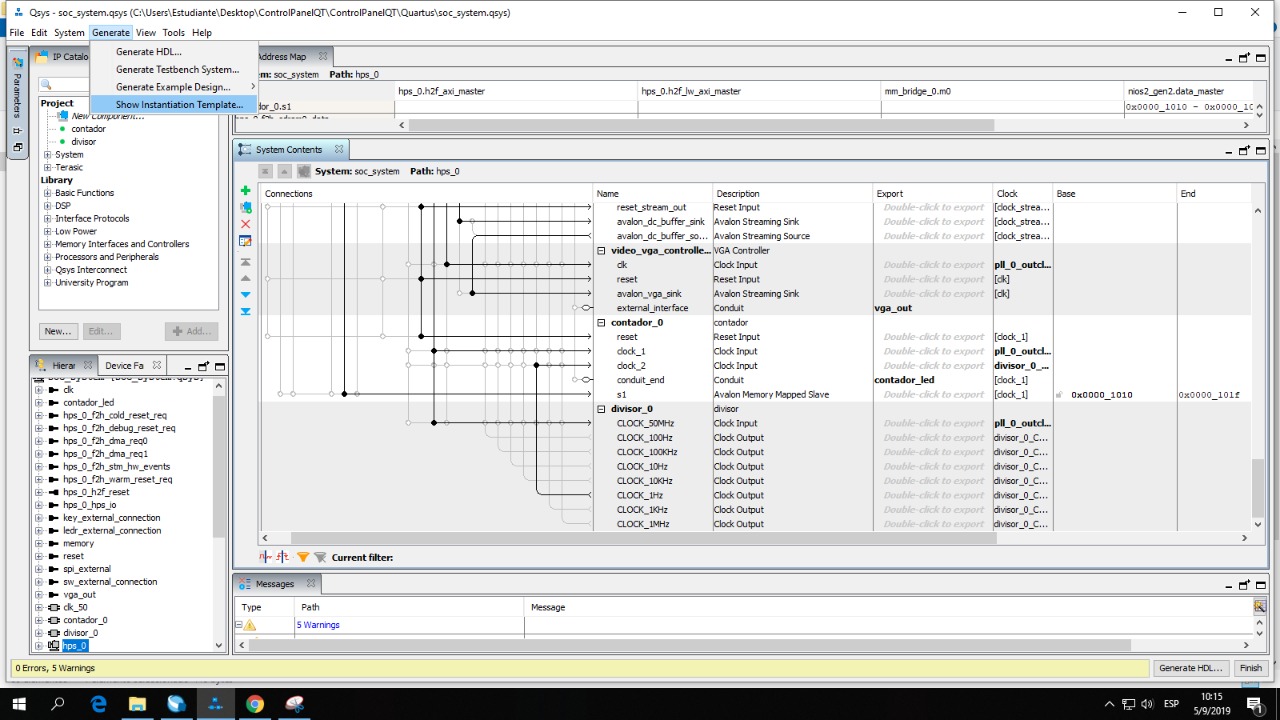


Ilustración - Instanciación

1. Clic en *Copy* para copiar el texto generado.

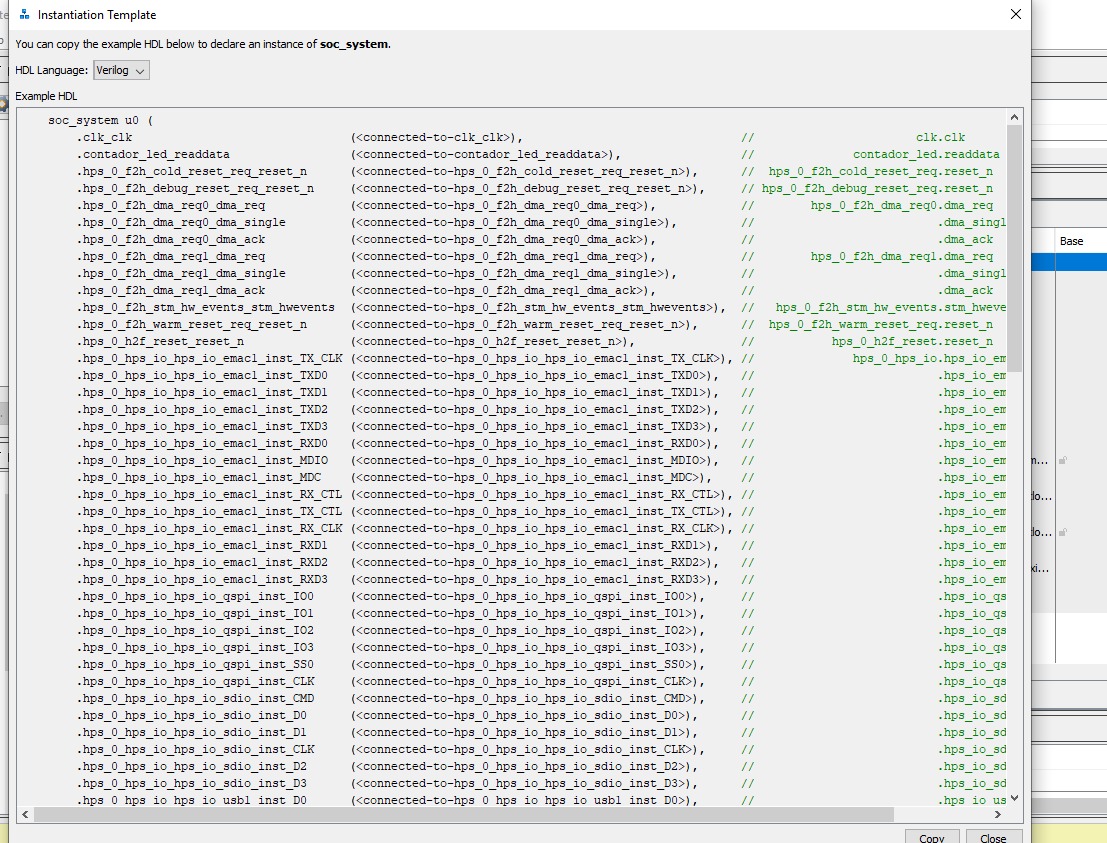


Ilustración - Instantiation Template

1. En el archivo *DE10\_Standard\_FB.v* reemplace la componente *soc\_system u0* por el texto generado en el paso anterior y complete las entradas y salidas que recibe dicho componente. Debe quedar lo más semejante al código adjunto en el siguiente enlace:

<https://drive.google.com/file/d/1AbAcdRImQLI_LZUMD9LCBaUVxVFCaM0z/view?usp=sharing>

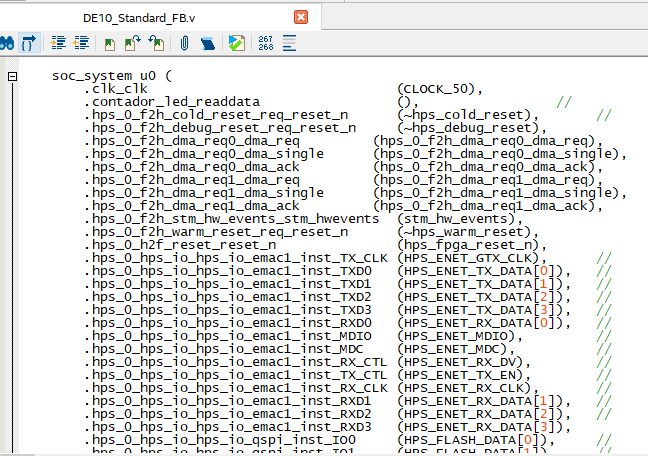


Ilustración - Entradas y salidas de la componente soc\_system u0

1. Compilación del proyecto. Repita el paso 6 y 7 para el archivo *DE10\_Standard\_FB.v.*
2. Cree una carpeta dentro del directorio */Quartus* y llámela *WORKSPACE* como indica la imagen.

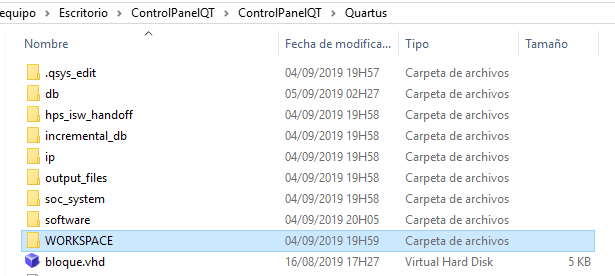


Ilustración - Espacio de trabajo WORKSPACE

1. Para programar el microprocesador. Abrir *Eclipse* desde el menú de *Quartus*, *Tools -> Nios II Software Build Tools for Eclipse*.

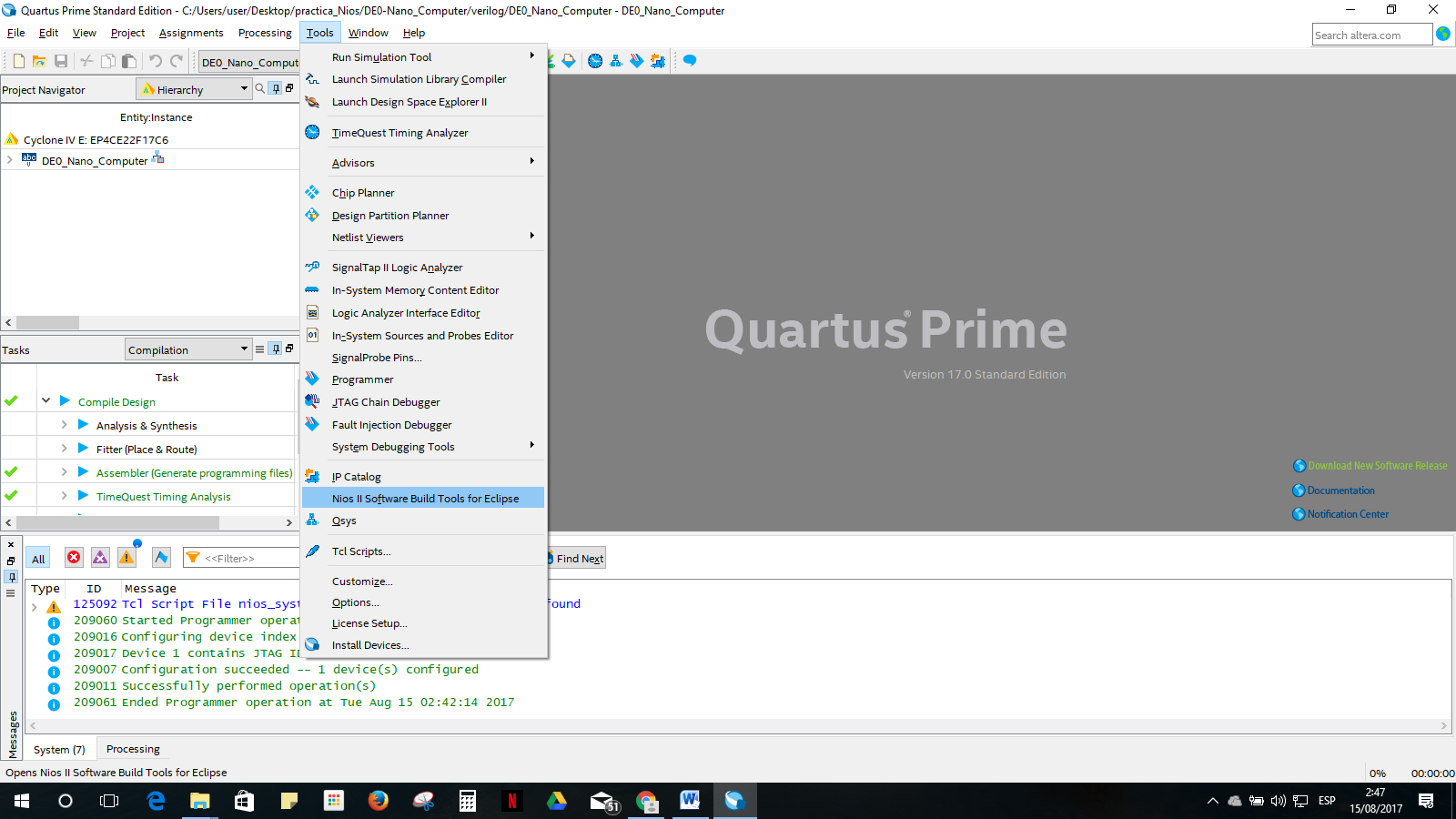


Ilustración - Seleccionar herramienta Eclipse

1. Lo primero que pedirá Eclipse será ubicar en donde va a estar alojado el proyecto que vayamos a crear, seleccione la carpeta *WORKSPACE* que se encuentra dentro de la carpeta del proyecto, como se encuentra en la Ilustración 10. Haga clic en OK.

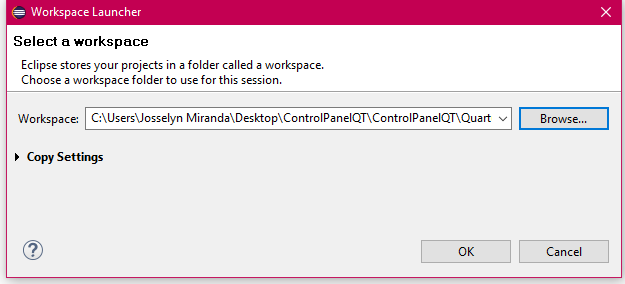


Ilustración - Ubicación del proyecto en Eclipse

1. Se abrirá el entorno de programación de *Eclipse*. Se procederá a crear un nuevo proyecto, para ello seleccione *File→New→Nios II Application and BSP from Template* como se visualiza en la Ilustración 11.

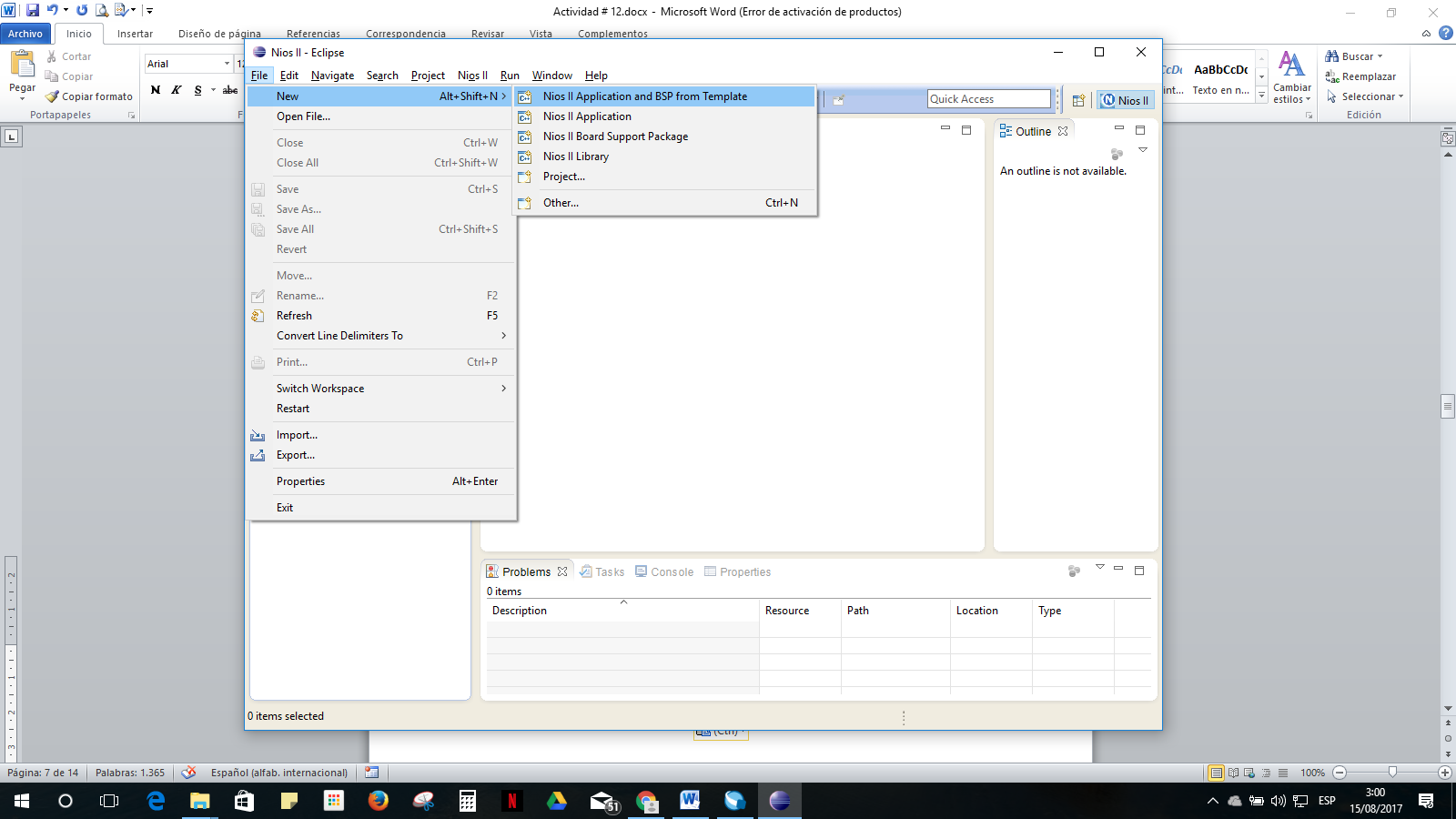


Ilustración - Seleccione una nueva aplicación para NIOS II

1. En la opción *SOPC Information File name*, proceda a buscar el archivo *soc\_system.sopcinfo* que tendrá toda la información de la computadora embebida, como se visualiza en Ilustración 12.

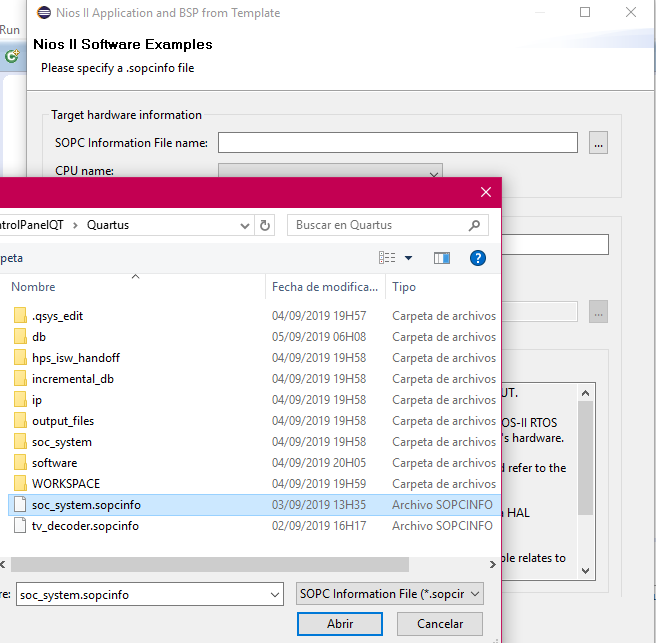


Ilustración - Seleccionar soc\_system.socinfo

1. En la opción *CPU* name se puede utilizar el núcleo disponible en la computadora embebida. Se utilizará el único núcleo *nios2\_gen*.
2. En *Project name* se elegirá el nombre del proyecto, utilice el nombre FINAL de preferencia.
3. En *Project Template*, seleccione la opción *Blank Project* para crear nuestro proyecto vacío. Haga clic en *Finish*.



Ilustración - Información para crear proyecto

En la sub-ventana de Project Explorer podrán visualizar dos carpetas: FINAL y FINAL\_bsp. En la primera se podrán crear todos los archivos de programación en lenguaje C/C++ para nuestro microprocesador, y la segunda carpeta contiene todos los componentes del hardware, como se visualiza en la Ilustración 13.

1. Se procederá a crear un archivo en lenguaje C, para ello haga clic derecho en la carpeta FINAL y seleccione *New → File* como se muestra en la Ilustración 14.

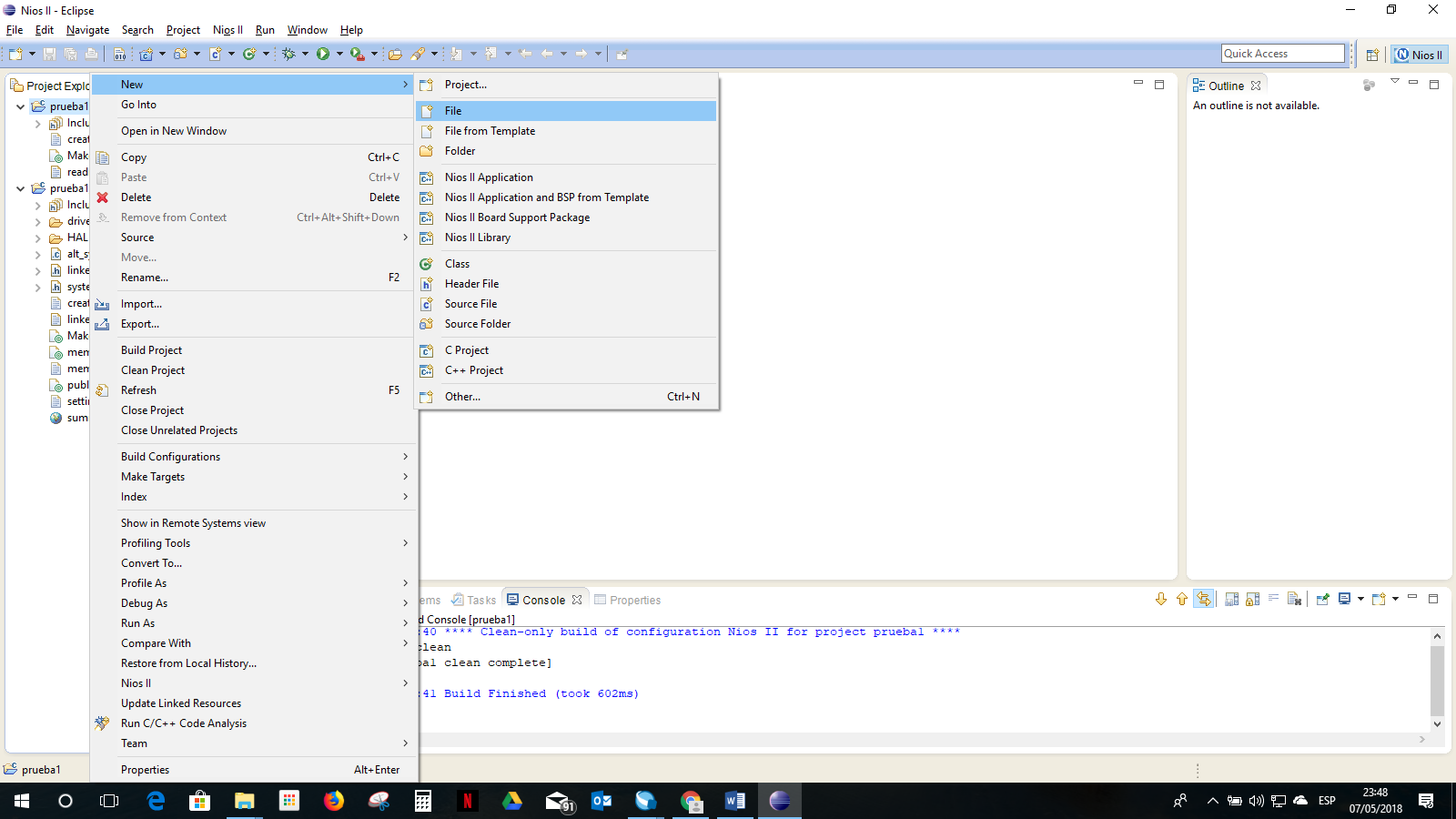


Ilustración - Crear archivo main.c en Eclipse

1. En la ventana *New File*, coloque el nombre *main.c* y haga clic en *Finish* como se muestra en la Ilustración 15.

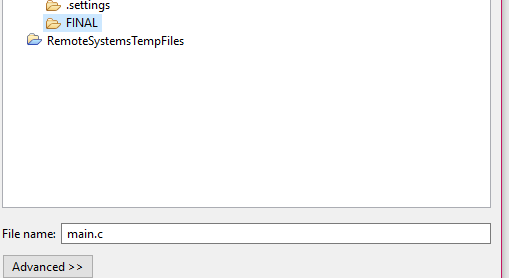


Ilustración - Ventana de la creación de un archivo en lenguaje C

1. Doble clic en el archivo main.c generado y copie el siguiente código:

#include <stdio.h>

#include <stdlib.h>

#include "io.h"

#include "altera\_up\_avalon\_video\_character\_buffer\_with\_dma.h"

int main (void)

{

alt\_up\_char\_buffer\_dev \*dev1;

alt\_up\_char\_buffer\_init(dev1);

dev1 = alt\_up\_char\_buffer\_open\_dev("/dev/video\_character\_buffer\_with\_dma\_0");

alt\_up\_char\_buffer\_clear(dev1);

volatile int \* SW\_switch\_ptr = (int \*) 0x00004000;

volatile int \* contador = (int \*) 0x00001010; // contador

int \* memoria = (int \*) 0x00080000; //memoria

int SW\_value;

int consulta\_val;

int tabla\_val;

//char input[100];

char time[100];

char read[50];

int \_consulta = 1;

int \_valor = 45;

IOWR(memoria,0x0008,\_consulta);

IOWR(memoria,0x0009,\_valor);

while (1){

//printf("El resultado es: %d\n",\*(contador+2));

//int value = (int) jota;

consulta\_val = IORD(memoria,0x0008); //consultamos el numero de consulta;

printf("Consulta: %d\n",consulta\_val);

SW\_value = \*(SW\_switch\_ptr);

int band = 0;

\*(contador+1)=0;

\*(contador)=4;

\*(contador)=2;

alt\_up\_char\_buffer\_string(dev1, "Utilice una combinación de switch (SW0,SW1,SW2) para realizar consulta. ", 0,5);

alt\_up\_char\_buffer\_string(dev1, "Utilice SW9 para mostrar la consulta realizada. ", 0,6);

if(SW\_value&512){

alt\_up\_char\_buffer\_clear(dev1);

tabla\_val = IORD(memoria,0x0008);

sprintf(read, "%d", tabla\_val);

printf("Valor: %d\n",tabla\_val);

if(consulta\_val==1){

alt\_up\_char\_buffer\_string(dev1,"Mostrando consulta:", 0,1);

alt\_up\_char\_buffer\_string(dev1,"TABLA: sensor\_temp1:", 0,3);

alt\_up\_char\_buffer\_string(dev1,"Temperatura (ultimo registro):", 0,4);

alt\_up\_char\_buffer\_string(dev1, read, 50,4);

alt\_up\_char\_buffer\_string(dev1, "En el segundo: ", 0,47);

sprintf(time, "%d", \*(contador+2));

alt\_up\_char\_buffer\_string(dev1, time, 17,47);

band = 1;

}

if(consulta\_val==2){

alt\_up\_char\_buffer\_string(dev1, "Mostrando consulta:", 0,1);

alt\_up\_char\_buffer\_string(dev1, "TABLA: sensor\_temp2:", 0,3);

alt\_up\_char\_buffer\_string(dev1, "Temperatura (ultimo registro):", 0,4);

alt\_up\_char\_buffer\_string(dev1, read, 50,4);

alt\_up\_char\_buffer\_string(dev1, "En el segundo: ", 0,47);

sprintf(time, "%d", \*(contador+2));

alt\_up\_char\_buffer\_string(dev1, time, 17,47);

band = 1;

}

if(consulta\_val==3){

alt\_up\_char\_buffer\_string(dev1, "Mostrando consulta:", 0,1);

alt\_up\_char\_buffer\_string(dev1, "TABLA: sensor\_temp3:", 0,3);

alt\_up\_char\_buffer\_string(dev1, "Temperatura (ultimo registro):", 0,4);

alt\_up\_char\_buffer\_string(dev1, read, 50,4);

alt\_up\_char\_buffer\_string(dev1, "En el segundo: ", 0,47);

sprintf(time, "%d", \*(contador+2));

alt\_up\_char\_buffer\_string(dev1, time, 17,47);

band = 1;

}

}

if(band==1)

{ usleep(4000000);

SW\_value = \*(SW\_switch\_ptr);

while(SW\_value&512){

alt\_up\_char\_buffer\_string(dev1, "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*", 0,49);

alt\_up\_char\_buffer\_string(dev1, "Regrese el switch a la posicion original para seguir consultado. ", 0,50);

SW\_value = \*(SW\_switch\_ptr);

printf("%d\n",SW\_value);

usleep(90000);

}

alt\_up\_char\_buffer\_clear(dev1);

band =0;

}

}

}

Este código permite leer en memoria y presentarle al usuario por VGA la respuesta de la consulta a la base de datos configurada en el HPS.

1. Luego proceda a hacer clic derecho en la carpeta *FINAL* y seleccione la opción *Build Project* como se muestra en la Ilustración 16.

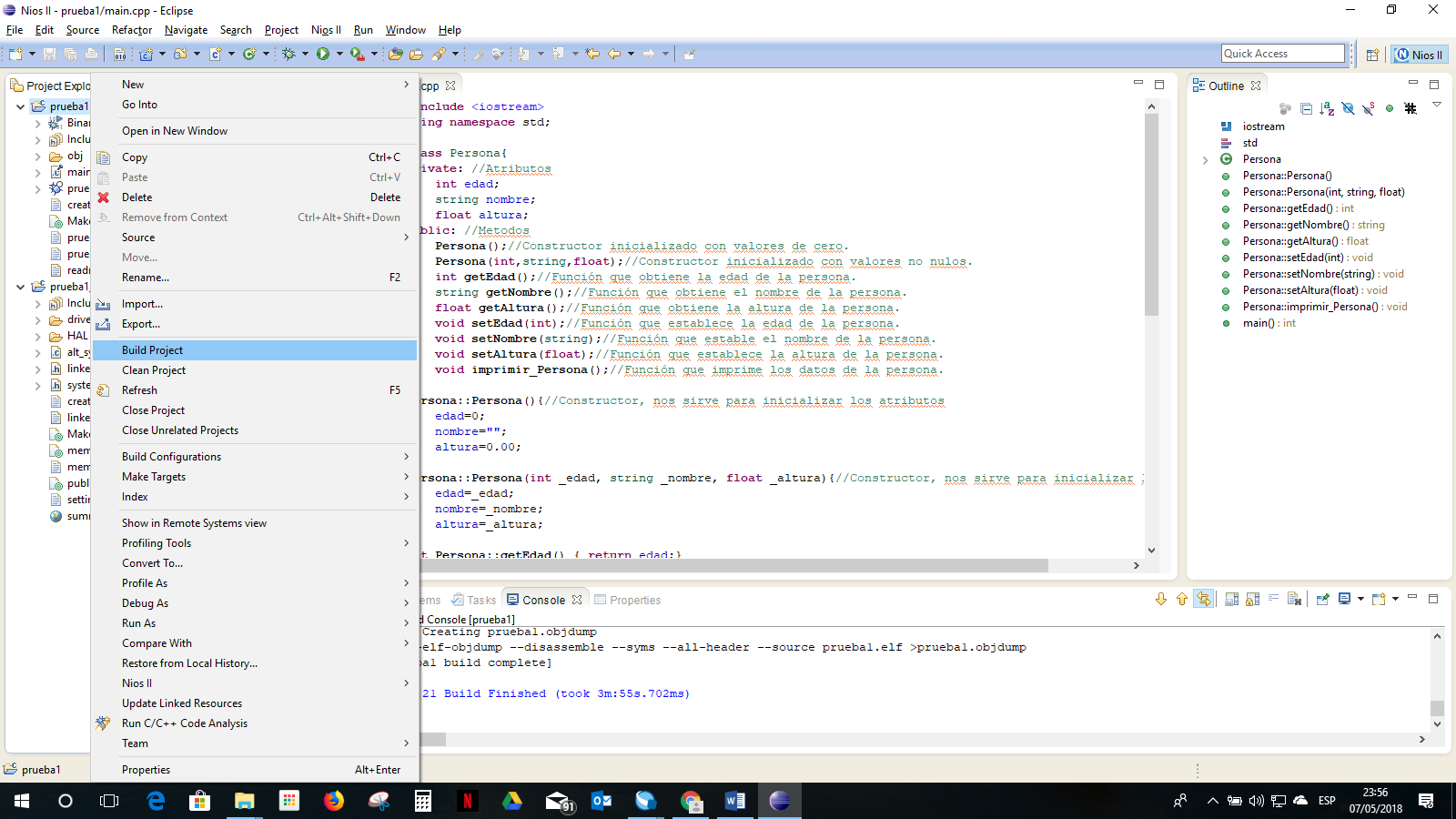


Ilustración - Compilación del proyecto

1. Generar el archivo que utilizará el NIOS II para correr el script cuando inicie el sistema.



Ilustración - Generar archivo .

1. Agregar el archivo generado en el paso anterior al *NIOS II* por medio del software *Qsys*. Doble clic en el componente *on\_chip\_memory*, en la opción *tal -> tal* seleccionar el archivo alojado en la ruta por defecto */tal/tal.*



Ilustración - Añadir archivo al NIOS II

1. Repita el paso 16.
2. Repita el paso 7. Una vez que la compilación llegue al 100% se genera un archivo *soc\_system.dtbo* en el directorio */Quartus* y otro archivo *soc\_system.rbf* en el directorio */Quatus/output\_files*. Conecte la MicroSD a la computadora donde esta trabajando y copie estos dos archivos en el directorio */lib/firmware* de la *MicroSD*.



Ilustración - Actualización de archivos de configuración FPGA en Linux

Estos archivos son necesarios para que el Sistema operativo Linux configure la parte de la *FPGA* cuando el sistema arranque, el archivo *dtbo* corresponde a la superposición FPGA device tree y *rbf* corresponde al flujo de bits de configuración de la FPGA.

1. Conecte la *MICROSD* a la tarjeta de desarrollo DE10 Standard.
2. Cambie el Swtich MSEL[4:0] a 01010.
3. Proceda a conectar la tarjeta de desarrollo DE10 Standard a la fuente de alimentación.

## Configuración del Servidor de base de datos en Linux

1. Después de arrancar el sistema operativo, utilizar el usuario por defecto para iniciar sesión en el mismo. *User: root, sin contraseña*.

# ENLACE EN DRIVE

Carpeta compartida con acceso a lectura

<https://drive.google.com/drive/folders/1JTLbT8e3Wnj9QqhATpKR9N5PE9A0YCby?usp=sharing>

# ENLACE EN GITHUB

<https://github.com/jocammir/Sistema_gestion_base_de_datos_FPGA_HPS_DE10Standard.git>

Dado que es un repositorio privado debe solicitar permisos al siguiente correo electrónico: [jocammir@espol.edu.ec](mailto:jocammir@espol.edu.ec)

1. <https://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&No=1081&PartNo=4> [↑](#footnote-ref-1)