Universidad de La Laguna

Proyecto Diseño de Procesadores

Difuminado de imagen

Autores: Melissa Díaz Arteaga

Rafael González de Chaves González

Índice:

- Introducción……2
- Creando un hardware básico……3
    - 1. Especificar los ajustes de la FPGA y el reloj……4
    - 2. Agregar la CPU Nios II, la memoria y otros componentes……4
        - Memoria on-chip……4
        - JTAG UART……6
        - Timer ……7
        - System ID Peripheral……7
        - PIO Signals……8
    - 3. Especificar las direcciones……9
    - 4. Generamos el sistema Qsys……9
    - 5. Asignación de pines a la FPGA……10
    - 6. Compilar el proyecto……11
- Módulos adicionales para la implementación de la matriz……13
    - Elementos añadidos……13
        - SRAM……13
        - SDRAM……13
        - Timestamp……14
    - Pruebas……16
        - SRAM……16
        - SDRAM……16
        - Timestamp……18
- Implementación del programa……18
- Conclusiones……22
- Bibliografía……22

Introducción:

En este trabajo vamos a implementar un programa en lenguaje c que, utilizando la FPGA, difumine una imagen dada utilizando una matriz 3x3 para calcular los valores de los pixeles de la imagen resultante.

- En primer lugar, crearemos nuestro hardware básico utilizando el programa Quartus II.

- Posteriormente añadiremos los módulos necesarios para hacer funcionar nuestro programa.

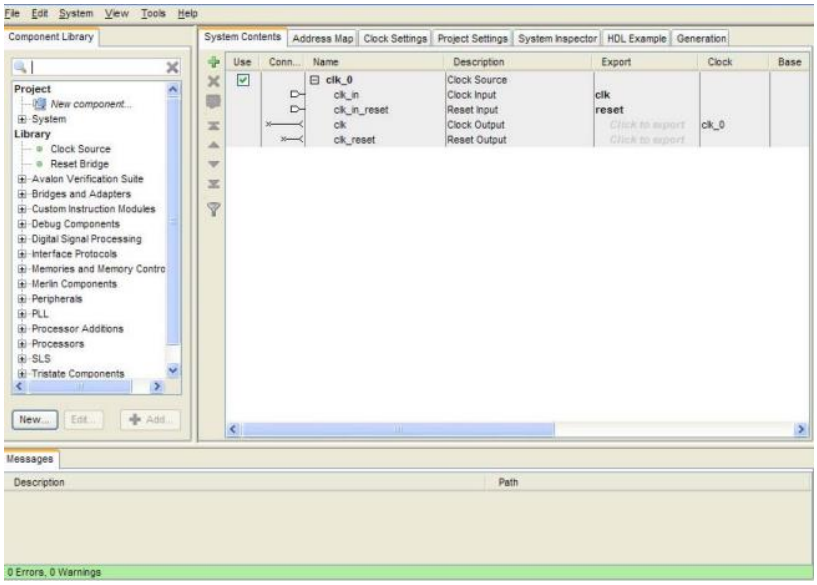
- Finalmente implementaremos el programa en lenguaje c para el difuminado de la matriz.

Analizaremos las decisiones de diseño, así como, en cada uno de los apartados anteriormente especificados, el paso a paso de la implementación.

Realizaremos, además, pruebas utilizando un módulo timestamp el cual nos permitirá calcular los ciclos de reloj que tarda nuestro programa en ejecutarse.

Creando un hardware básico:

En este apartado desarrollaremos el hardware básico necesario para comunicarnos con la FPGA, para ello, una vez tenemos el **Quartus II** abierto crearemos un nuevo ***Qsys System*** desplegando la pestaña ***System Contents***. En ella definiremos las características de hardware del sistema Nios II, la CPU que vamos a utilizar y los componentes vamos a incluir en el sistema.



Comenzaremos realizando los siguientes pasos:

1. Especificamos los ajustes de la FPGA y el reloj deseados.
2. Agregamos la CPU Nios II, la memoria y otros componentes.
3. Especificamos las direcciones.
4. Generamos el sistema Qsys.
5. Asignación de pines de la FPGA.
6. Compilar el proyecto.

1. Especificar los ajustes de la FPGA y el reloj:

En este apartado especificaremos los ajustes de la FPGA y reloj, para ello:

-Primero, en la pestaña de ***Project Settings*** seleccionamos la familia del dispositivo de nuestra FPGA, en este caso **Cyclone II**.

- Posteriormente en la pestaña de ***Clock Settings*** especificamos un reloj. Por defecto tendrá el nombre **clk_0** y especificaremos una frecuencia de **50MHz**.

2. Agregar la CPU Nios II, la memoria y otros componentes.

Memoria on-chip:

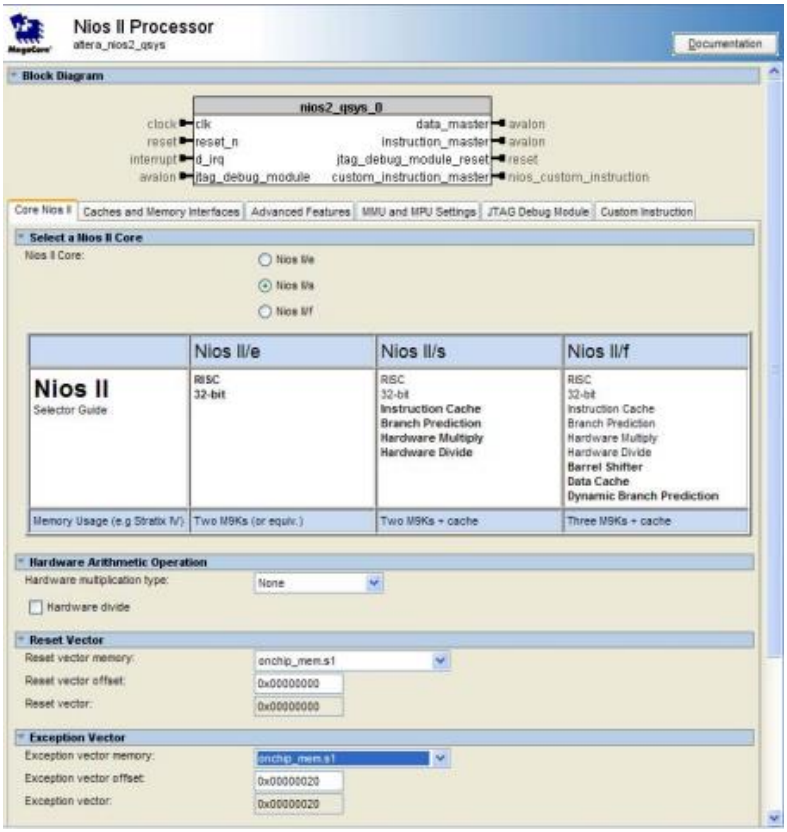
Ahora añadiremos la memoria O-Chip de la FPGA ya que, los sistemas requieren al menos una memoria para datos e instrucciones. Para este proyecto utilizaremos una memoria de 20 KB y la agregamos siguiendo los siguientes pasos:

- En la pestaña ***Component Library*** seleccionaremos el apartado ***Memories and Memory Controllers -> On-Chip -> On-Chip –Memory (RAM or ROM)***. Especificando un **Block type** automático y un tamaño de memoria total de 20480.Luego, haciendo click derecho sobre el módulo en la ventana ***System Contents*** cambiaremos el nombre por **onchip_mem**.

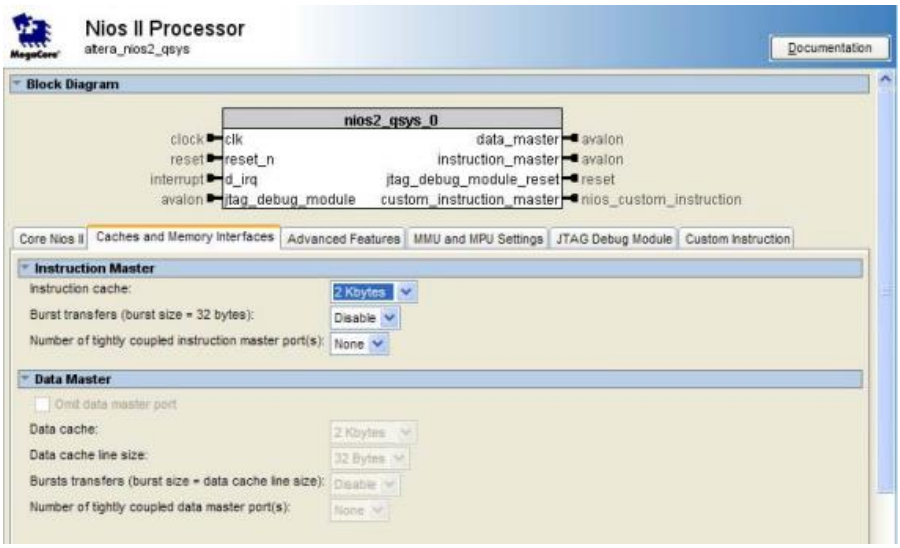
- Añadiremos la CPU para Nios II y la configuramos para 20 KB de memoria on-chi. Para ello, en la pestaña ***Component Library***, seleccionaremos **Nios II Processor**. Posteriormente nos saldrá una ventana en la que seleccionaremos la pestaña ***Core Nios II***. En esta podremos especificar el tipo de CPU que, en este caso es **Nios II/s**, el tipo de multiplicación hardware **ninguno**, desactivaremos la división hardware y tras aceptar volveremos a la pestaña ***Qsys System Contents***.

Ahora podremos ver la instancia de la CPU Nios II en la tabla de componentes y cambiaremos entonces el nombre por cpu.

- Configuraremos las conexiones conectando al reloj clk al **clk_0** y al **clk 1** y puerto **clk_reset** del **clk_0** al puerto **reset1** del On-chip y al puerto **reset_n** del procesador Nios II. Ahora volveremos a abrir el ***Nios II Processor*** y en la opción ***Reset Vector Memory*** seleccionaremos **onchip_mem.s1** y el tipo **0x0** en el recuadro de ***Reset Vector offset***. En el ***Exception Vector*** seleccionaremos también **onchip.mem.s1** y escribiremos 0x20 en el cuadro de ***Exception vector offset***.



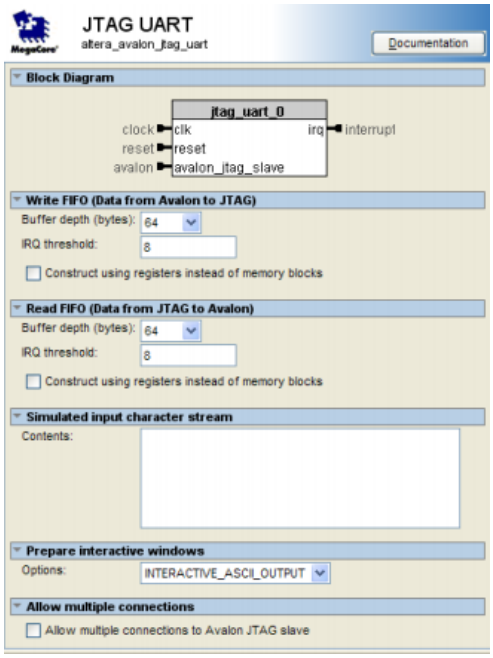
Luego, en la pestaña ***Caches and Memory Interfaces*** seleccionaremos en la ***Instruction Caché*** la opción de **2Kbytes**. Desactivaremos, además, el ***Burst transfers*** y el ***Number of tightly coupled instruction master port(s)*** seleccionando ninguno. Al finalizar volveremos a la pestaña ***Qsys System Contents***.



JTAG UART:

El siguiente módulo que añadiremos es el **JTAG UART** el cual proporciona una forma conveniente de comunicar datos con el procesador Nios II a través del cable de descarga USB-Blaster.

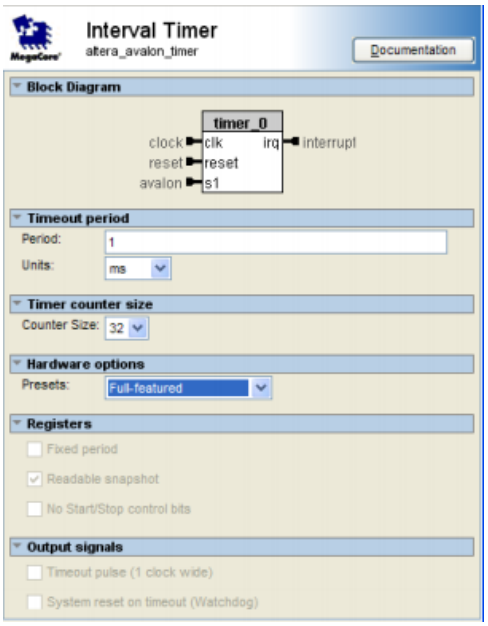
Para ello seleccionamos la pestaña ***Component Library-> Protocols -> Serial -> JTAG UART***.



Ahora, cambiaremos el nombre en la ventana e ***Qsys System Contents*** por **jtag_uart** y conectaremos el **clk_0** al **clk** del JTAG UART así como el **clk_reset** del clk_0 al **reset** del JTAG UART. Y finalmente el puerto **data_master** de la cpu al **avalan_jtag_slave** del JTAG UART.

Timer:

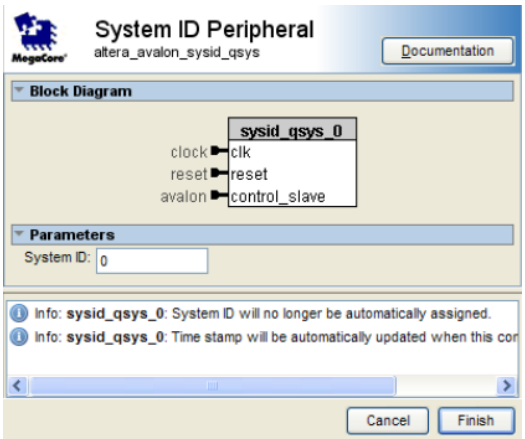
Es necesario añadir un **timer** el cual permite un cálculo preciso del tiempo. Para ello, en la pestaña ***Component Library-> Peripherals->Microcontroller Peripherals-> Interval Timer***. En la pestaña que nos aparece cambiaremos en el apartado ***Presets*** la opción **Full-featured**.



Cambiaremos el nombre por **sys_clk_timer** y conectaremos el **clk** del clk_0 al **clk** del timer además del **clk_reset** del clk_0 al **reset** del timer. Finalmente conectaremos **data_master** al puerto **s1** del timer.

System ID Peripheral

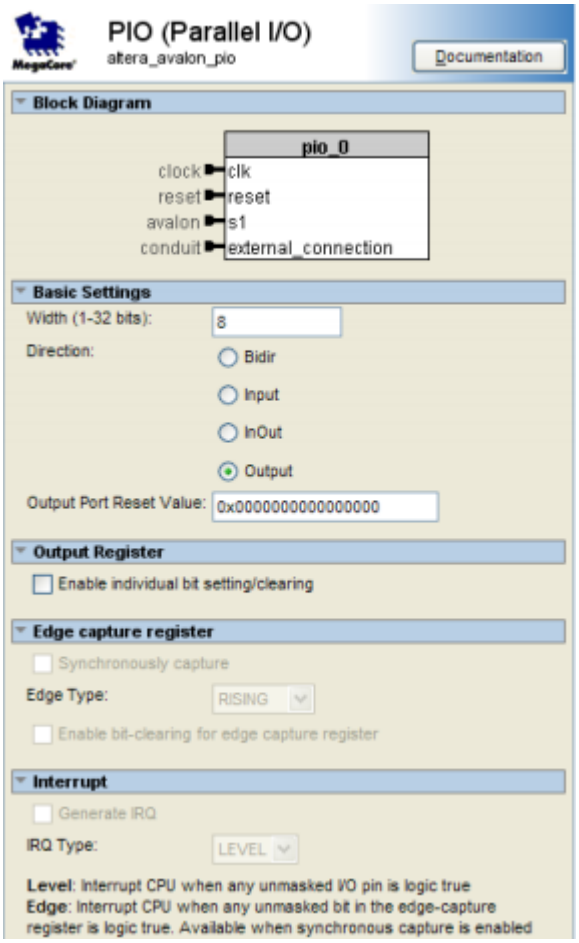
Implementaremos un **system ID peripheral** el cual salvaguarda contra descargas accidentales de software compilado para un sistema Nios II. Lo seleccionaremos ***Component Library-> Peripherals-> Debug and Performance-> System ID Peripheral.***



Cambiaremos el nombre por **sysid** y conectaremos el **clk** y el **reset** del clk_0 al **clk** y el **reset** del ***system ID peripheral*** respectivamente. Finalmente conectaremos **data_master** de la cpu al **control_slave** del system ID peripheral.

PIO Signals:

Implementaremos además un PIO Signals el cual provee un método fácil de recibir señales de entrada y salida. Para ello seleccionaremos ***Component Library -> Peripherals-> Microcontroller Peripherals -> PIO (Parallel I/O).***

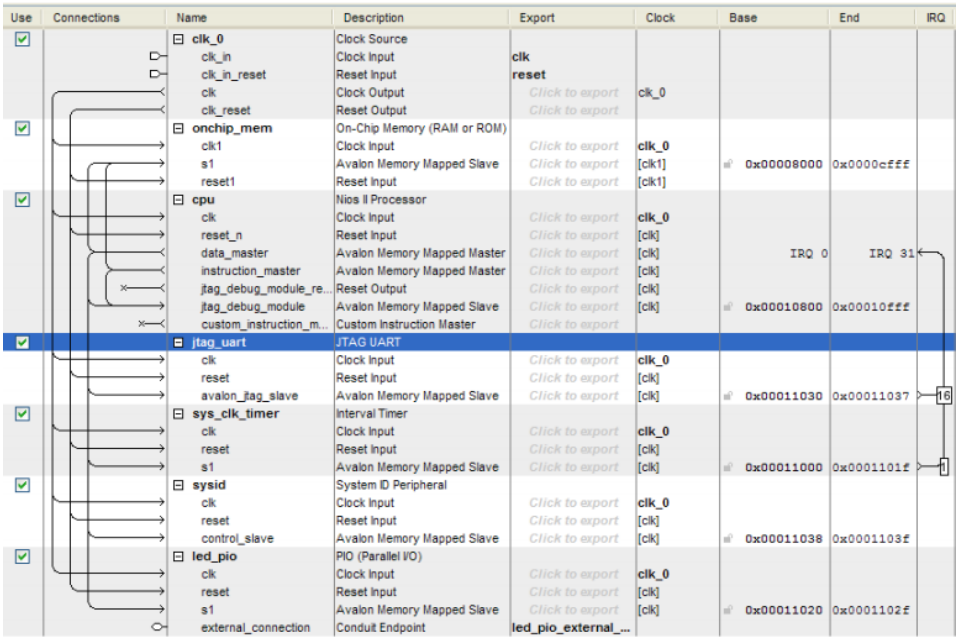


Cambiaremos el nombre a **led_pio** y conectaremos el **clk** y el **reset** del clk_0 al **clk** y el **reset** del PIO. Finalmente conectaremos **data_master** de la cpu al puerto **s1** del PIO. En la fila ***external_connection*** y la columna ***Export*** seleccionaremos **export** para exportar los puertos de PIO.

3.Especificar las direcciones.

Para especificar las direcciones utilizaremos el botón ***Assing Base Addresses*** que se encuentra en la pestaña de ***System*** en el menú de la parte superior. Esto reorganizará las direcciones para que no se solapen.

Debería quedarnos el siguiente esquema:

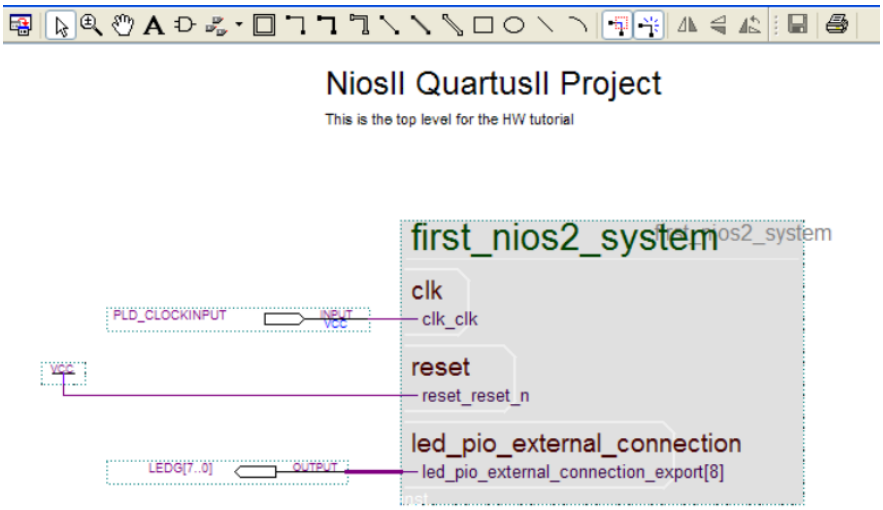


4. Generamos el sistema Qsys.

Para generar el Qsys System utilizaremos la pestaña ***Generation***, seleccionaremos **None** en los recuadros ***Creatte simulation model*** y ***Create teshbench Qsys system***. Luego, seleccionaremos **Generate** guardaremos los cambios con el nombre de **first_nios2_system**.

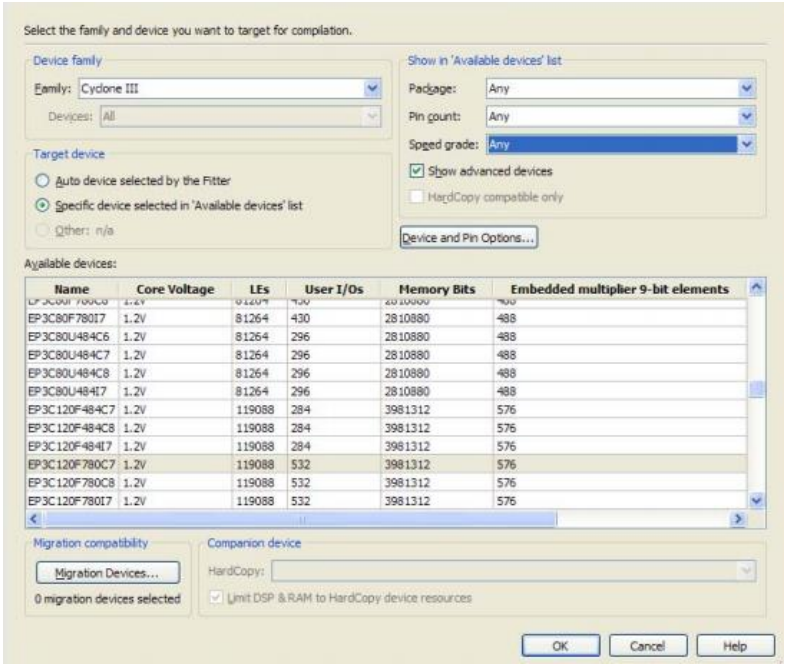
Ya tenemos nuestro hardware generado.

El siguiente paso sería integrar el ***Qsys System*** en el ***Quartus II Project***. Para instanciar el módulo del sistema en ***Libraries*** expandiremos ***Project*** seleccionaremos **first_nios2_system** y veremos como aparece el dibujo de nuestro hardware en la ventana.

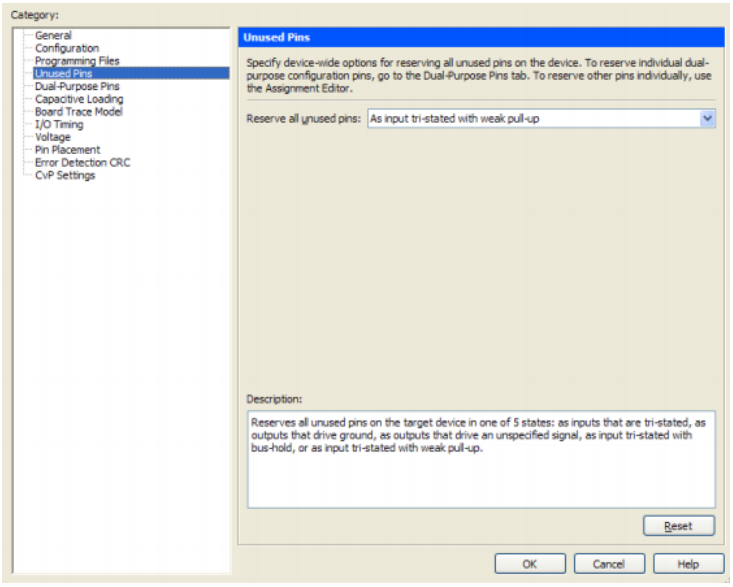


5. Asignación de pines a la FPGA.

Para asignar los pines de nuestro proyecto a los de la FPGA utilizaremos la pestaña ***Device*** del menú y seleccionaremos la configuración que aparece en la imagen, seleccionando además el modelo de la FPGA que en nuestro caso es **EP2C20F484C7.**

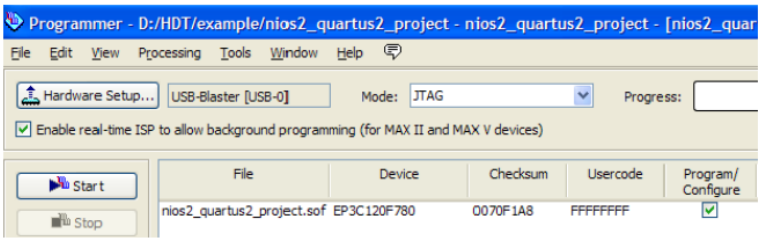


Posteriormente seleccionaremos la pestaña ***Pin Planner*** del menú y en la ventana que aparece localizaremos **PLD_CLOCKINPUT** y para ***Location*** seleccionaremos **I/O Standard**. Luego volveremos a abrir la ventana ***Device*** y en la pestaña ***Device and Pin Options*** iremos a la página de ***Unused Pins***. Aquí, seleccionaremos **input tri-stated with weak pull-up** y aceptaremos.

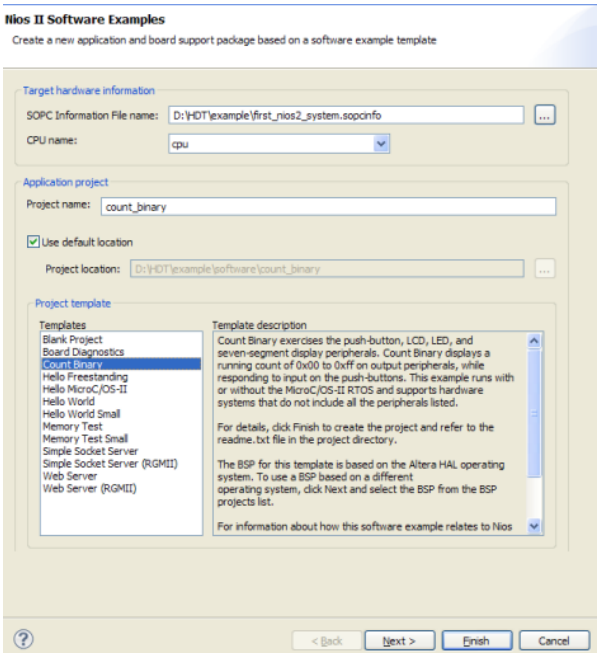


6. Compilar el proyecto.

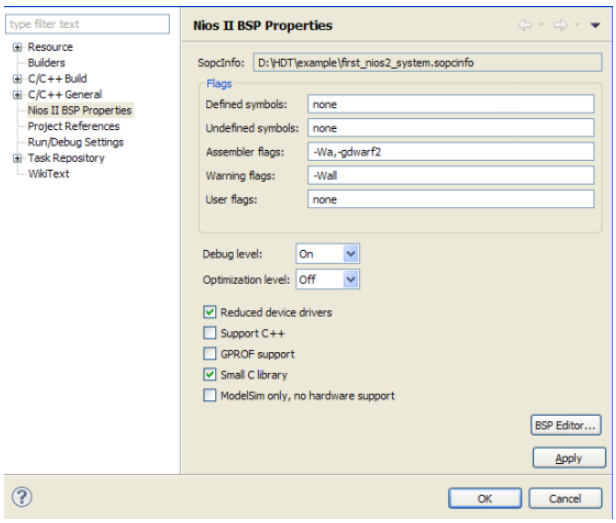
Ahora compilaremos el proyecto. Para ello, primero abriremos nuestro proyecto seleccionando ***Open Files*** y dándole al botón ***play***. Una vez compilado podemos conectar nuestra FPGA y utilizando la pestaña ***Tools*** del menú superior y seleccionado ***Programmer*** el hardware cargará el proyecto y arranque con el botón **start**.



Posteriormente crearemos un **count binary** para probar nuestro hardware. Para hacerlo, abriremos ***eclipse*** y crearemos un nuevo **Workspace Launcher**.

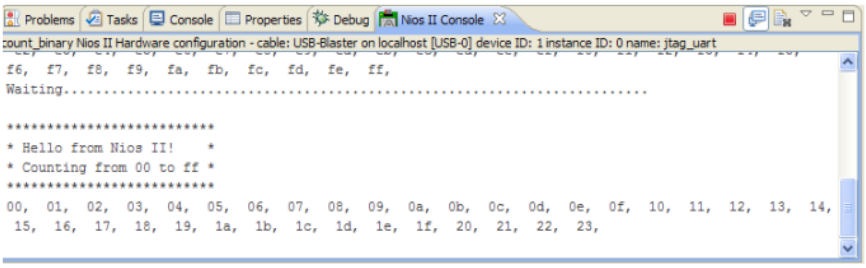


Luego en el ***Project Explorer*** crearemos un ***count_binary_bsp*** y en propiedades especificaremos la siguiente configuración:



Finalmente, al aceptar volveremos al ***Project Explorer*** y haciendo click derecho sobre el proyecto seleccionaremos ***Build Project*** y posteriormente Run as Nios II Hardware.

Deberá salirnos una ventana donde estará ejecutándose nuestro programa de prueba en c.



Módulos adicionales para la implementación de la matriz:

Partiendo de lo hecho anteriormente hemos realizado algunos cambios para poder implementar un programa en c que difumine una imagen dada utilizando una matriz de 3x3 con distintos valores la cual multiplica los pixeles circundantes. Estos datos se utilizan para calcular el nuevo pixel central suavizado.

Elementos añadidos:

Principalmente implementamos una **SRAM** para albergar datos ya que la memoria on-chip era demasiado limitada. Una vez conocimos la resolución máxima que tendrían las fotos a difuminar decidimos introducir una **SDRAM** la cual nos proporciona un espacio de 8Mb la cual nos permite albergar tanto el programa como la imagen origen, destino y la matriz 3x3.

SRAM:

Para implementar la SRAM añadimos un controlador de SRAM en ***University Program -> Memory -> SRAM/SSRAM Controller***. Y en la pestaña configuración seleccionamos en ***DE-Series Board*** la opción **DE1**.

Posteriormente conectamos el **clk** y el **reset** de la SRAM al **clk** y **reset_n** de clk_0 respectivamente. Luego conectamos **data_master** e **instruction_master** de la cpu al **avalon_sram_slave** de la SRAM. Luego exportamos el **external_interface**. Una vez exportado asignamos los pines al modelo que tenemos en el Nios II.

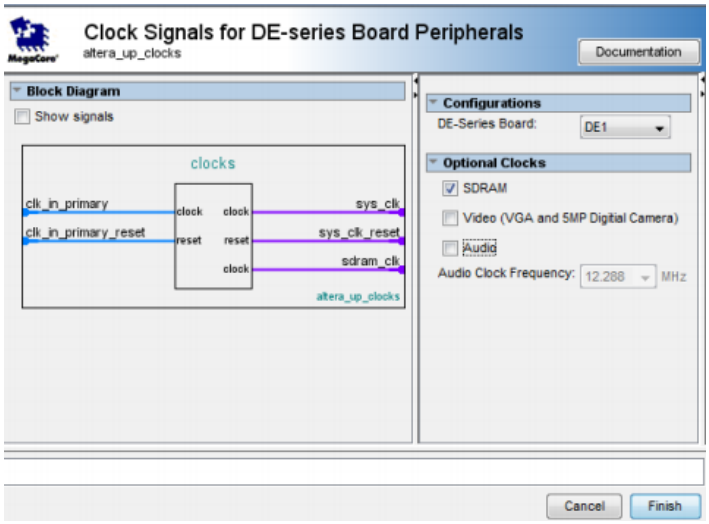
SDRAM:

Para realizar la implementación de la SDRAM seleccionaremos ***Qsys Memories and Memory Controllers -> External Memory Interfaces -> SDRAM Interfaces -> SDRAM Controller***

Luego en la ventana de configuración seleccionamos en el apartado ***Presets*** la opción **single NEC D4564163-A80 chip (64Mb x 16).** Cambiamos el nombre del módulo por sdram. Conectamos el **clk** y el **reset** de la sdram al **clk** y el **reset** del clk_0 y el **data_master** e **instruction_master** al puerto s1 de la SDRAM.

Finalmente exportamos el wire y los conectamos a los pines correspondientes en el Nios II.

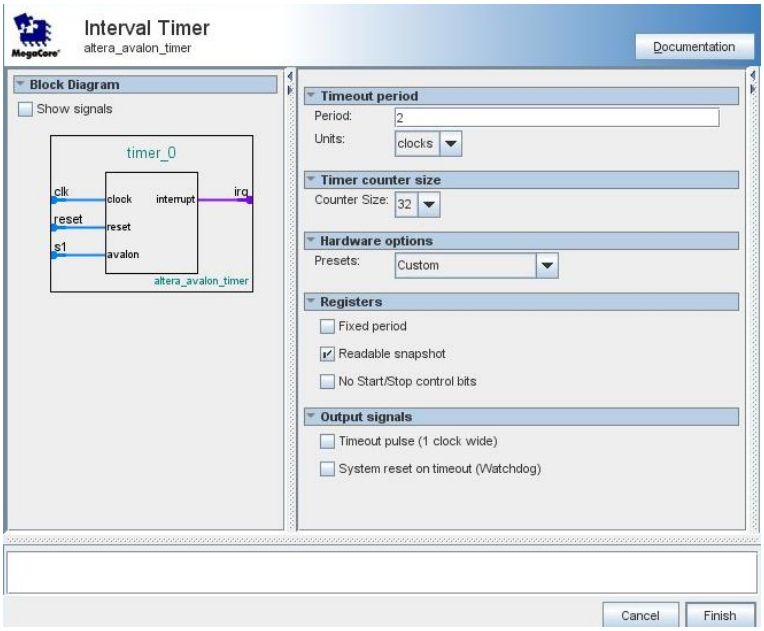
Además, la SDRAM necesita además un clk que denominamos **up_clocks_0**. Para implementarlo utilizamos la ***pestaña University Program -> Clocks Signals for DE-Series Board Peripherals del Component Library***. En las opciones de configuración seleccionamos **DE1** en ***DE-Series Board*** y en el apartado ***Optional Clocks*** seleccionaremos **SDRAM**.



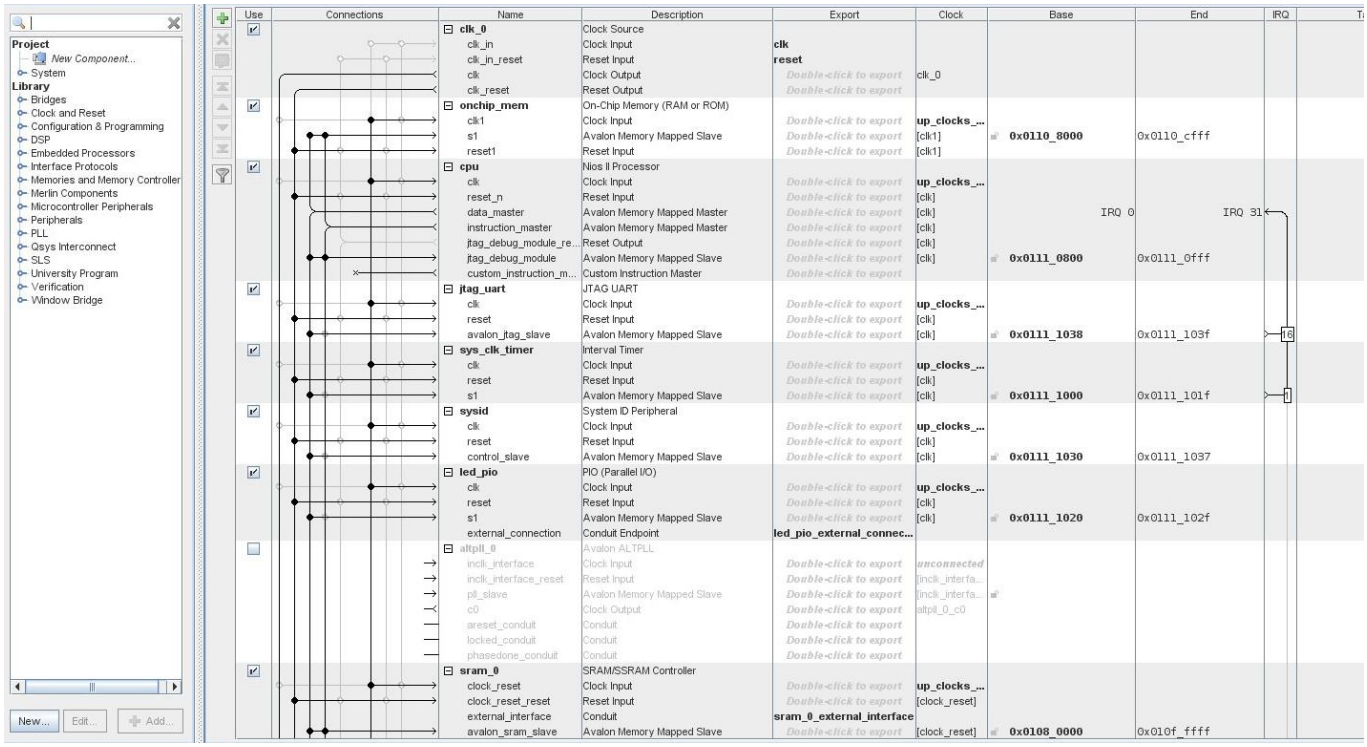
Finalmente, conectamos el **clk** y el **reset** del clk_0 a las entradas de **up_clocks_0** y la salida del **sys_clk** de up_clocks_0 a la entrada de **clk** del resto de componentes.

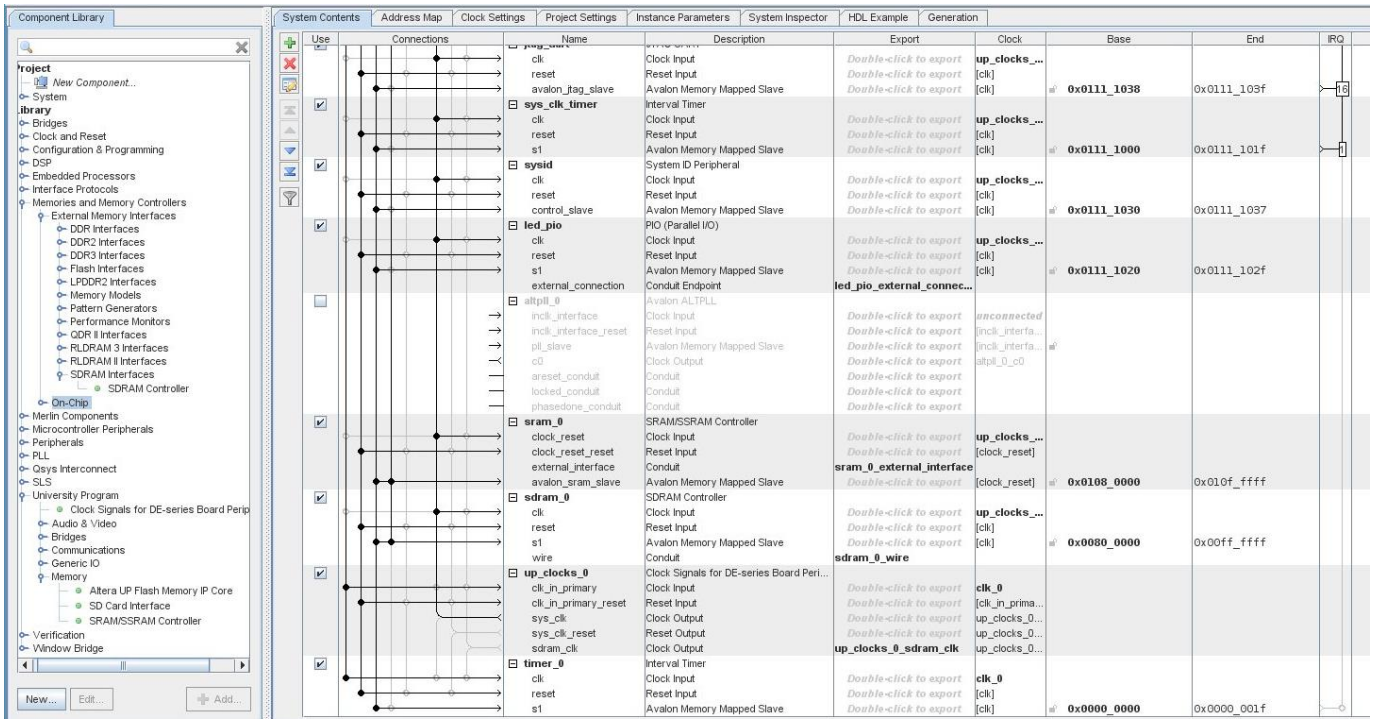
Timestamp:

Por último, implementamos un timestamp para contar ciclos de reloj y poder realizar pruebas de rendimiento. Para ello en la pestaña ***Component Library-> Peripherals->Microcontroller Peripherals-> Interval Timer***. En la pestaña que nos aparece especificaremos las opciones correspondientes:

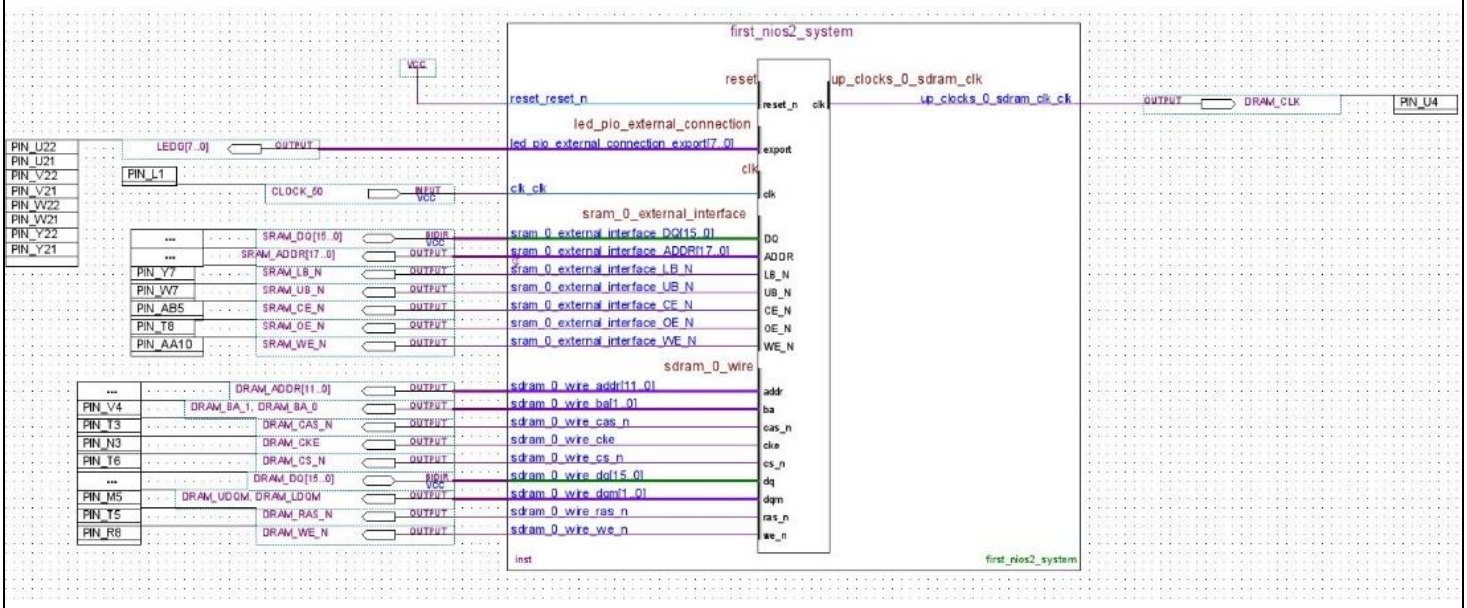


Al introducir los nuevos elementos la ventana de **Qsys** nos quedará de la siguiente forma:





La asignación de pines debería quedra de la siguiente forma:



Pruebas de funcionamiento:

Para verificar el funcionamiento de los módulos añadimos realizamos una serie de pruebas.

SRAM:

Añadimos al count_ binary el siguiente código:

```
55    char* num;
56    int i = 0;
57    for(i=0;i<(512*1024);i++){
58       num = 0x01080000 + i;
59       *num = 5;
60       if((int)*num != 5) printf("Num %x %d", num, *num);
61    }
62
```

En el recorremos y escribimos en las posiciones de memoria que corresponden a la SRAM.

SDRAM:

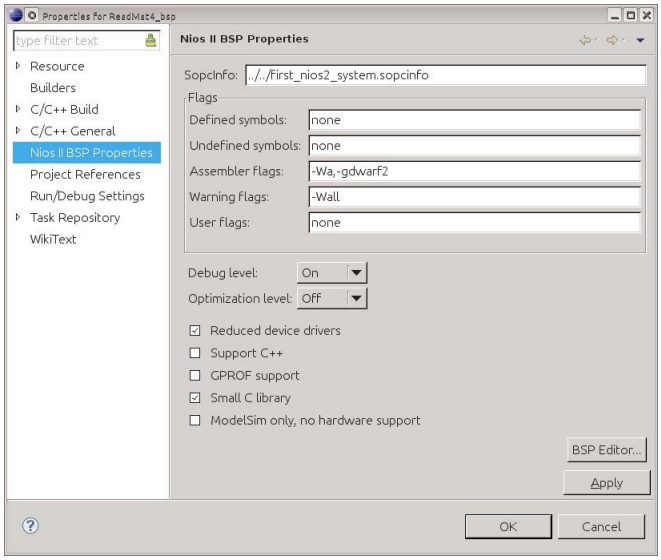
Añadimos al count_ binary el siguiente código:

```
55    char* num;
56    int i = 0;
57    for(i=0;i<(8*1024*1024);i++){
58       num = 0x00080000 + i;
59       *num = 5;
60       if((int)*num != 5) printf("Num %x %d", num, *num);
61    }
62
```

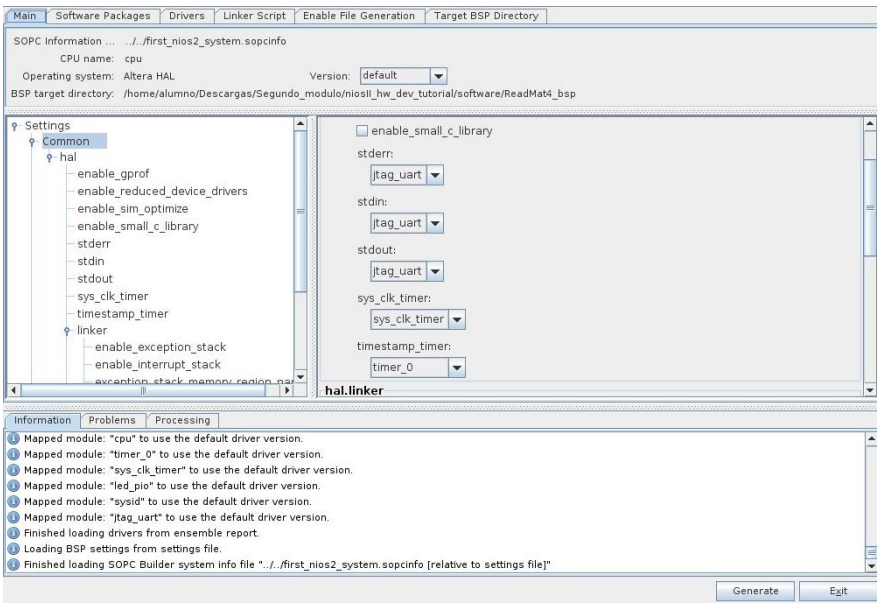
En el recorremos y escribimos en las posiciones de memoria que corresponden a la SDRAM.

Importante: Debemos tener en cuenta que para probar la sram o la sdram debemos tener las configuradas las siguientes opciones:

En ***Properties ->Nios II BSP Properties*** tener marcados ***Reduced devices*** **drives** y **Small C library**



Luego, tener seleccionado en el ***BSP editor*** la opción de **timer_0**.



Timestamp:

Añadiremos en el count_binary el siguiente código:

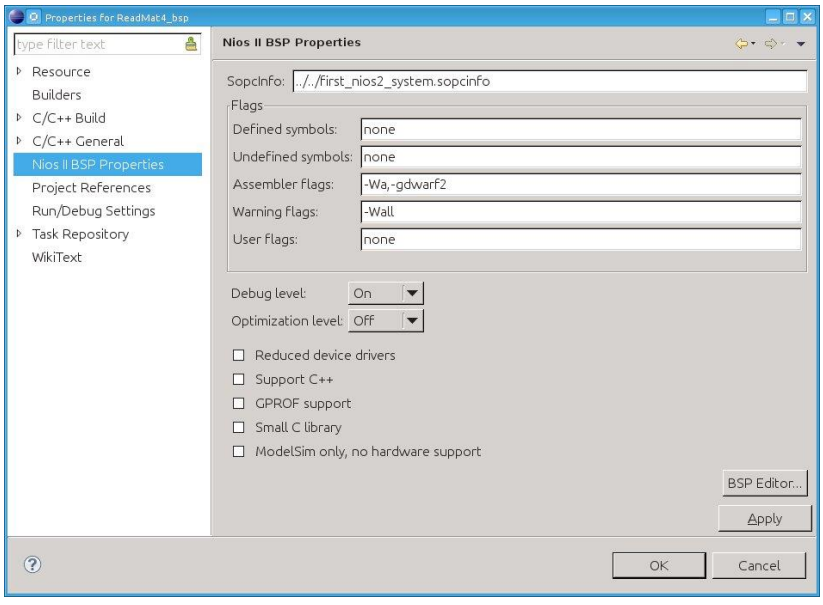
```
#include <stdio.h>
#include "sys/alt_timestamp.h"
#include "alt_types.h"

int main (void)
{
  alt_u32 time1;
  alt_u32 time2;
  alt_u32 time3;

  if (alt_timestamp_start() < 0)
  {
    printf ("No timestamp device available\n");
  }
  else
  {
    time1 = alt_timestamp();
    func1(); /* first function to monitor */
    time2 = alt_timestamp();
    func2(); /* second function to monitor */
    time3 = alt_timestamp();

    printf ("time in func1 = %u ticks\n",
            (unsigned int) (time2 - time1));
    printf ("time in func2 = %u ticks\n",
            (unsigned int) (time3 - time2));
    printf ("Number of ticks per second = %u\n",
            (unsigned int)alt_timestamp_freq());
  }
  return 0;
}
```

En el calcula el tiempo que tarda en ejecutar cada función func1, func2. Devuelve además la frecuencia del reloj de la cpu utilizada.

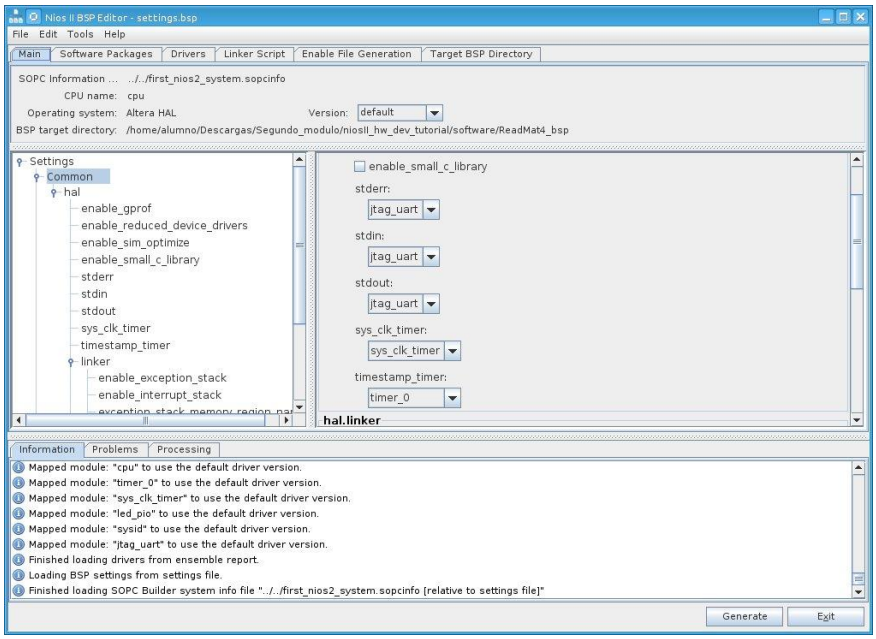
Implementación del programa:

Para implementar la matriz creamos un nuevo proyecto al que llamaremos Readmat5. Luego configuramos nuestro proyecto:

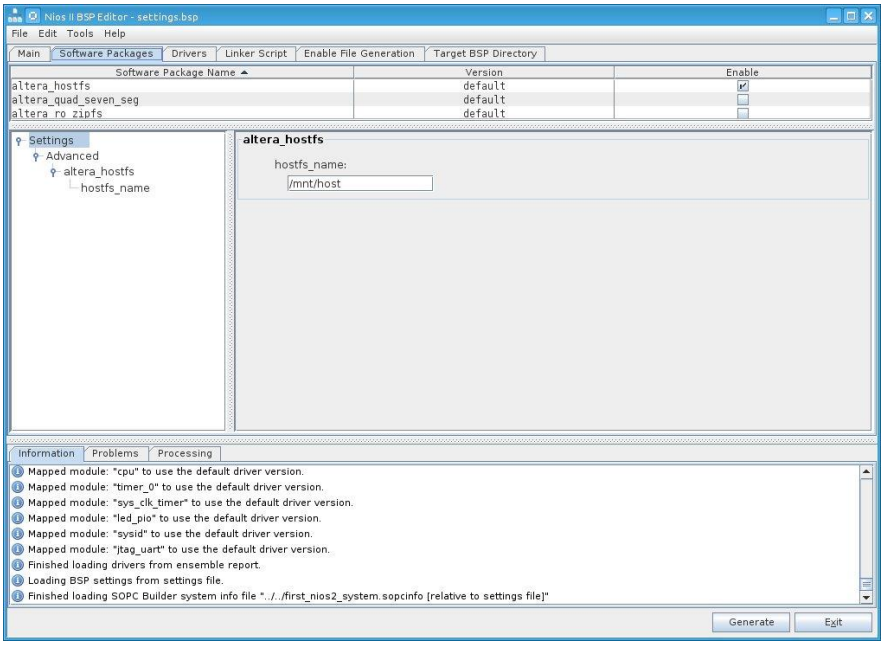
- Desmarcamos en ***Properties ->Nios II BSP Properties*** todos los campos tener la librería completa de c.



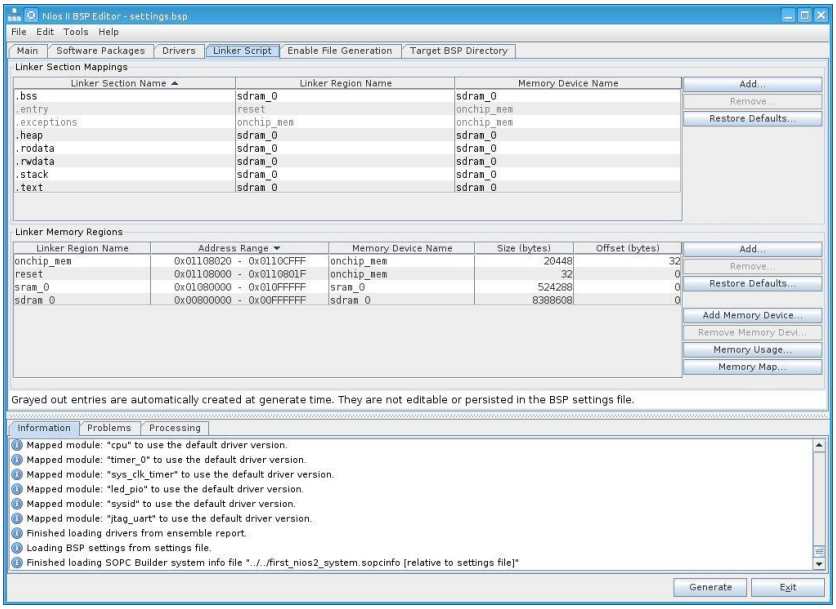
- Luego seleccionamos ***BSP editor*** la opción de **time_0**.



- Posteriormente en la pestaña ***Software Packages*** habilitamos **altera hostfs** para permitir acceder a archivos de la carpeta del proyecto. Podemos especificar una ruta, pero en este caso dejamos la predeterminada */mnt/host*.



- Por último, en la pestaña ***Linker Script*** asignaremos todo para que esté guardado en la **sdram_0**.



Una vez acabada la configuración añadimos los métodos de lectura(pgmread) y escritura(pgmwrite) facilitados por el profesor. Escogiendo el método de lectura y escritura por filas. Luego realizamos el código en c:

```
int main()
{
  printf('Hello from Nios II!\n');

  int w = 512, h = 512;
  const char* filename = '/mnt/host/inputImage.pgm';

  filtrado(filename, w, h);

  printf('Termino");

  return 0;
}

int filtrado(const char* filename, int w, int h)
{
    int i,j;
    int filtro_w = 3, filtro_h = 3;

    unsigned char **filtro;
    filtro = (unsigned char**)calloc((filtro_w), sizeof(unsigned char*));

    for(i=0; i<w; i++)
        filtro[i] = (unsigned char**)calloc((filtro_h), sizeof(unsigned char*));

    filtro[0][0] = 20;
    filtro[0][1] = 50;
    filtro[0][2] = 20;
    filtro[1][0] = 50;
    filtro[1][1] = 60;
    filtro[1][2] = 50;
    filtro[2][0] = 20;
    filtro[2][1] = 50;
    filtro[2][2] = 20;

    unsigned char **data_dest;
    data_dest = (unsigned char**)calloc((w-2), sizeof(unsigned char*));
```

```
for (i=0 ; i<w; i++)
    data_dest[i] = (unsigned char*)calloc((h-2), sizeof(unsigned char));

unsigned char **data = pgmread(filename, &w, &h);

alt_u32 time1;
alt_u32 time2;
alt_u32 time3;

if(alt_timestamp_start() < 0)
{
    printf("No timestamp device available\n");
}
else
{
    time1 = alt_timestamp();

    for(i=1; i<(w-1); i++){
        for(j=1; j<(h-1); j++){
            char pixel = 0;
            int calculo = (int)filtro[0][0] * (int)data[i-1][j-1];
            calculo += (int)filtro[0][1] * (int)data[i][j-1];
            calculo += (int)filtro[0][2] * (int)data[i+1][j-1];
            calculo += (int)filtro[1][0] * (int)data[i-1][j];
            calculo += (int)filtro[1][1] * (int)data[i][j];
            calculo += (int)filtro[1][2] * (int)data[i+1][j];
            calculo += (int)filtro[2][0] * (int)data[i-1][j+1];
            calculo += (int)filtro[2][1] * (int)data[i][j+1];
            calculo += (int)filtro[2][2] * (int)data[i+1][j+1];

            pixel = calculo/900;

            data_dest[i-1][j-1] = pixel;
        }
    }

    time2 = alt_timestamp();

    printf("Time filtering the image = %u ticks\n", (unsigned int) (time2 -time1));
    printf("Number of ticks per second = %u\n", (unsigned int)alt_timestamp_freq());
}

const char* filedest = "/mnt/host/ImpageDest.pgm";
pgmwrite(filedest, w-2, h-2, data_dest, NULL, 1);

return (0);
```

Para realizar el difuminado de la matriz creamos una función filtrado a la que se le pasa el nombre de la imagen y el tamaña (ancho y alto). Luego creamos una char ** del tamaño de la matriz original, restándole 2 pixeles de ancho y 2 pixeles de alto. También creamos una matriz de 3x3, la cual rellenamos con los valores de suavizado deseados.

Posteriormente leemos la imagen con el método pgmread el cual devuelve un char **. Una vez leída recorremos la imagen original para calcular los nuevos pixeles a partir de la matriz de suavizado. Con ello rellenamos el char ** que corresponde a la imagen destino. Finalmente la guardamos con el método pgmwrite.

Por último, compilamos el proyecto, conectamos la placa utilizando el ***Programmer*** y ejecutamos el programa. El resultado se puede ver en las siguientes imágenes:

Imagen original

Imagen final

Conclusiones:

- Al realizar pruebas con el timestamp pudimos observar que el menor tiempo se obtenía con una frecuencia de 50 MHz para la que mostraba un tiempo de 600 millones de ciclos.

- Para el método de escritura escogido, la columna final de la imagen se escribe al inicio de la imagen resultante.

- Los valores elegidos son para un difuminado muy suave.

- Es indispensable la implementación de la SDRAM ya que en caso contrario no tenemos suficiente espacio para almacenar el programa, la matriz origen, destino y la matriz 3x3.

- La mayoría dificultad del proyecto fue la familiarización con la FPGA y el programa Quartus II.

Bibliografía:

Introducción: https://campusvirtual.ull.es/1617/mod/resource/view.php?id=165995

Desarrollo NiosII: https://campusvirtual.ull.es/1617/mod/resource/view.php?id=165996

Procesador NiosII: https://campusvirtual.ull.es/1617/mod/resource/view.php?id=165999

Programación NiosII: https://campusvirtual.ull.es/1617/mod/resource/view.php?id=166000

Programación HAL: https://campusvirtual.ull.es/1617/mod/resource/view.php?id=166001

Módulos de librerías: https://campusvirtual.ull.es/1617/mod/resource/view.php?id=166003

Utilización SDRAM: https://campusvirtual.ull.es/1617/mod/resource/view.php?id=166004

Asignación de pines: https://campusvirtual.ull.es/1617/mod/resource/view.php?id=166008