California State University, Fresno
Lyles College of Engineering
Electrical and Computer Engineering Department

TECHNICAL REPORT

Assignment: Final Project

Experiment Title: RTOS, SD card. VGA

Course Title: ECE 178 (Embedded Systems)

Instructor: Dr. Reza Raeisi

Prepared by: Omer Al Sumeri , Aryan Singh, Oludayo Iredele

Date Submitted: 12/08/2023

INSTRUCTOR SECTION

Comments: _____
_____
_____
_____
_____
_____
_____

Final Grade: _____

TABLE OF CONTENTS

1. Objective.....3
2. Hardware Requirements.....3
3. Background.....3
4. Project Overview.....6
5. Procedure.....6
6. Analysis.....35
7. Conclusion.....42

1. Objective

This project seeks to integrate diverse concepts learned in class, aiming to demonstrate a cohesive understanding of the subject matter.The system designed in this project interfaces with the SD card and VGA interface to display images on the screen. Using RTOS, the system is able to perform other tasks as well.

2. Hardware Requirements

- DE2-115 FPGA board
- Computer with the Intel Quartus Lite program
- Computer with Nios II Software Build Tools for Eclipse
- VGA cable
- SD card
- Monitor

3. Background

In order to accomplish this lab, the processor designed in the previous lab will be required, also knowledge of any HDL such as verilog and the ability to program in c/c++ language to write the software application. This will be needed to create the custom system using Quartus Prime and Qsys. Quartus Prime is a software that enables developers to perform analysis and synthesis of HDL designs.

The RTOS employed in this project serves as the operational backbone, enabling seamless multitasking and efficient resource management. This real-time capability is crucial for handling dynamic user inputs and executing tasks with precision.

The DE2-115 board is going to be used to implement this final project, so in depth knowledge of the different features of the board is important. The figure below shows the board and its major features.

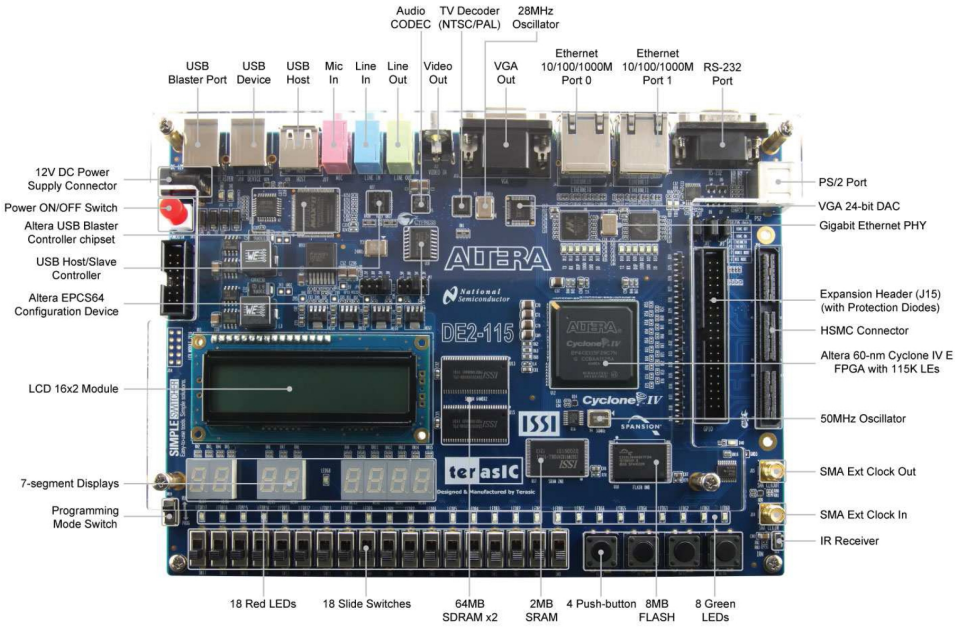

Figure 1: DE2-115 board.

One of the major peripheral interfaces that will be used is the VGA port, so our system will need to implement the vga ip on the fpga. The video-out port supports a screen resolution of 640x480 which has strict timing requirements that need to be met in order for the vga to work properly. When adding this feature to the custom system, understanding how the pixel buffer, character buffer, DMA controller, RGB sampler, and double buffer work will be important.

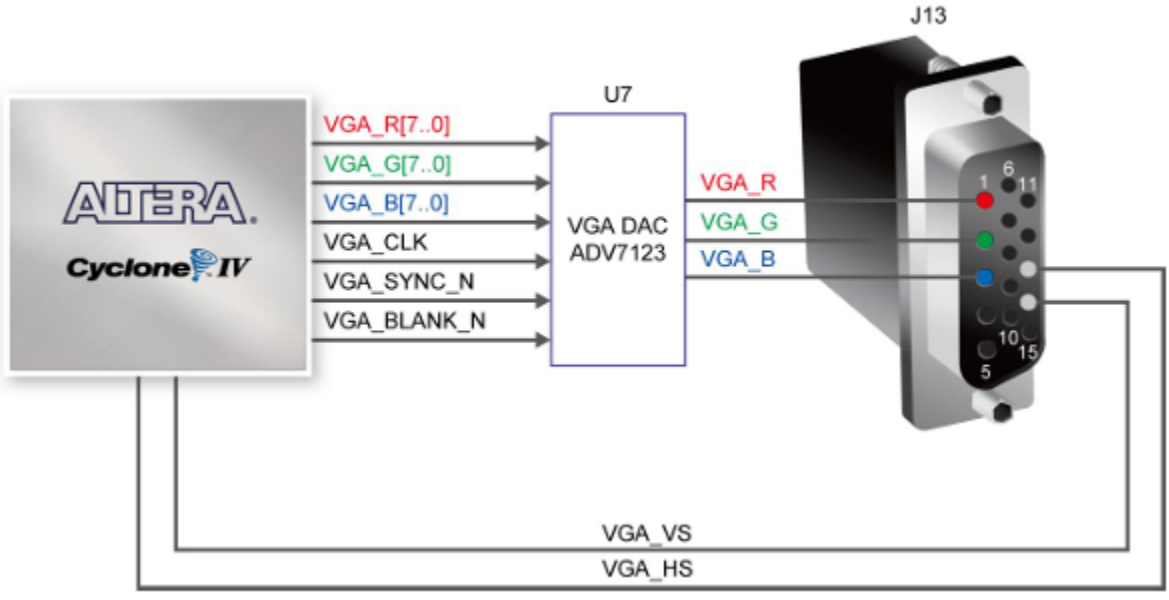

Figure 2: Connection between the FPGA and VGA.

The DE2-115 board also comes with a controller for interfacing (read/write) to an SD card. A HAL interface is also provided to access file systems stored on the SD card which must be formatted using the FAT-16 standard.

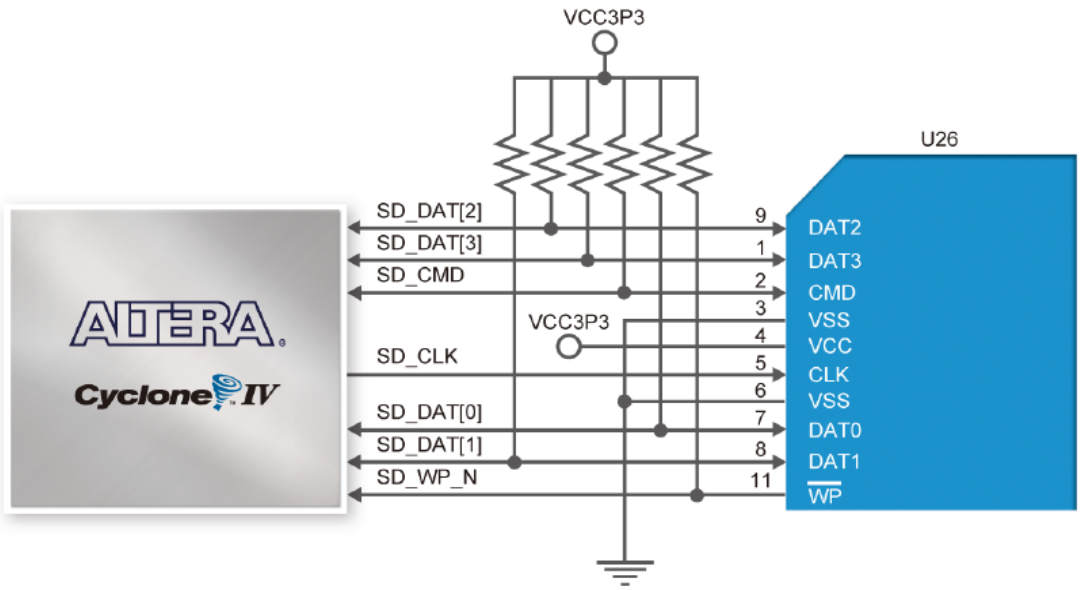

Figure 3: Connections between FPGA and SD Card.

The DE2-115 board typically includes a set of push buttons, often labeled as KEY0 to KEY3 or similar. These buttons can be used as inputs to the FPGA, allowing users to interact with the board. Mechanical buttons can produce electrical noise when pressed or released, leading to multiple transitions that can be misinterpreted, so the board solves this problem with the use of schmitt triggers.

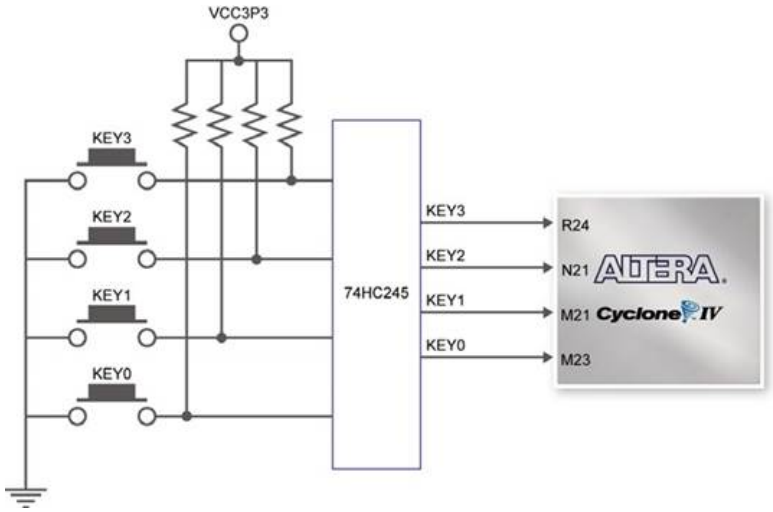

Figure 4: DE2-115 Key Push Buttons connections.

4. Project Overview

The primary objectives encompass the dynamic display of images stored within the FPGA on an external monitor and the read/write operations involving an SD card.

Program 1: RTOS implementation
This program implements a simple RTOS that enables multiple tasks to be scheduled and performed based on priority.

Program 2: SD Card Read/Write Operations
The second major task involves seamless interaction with an SD card. The RTOS enables the execution of read and write operations, providing a dynamic interface for data storage and retrieval.

Program 3: VGA image display
The primary task involves reading image files from an SD card and displaying the images on an external monitor using the VGA interface.

5. Procedure

Open the Quartus Prime program and run through the process of creating a new project using the setup wizard. Figures 5 to 10 show the steps.

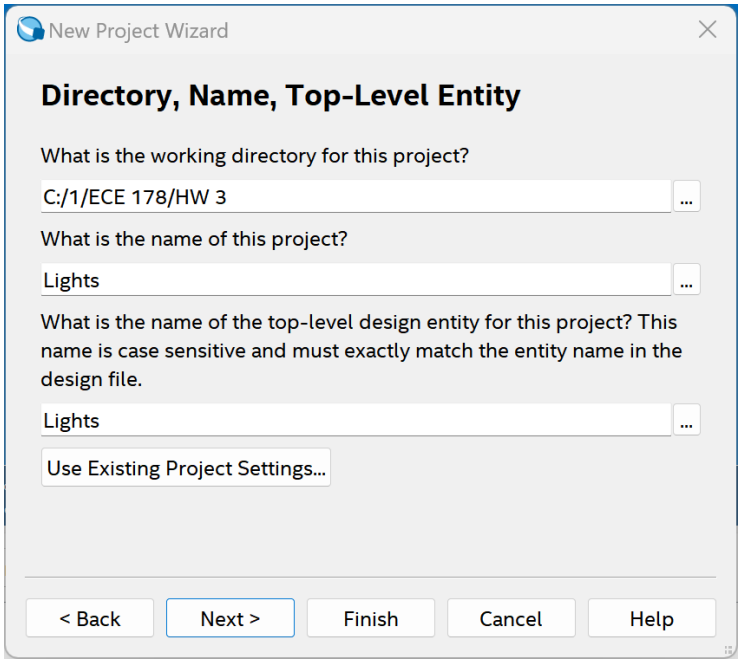

Figure 5: New Project Wizard.

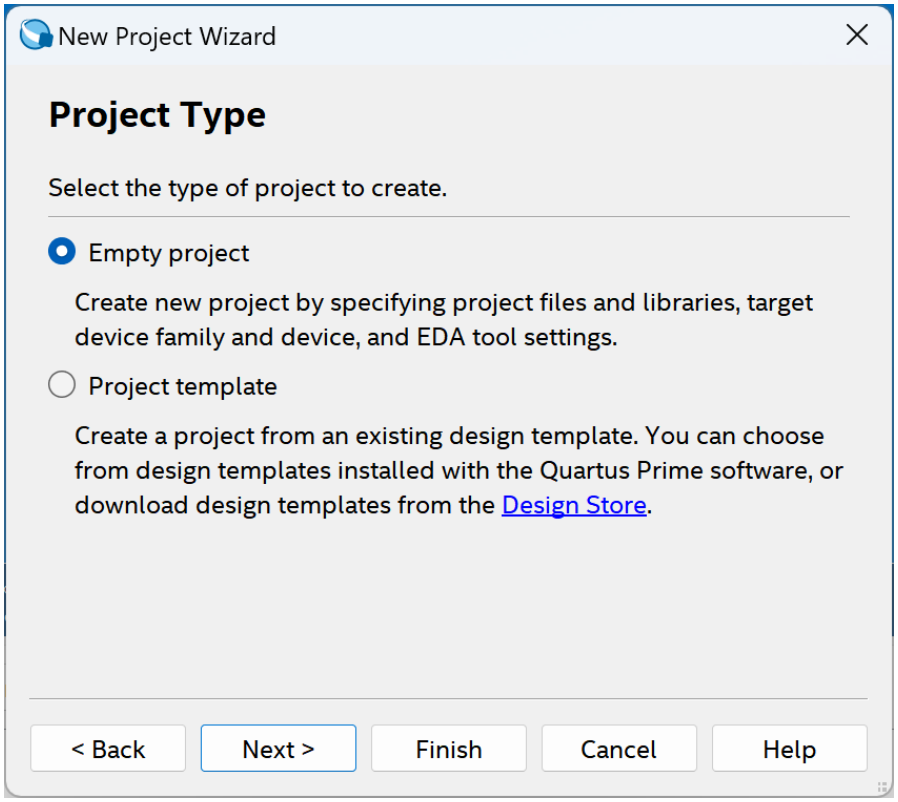

Figure 6: Project type window.

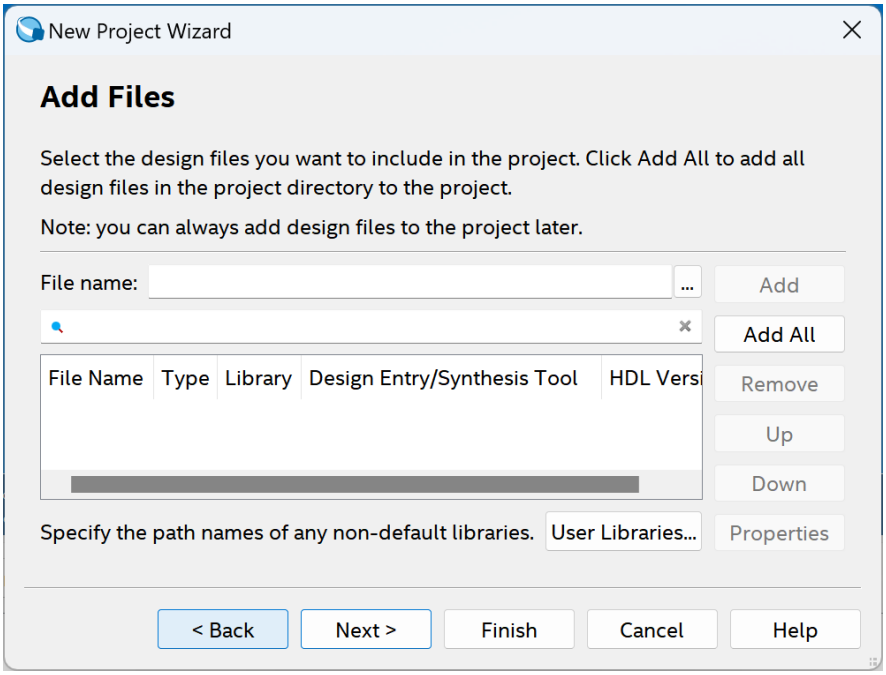

Figure 7: Add files window.

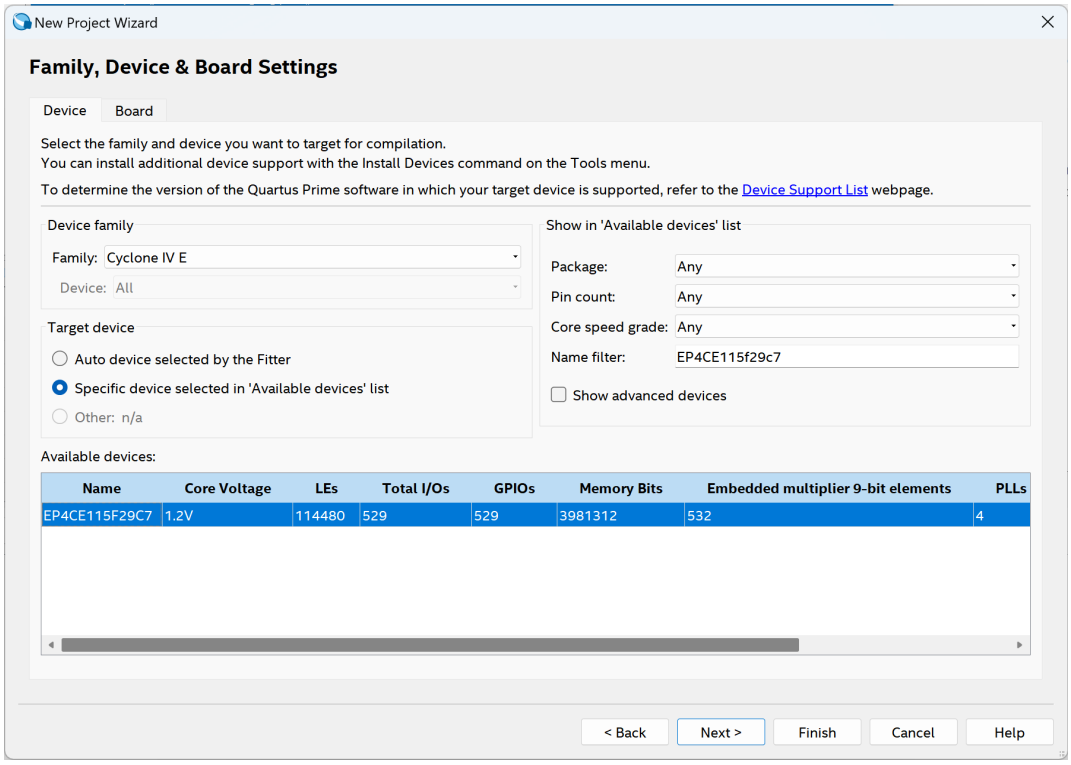

Figure 8: Family, Device and Board Settings.

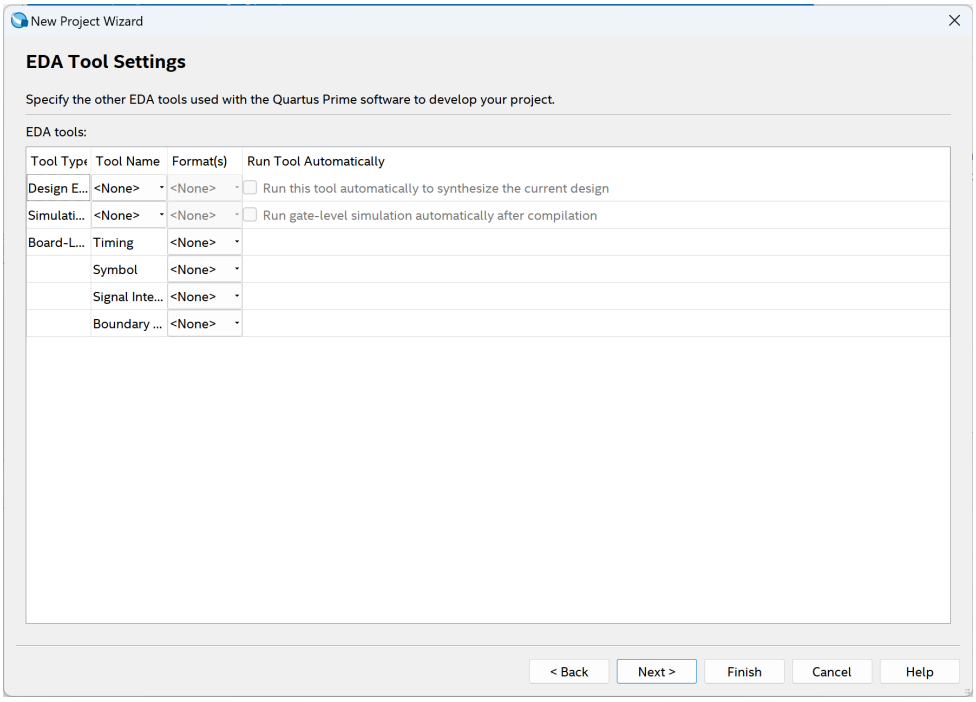

Figure 9: EDA Tool Settings.

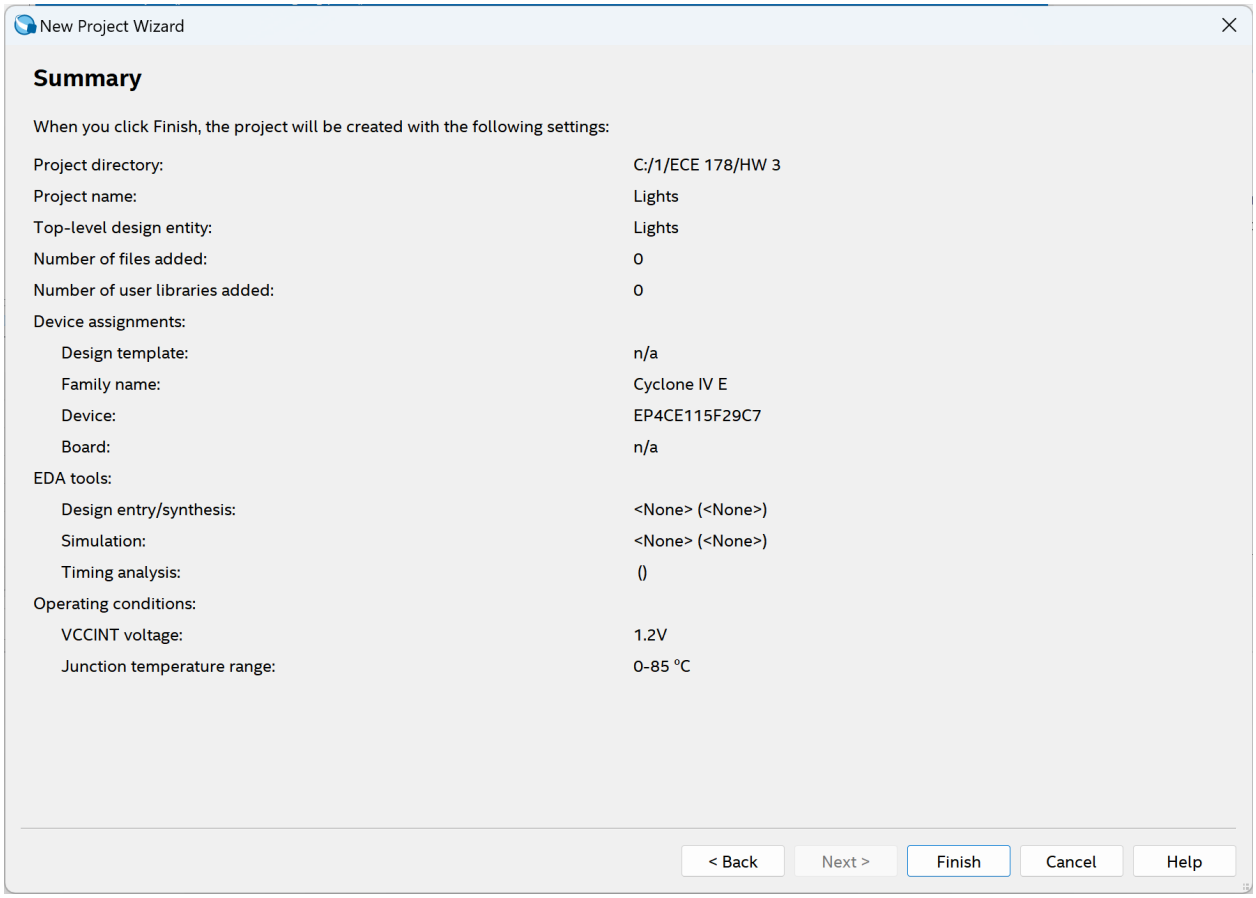

Figure 10: Summary window.

After setting up the project, it's time to design the custom system using qsys. Open Qsys by navigating to the ‘Tools’ dropdown menu.

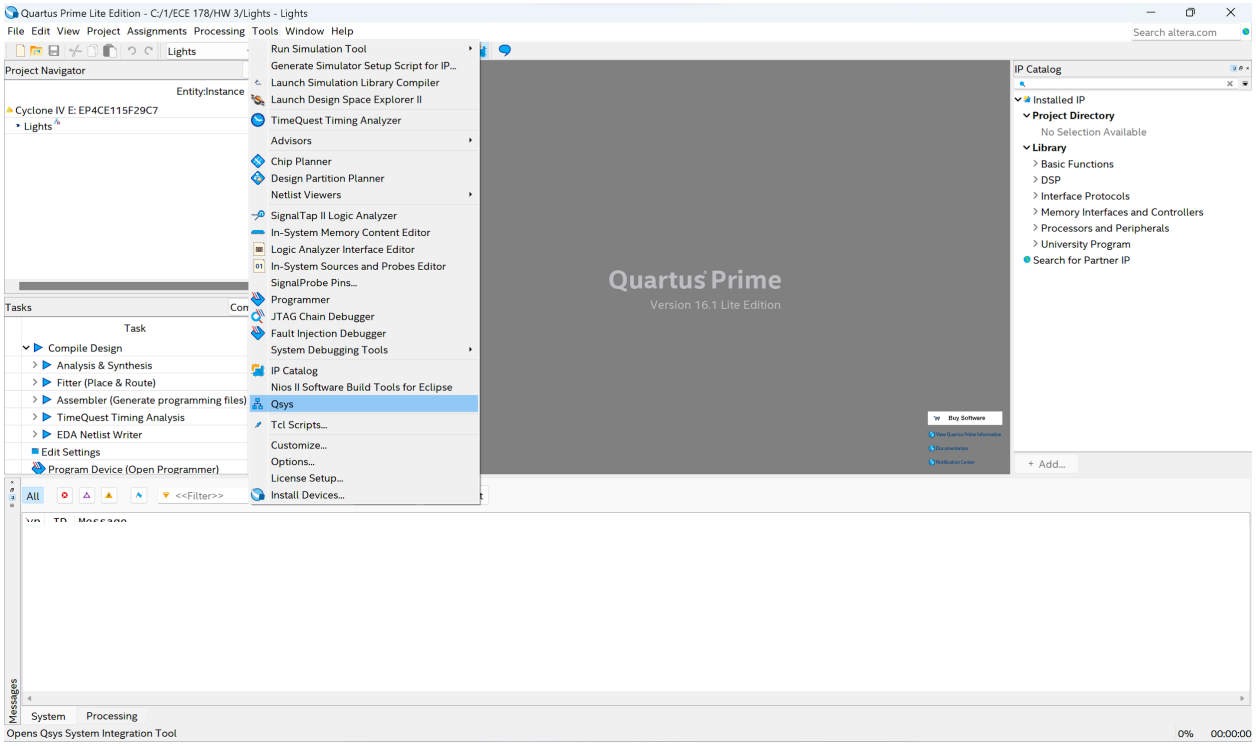

Figure 11: Tools menu.

The first components for a basic working system that was added to the custom design is the Nios II classic processor, specifically Nios II/e option. Also the On-chip memory (RAM or ROM) was added to serve as the system memory.

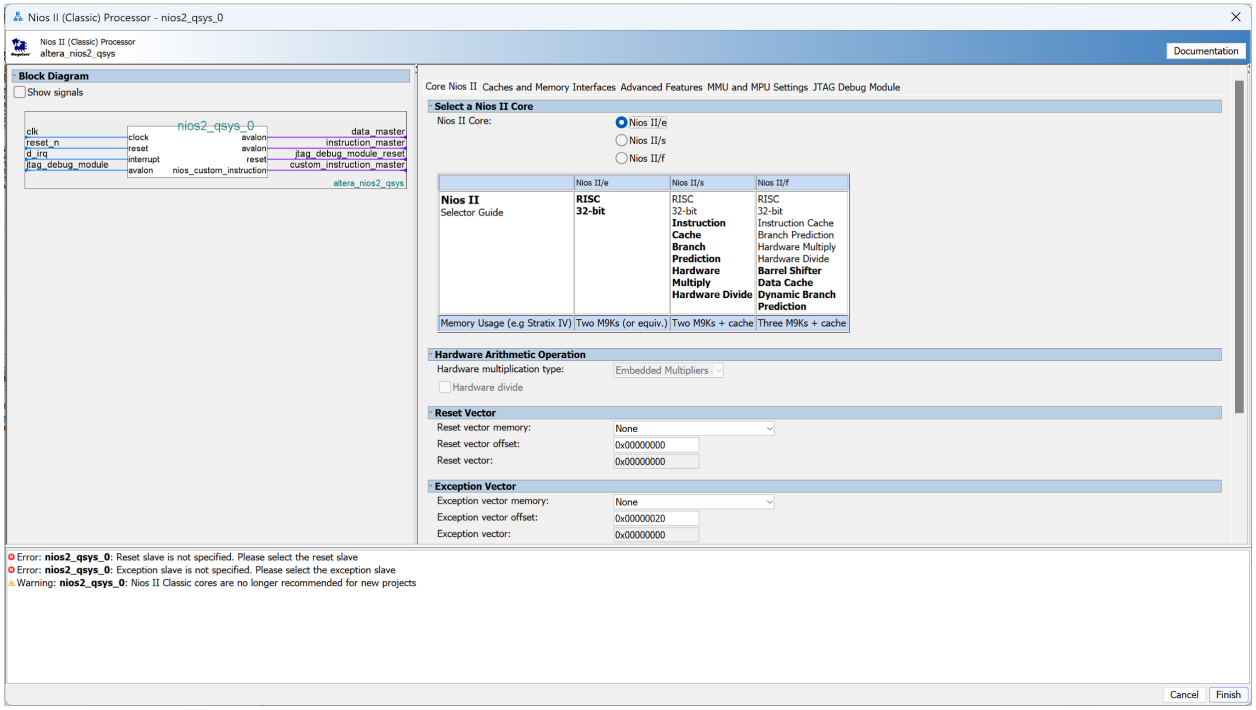

Figure 12: NIOS II/f processor configuration window.

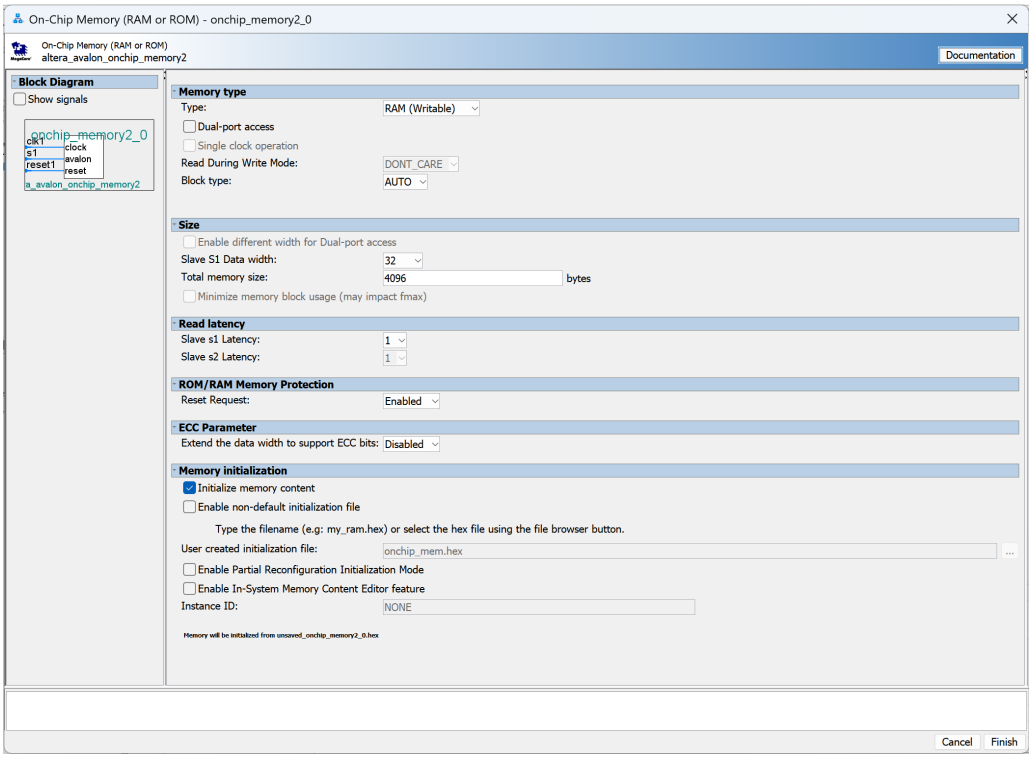

Figure 13: On-chip memory (RAM or ROM) configuration window.

The image presented through the video-out port originates from two distinct sources: a pixel buffer and a character buffer. The SRAM/SSRAM controller is going to act as the pixel frame buffer for the vga interface. The pixel buffer for the video-out port holds the data(color) for each pixel that will be displayed to the monitor.

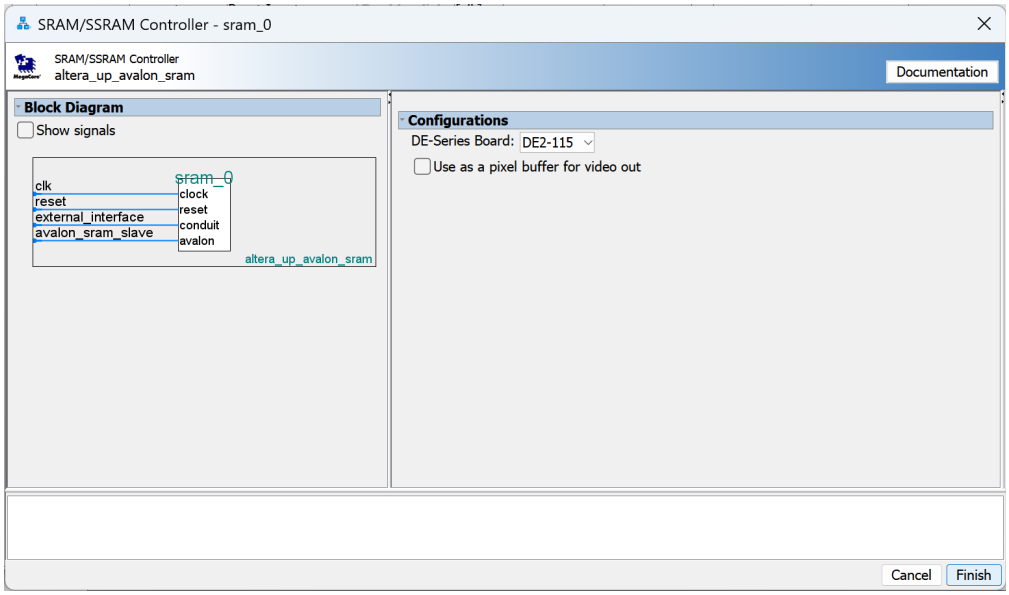

Figure 14: SRAM/SSRAM controller configuration window.

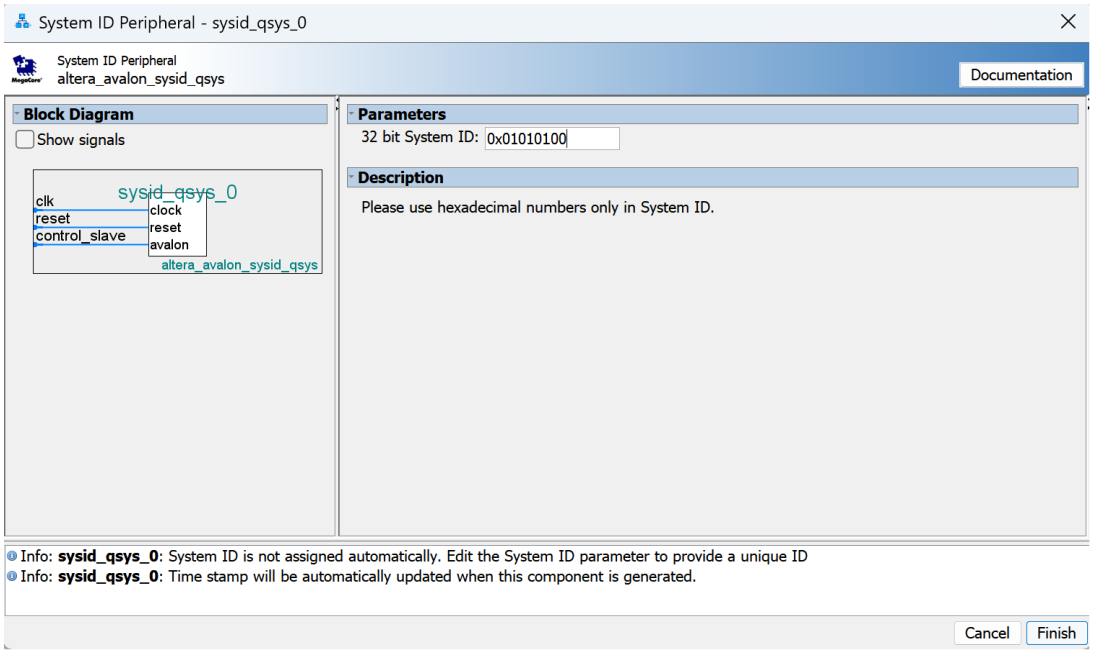

Figure 15: System ID configuration window.

There are five PIO components of different configurations to represent the switches, red Leds, green Leds, and the key pushbuttons. The figure 18 to 22 shows the configuration settings for the different PIOs.

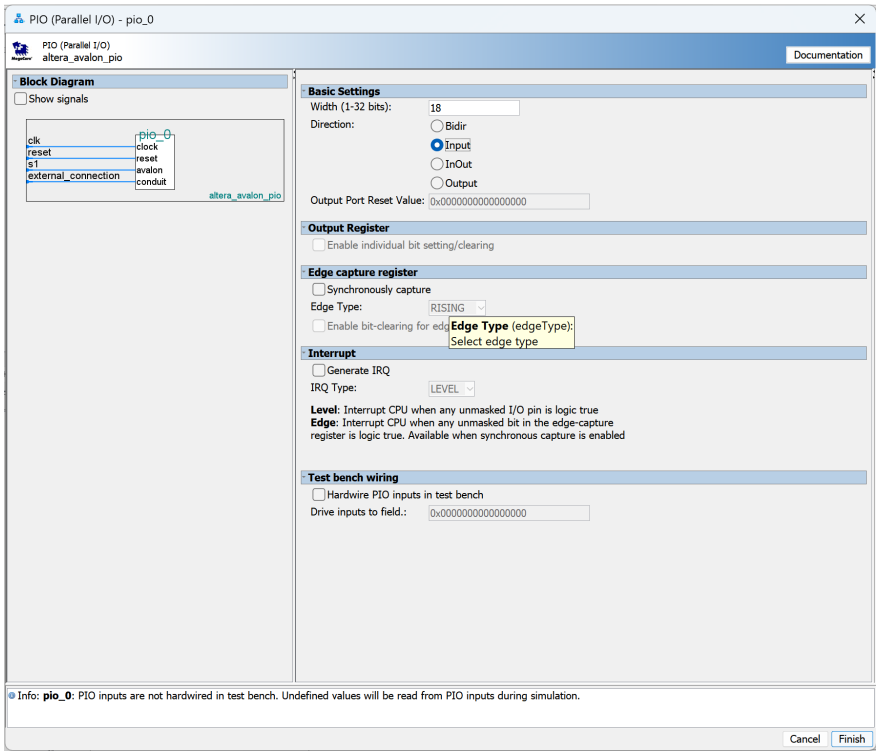

Figure 16: PIO switches configuration window.

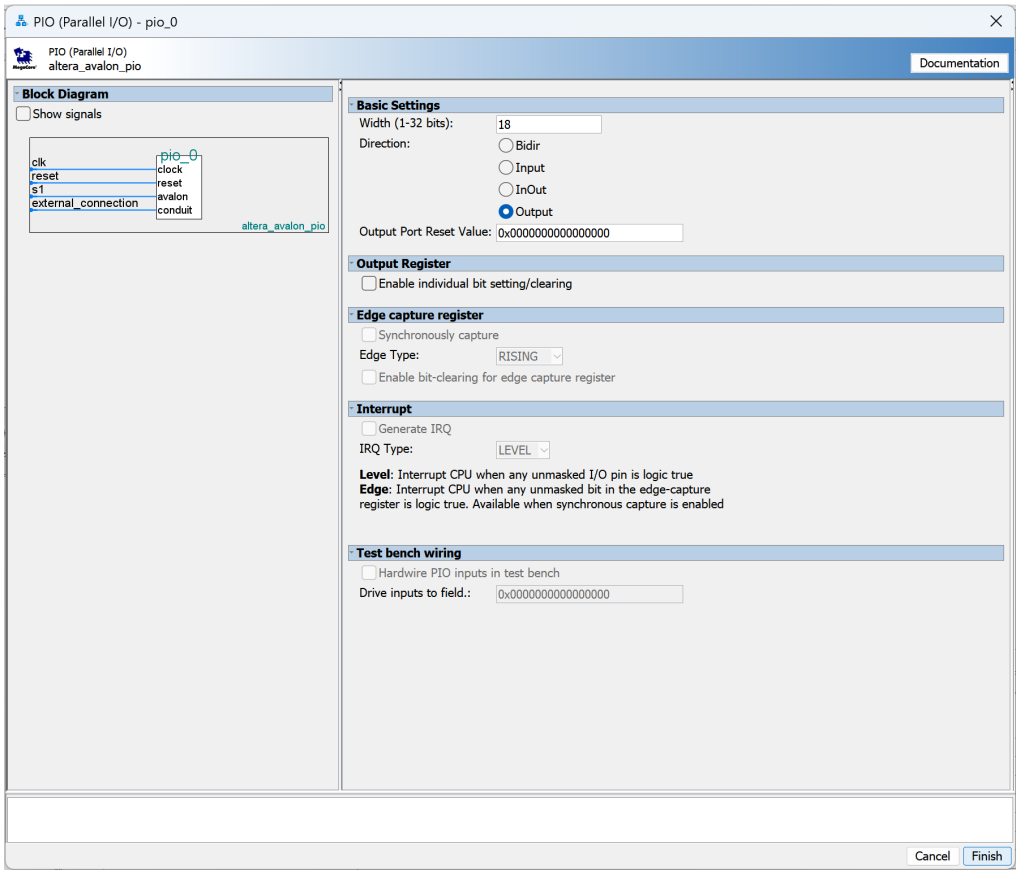

Figure 17: PIO Leds configuration window.

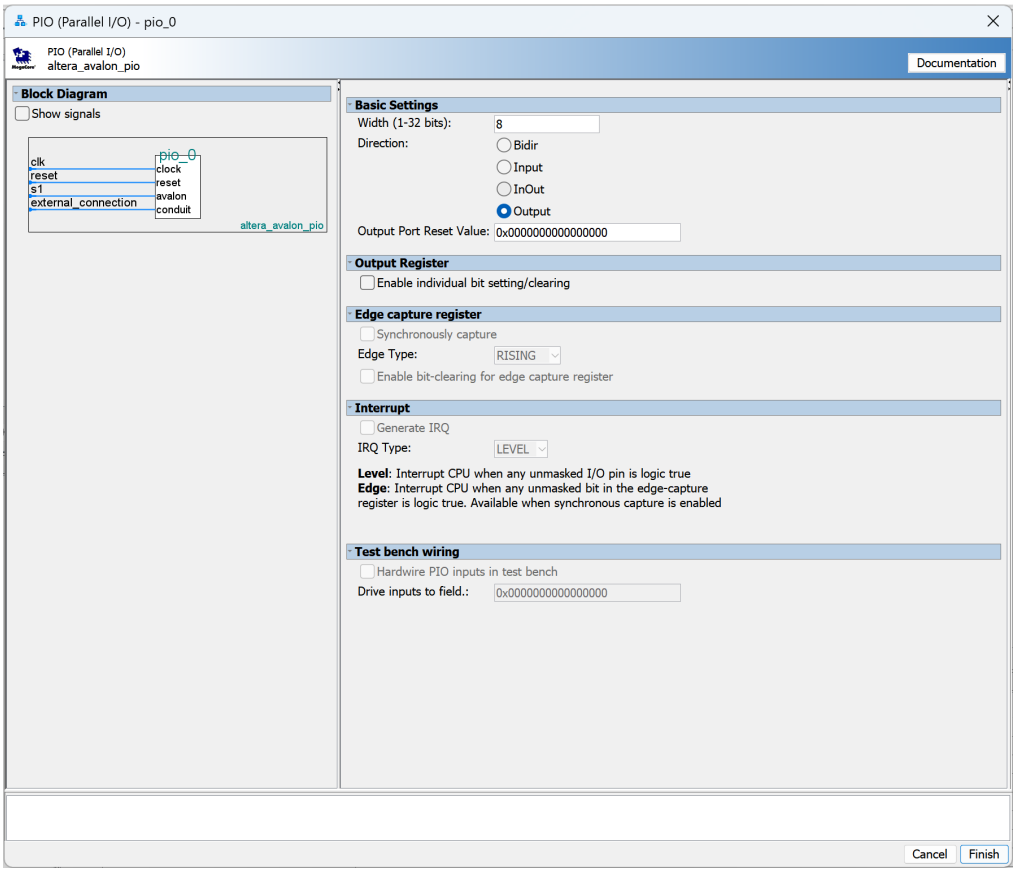

Figure 18: PIO green Leds configuration window.

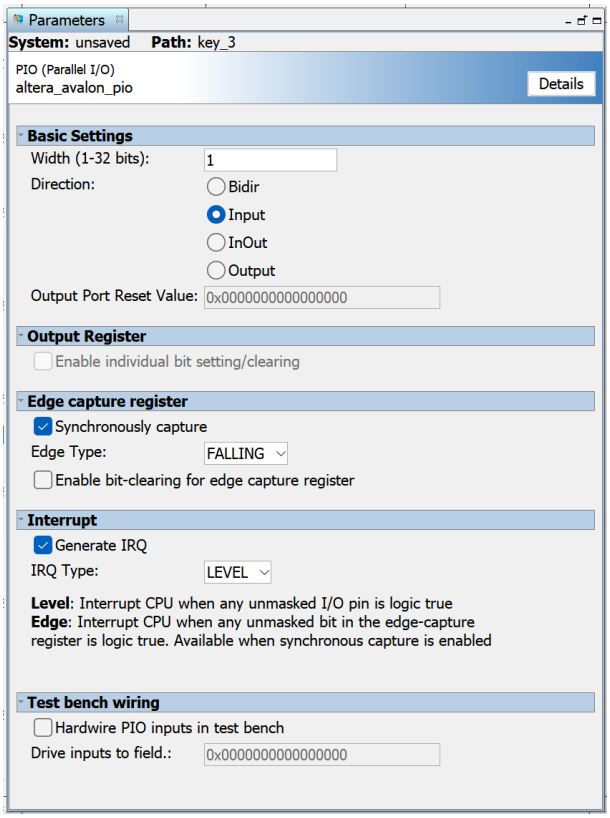

Figure 19: PIO keys configuration window.

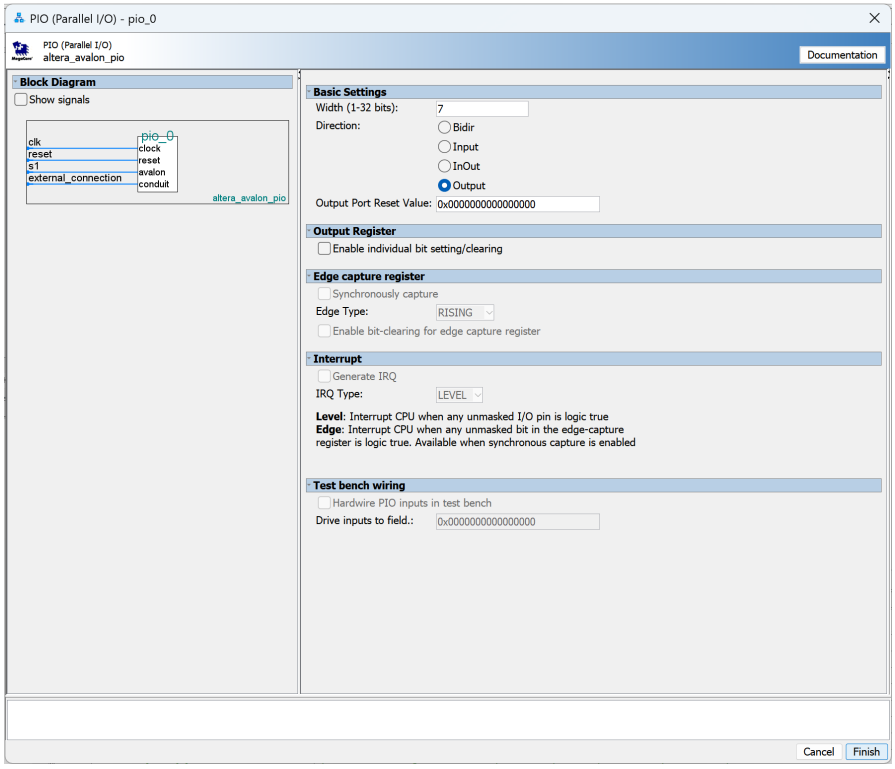

Figure 20: PIO seven segment display configuration window.

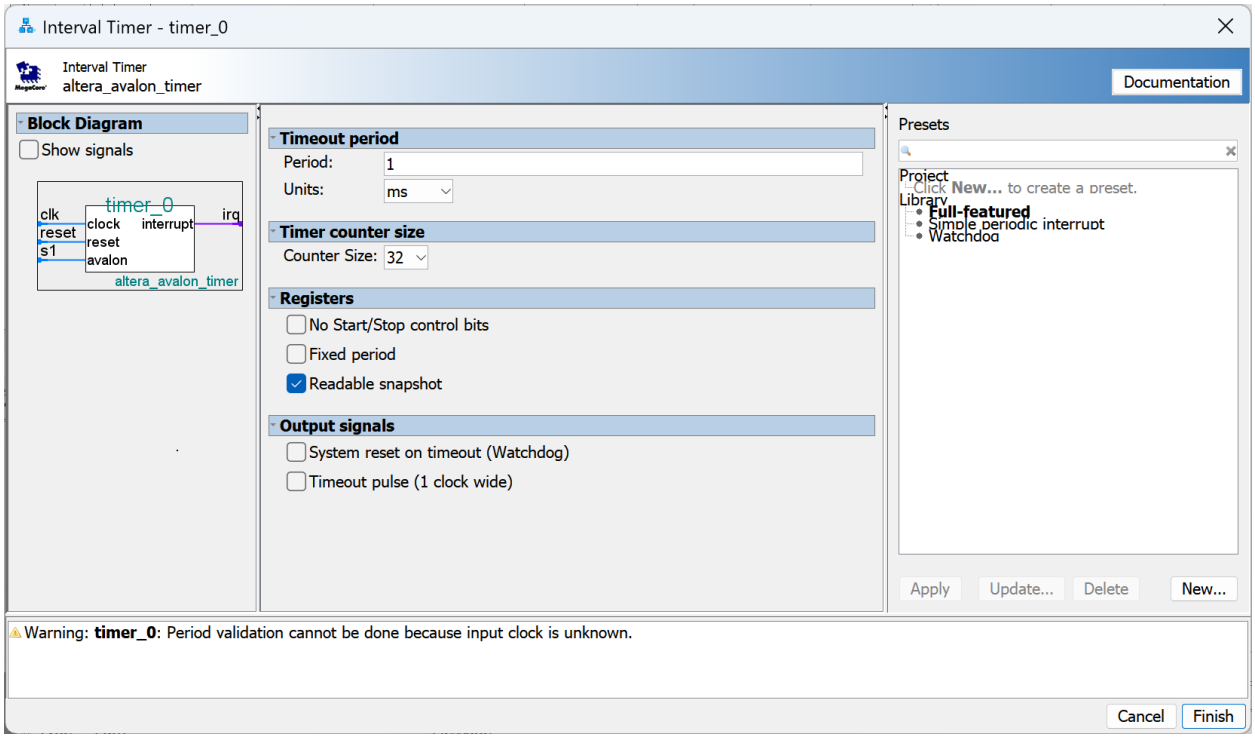

Figure 21: Interval timer configuration window.

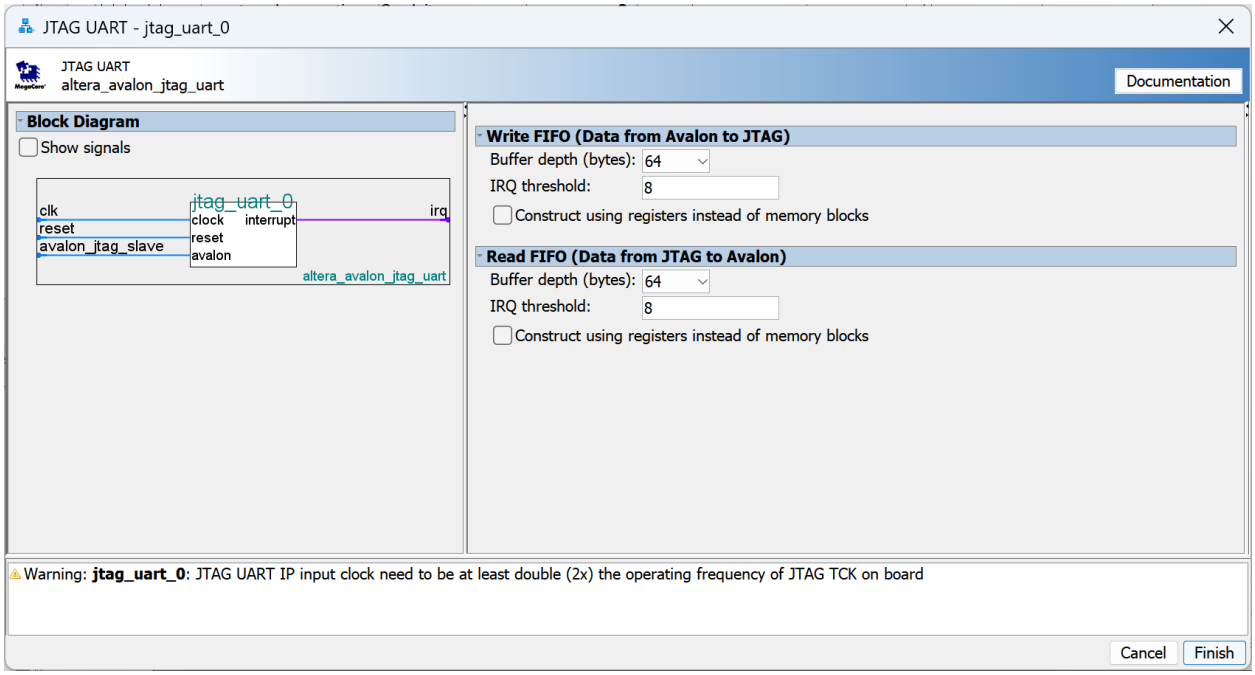

Figure 22: JTAG UART configuration window.

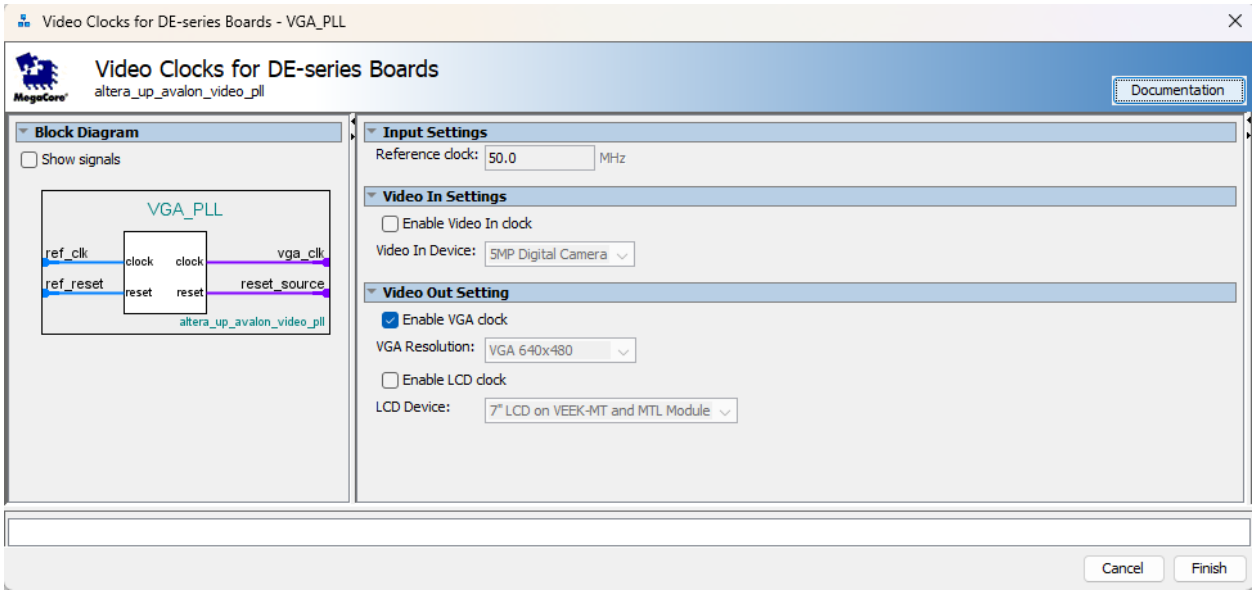

Figure 23: Video Clocks configuration window.

The pixel buffer controller efficiently transfers pixel data between memory and display devices, facilitating smooth and responsive graphics rendering.

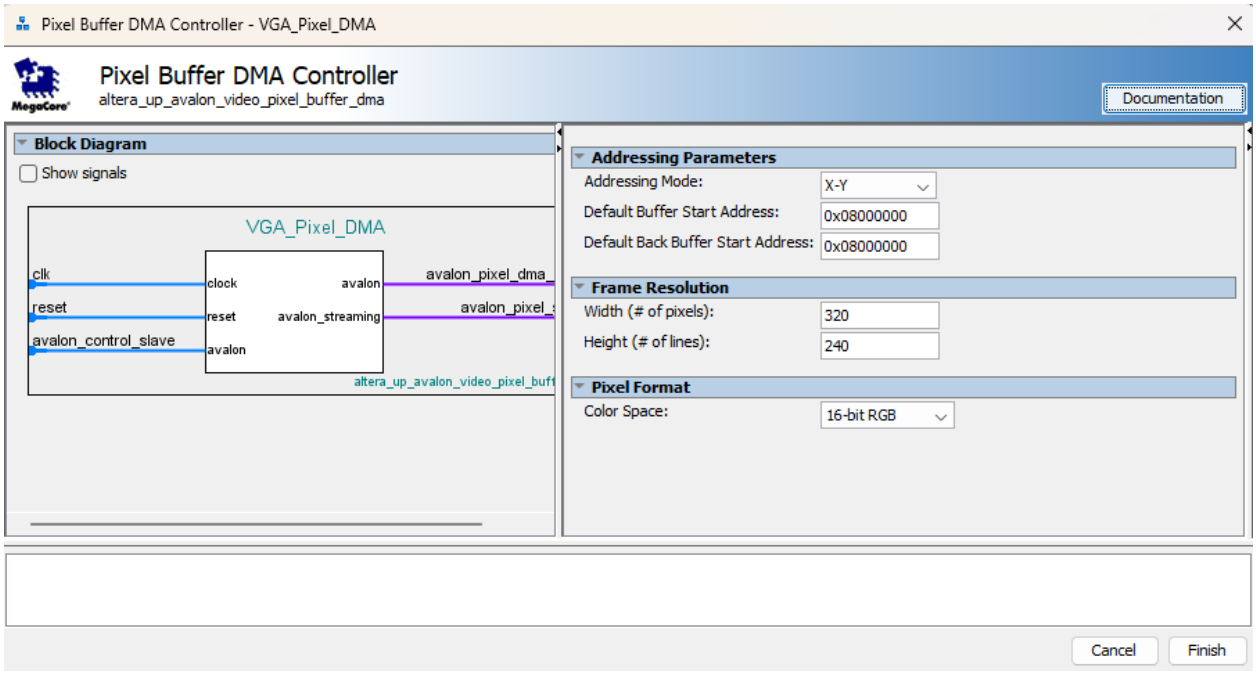

Figure 24: Pixel Buffer configuration window.

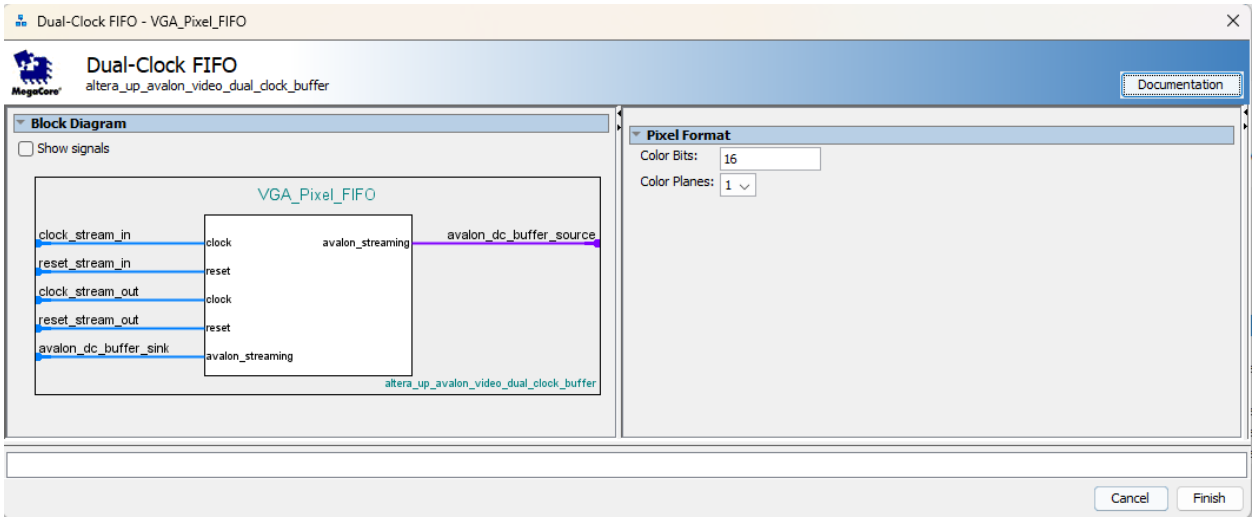

Figure 25: Dual clock FIFO configuration window.

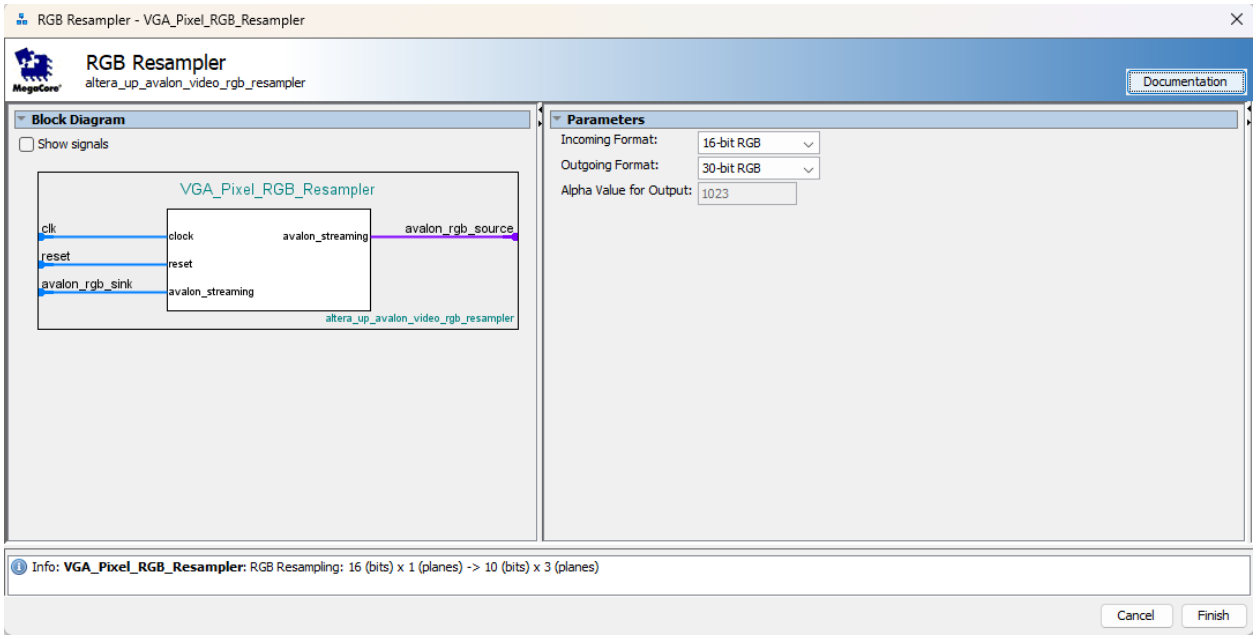

Figure 26: RGB resampler configuration window.

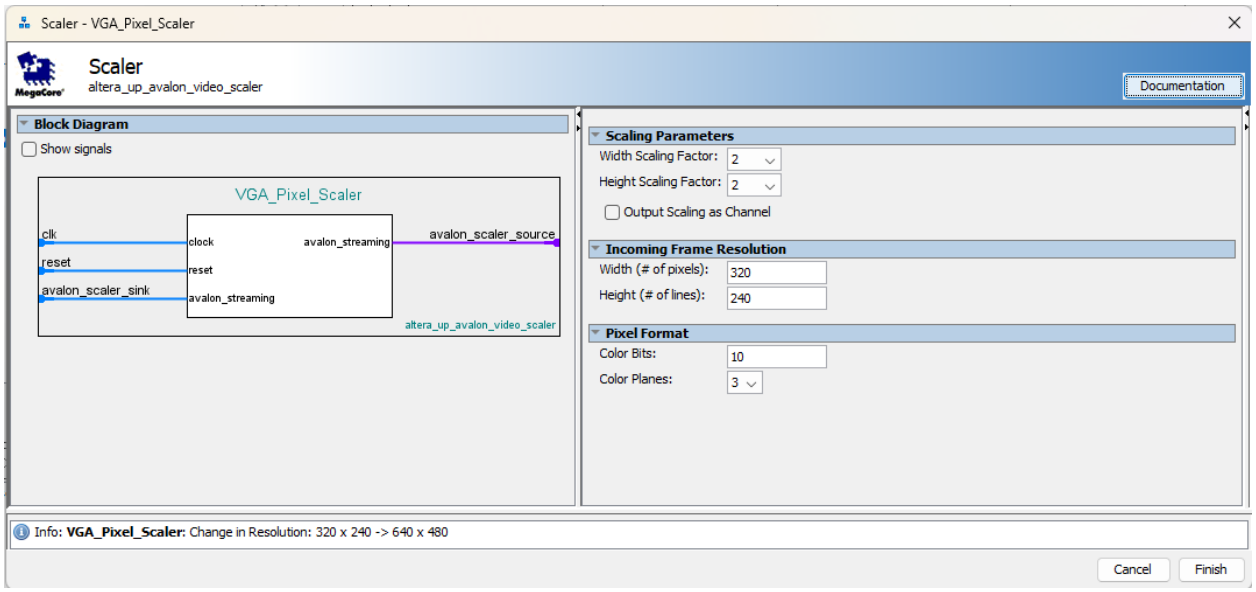

Figure 27: Scaler configuration window.

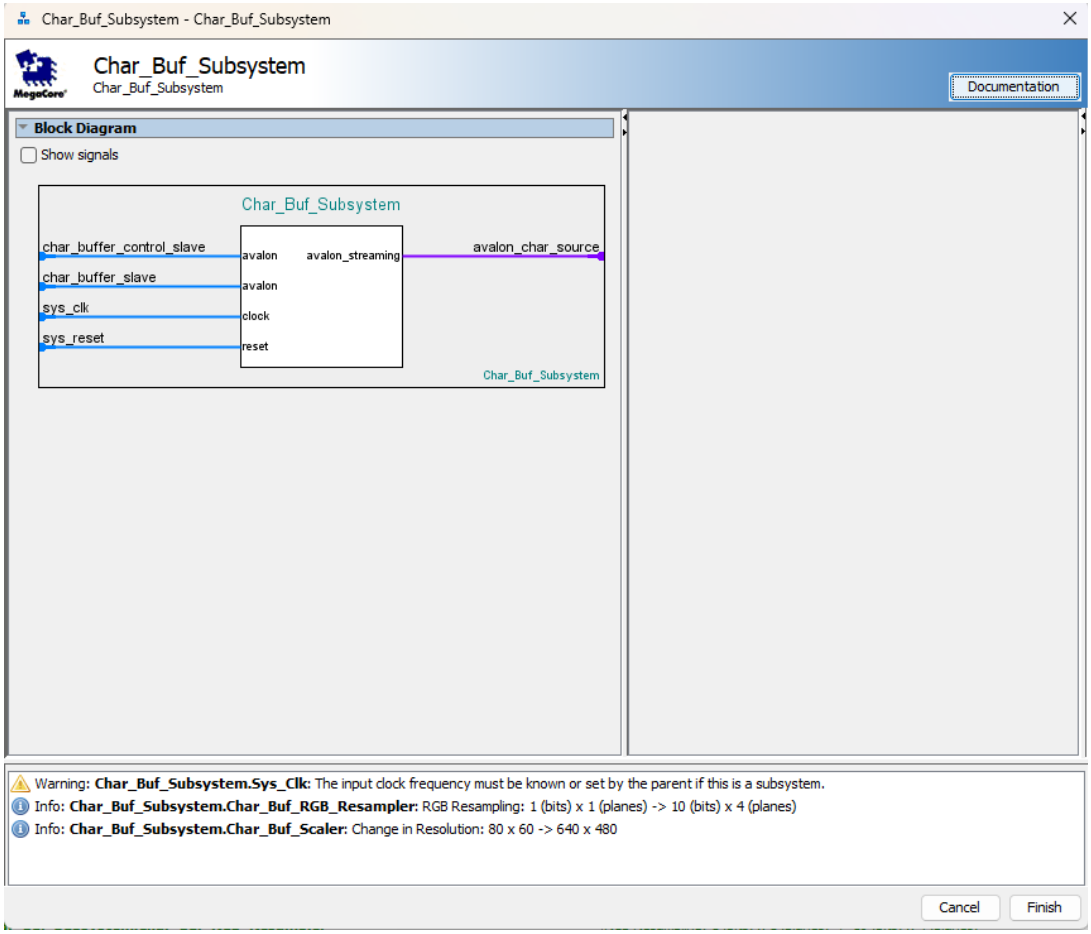

Figure 28: Character buffer configuration window.

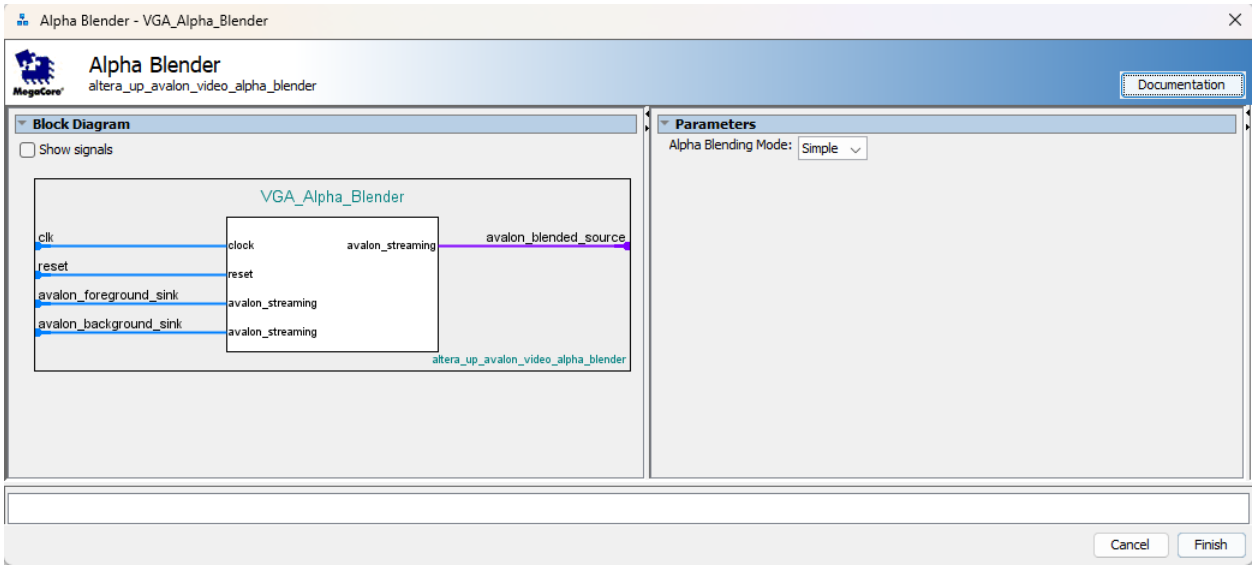

Figure 29: Alpha blender configuration window.

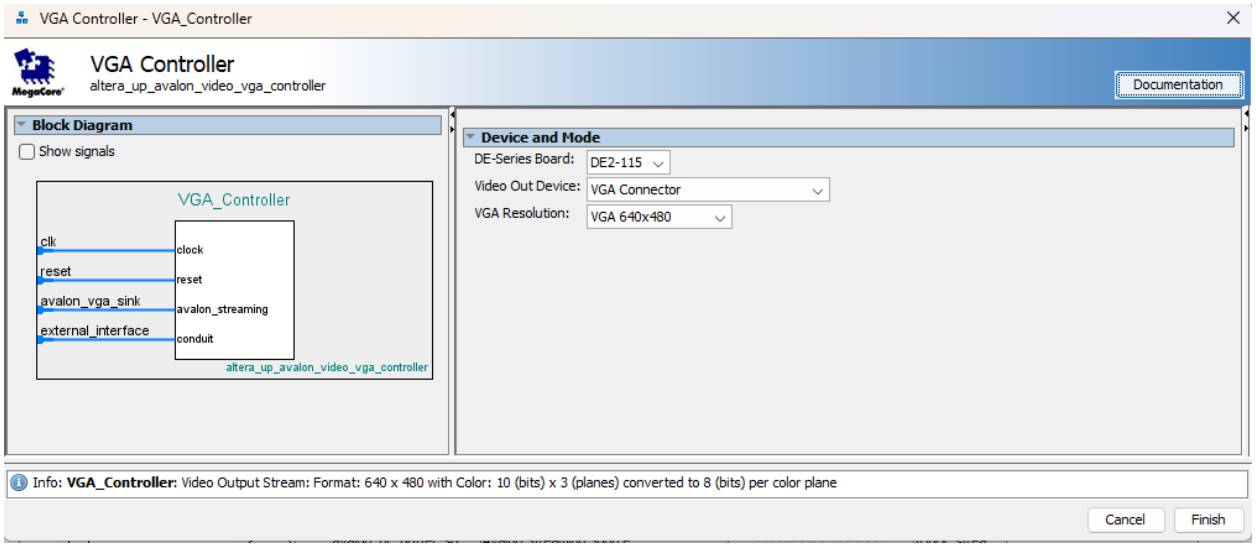

Figure 30: VGA controller configuration window.

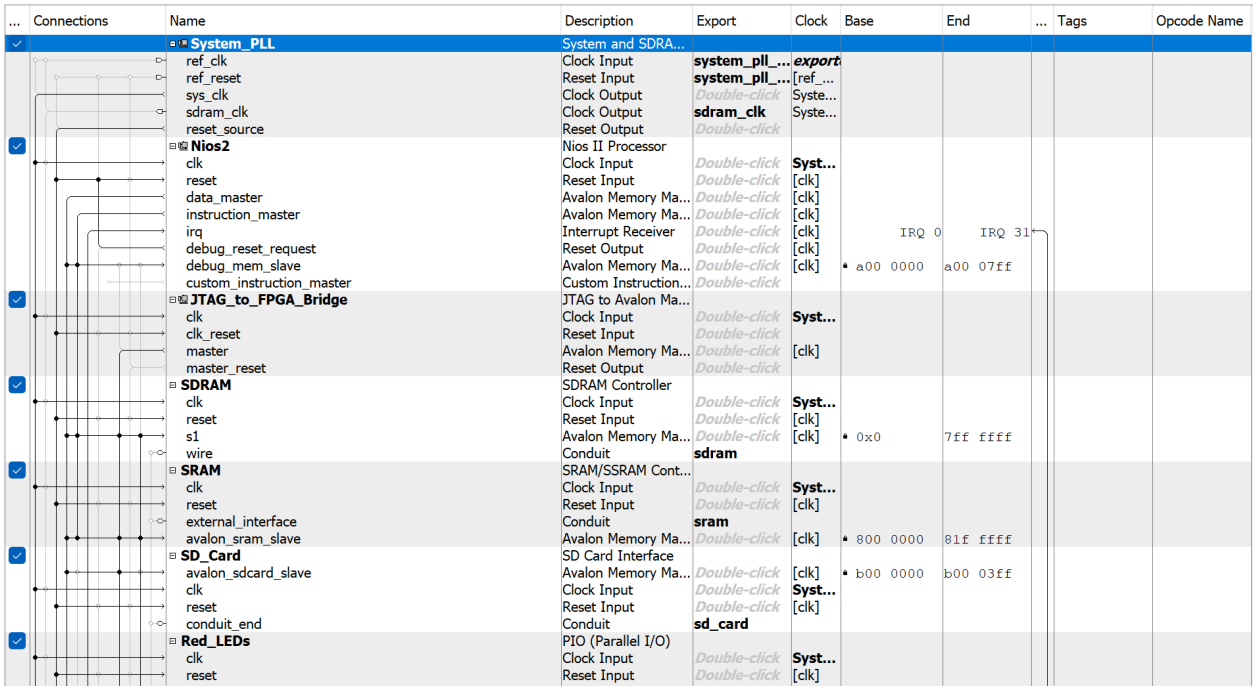

Figure 31: Connections to IP blocks.

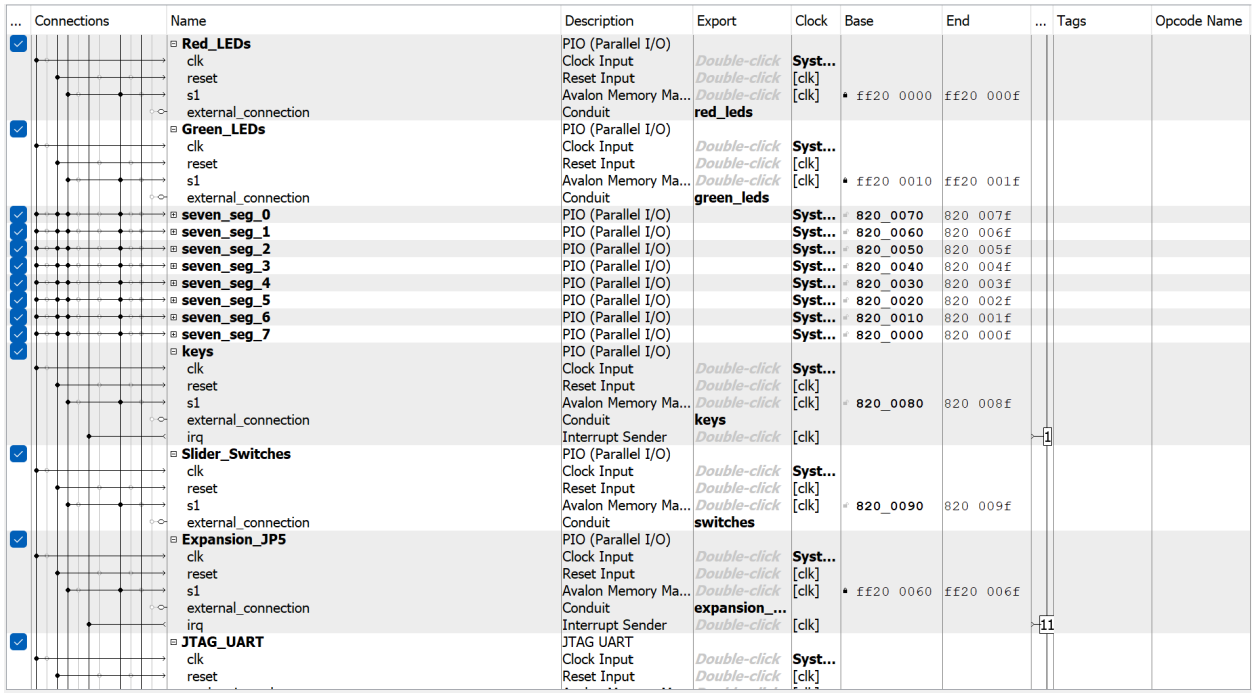

Figure 32: Connections to IP blocks.

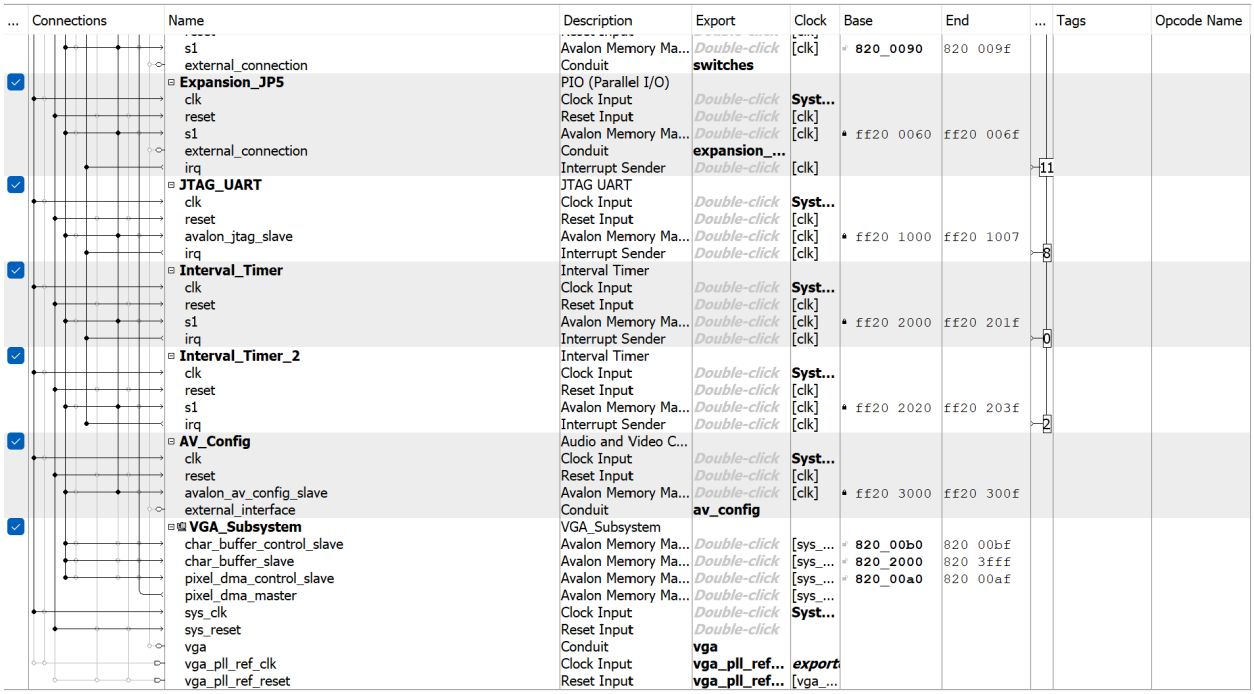

Figure 33: Connections to IP blocks.

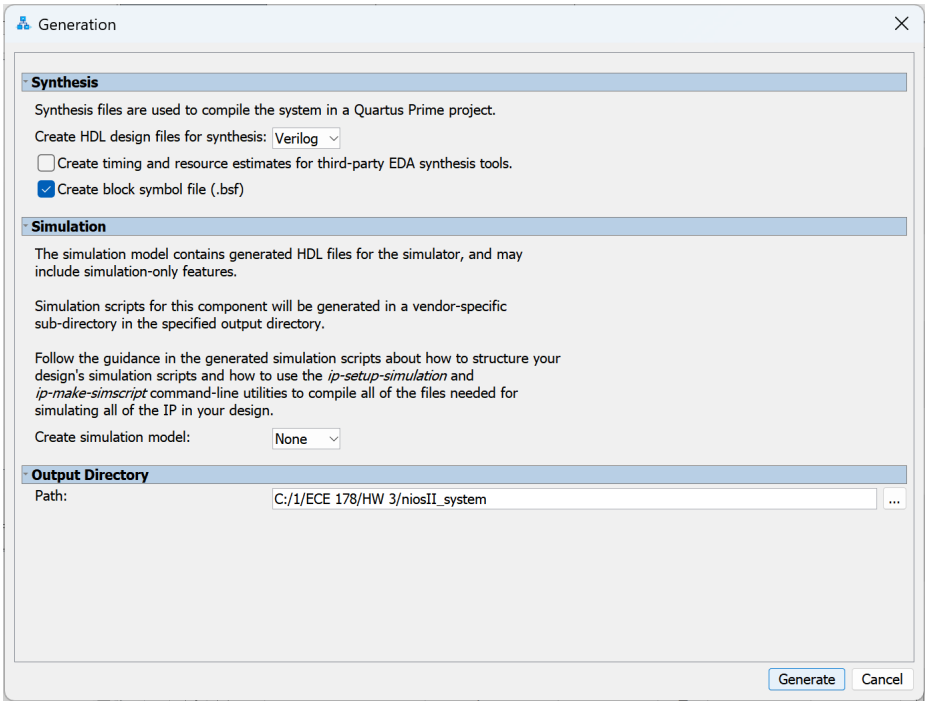

Figure 34: Generate/Synthesis window.

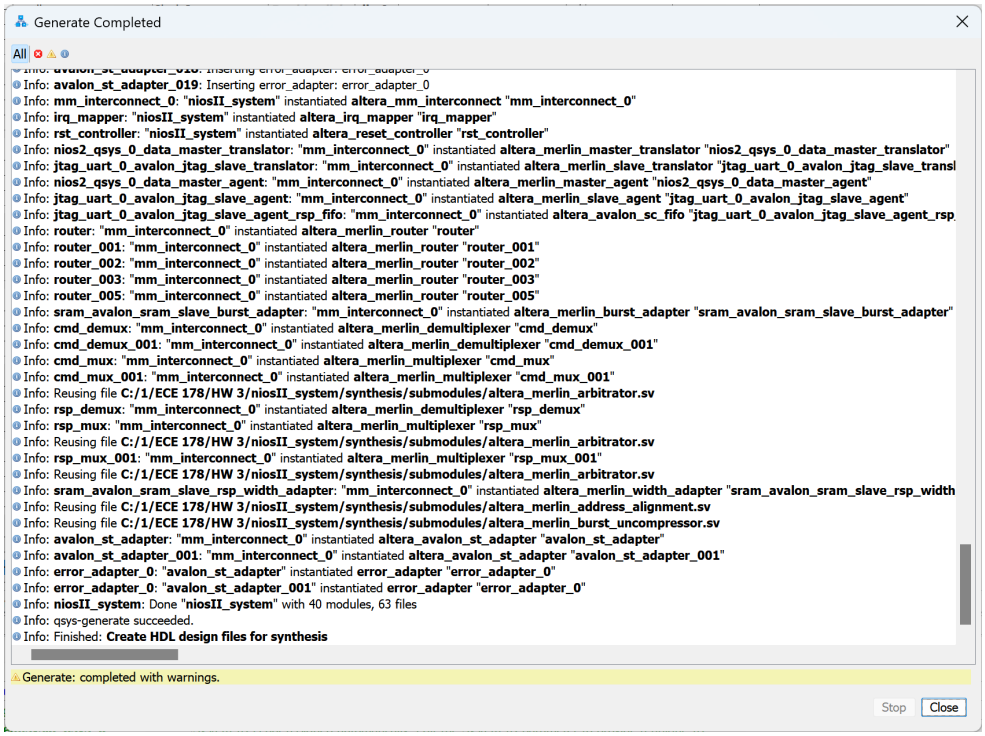

Figure 35: Generation completed window.

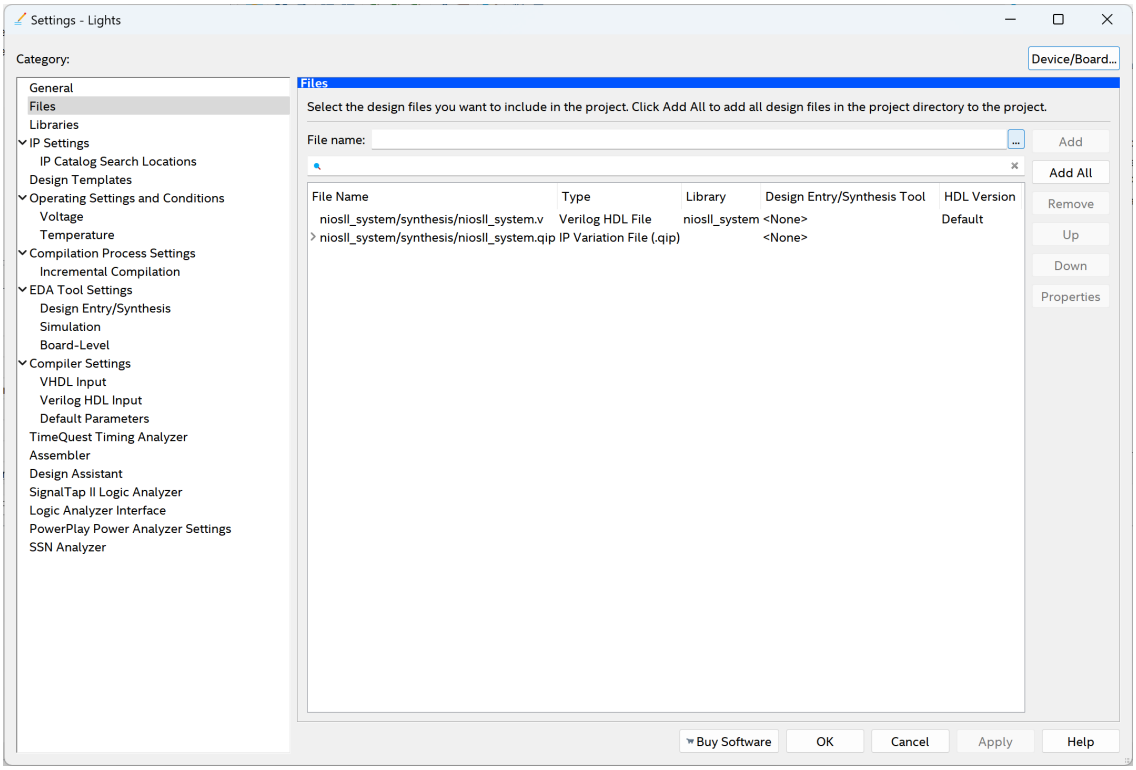

Figure 36: Adding files .v and .qip files.

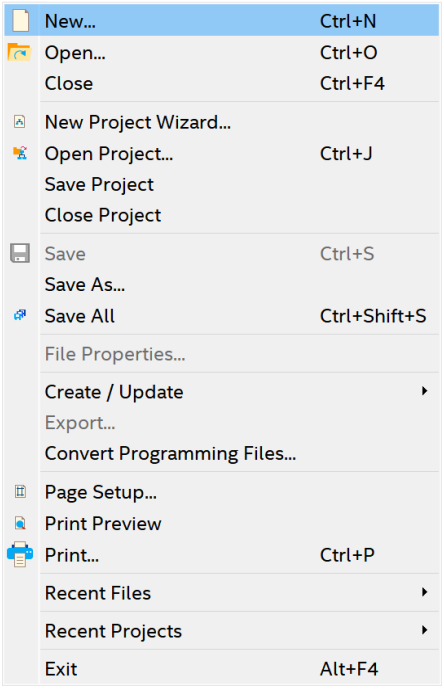

Figure 37: Create new file.

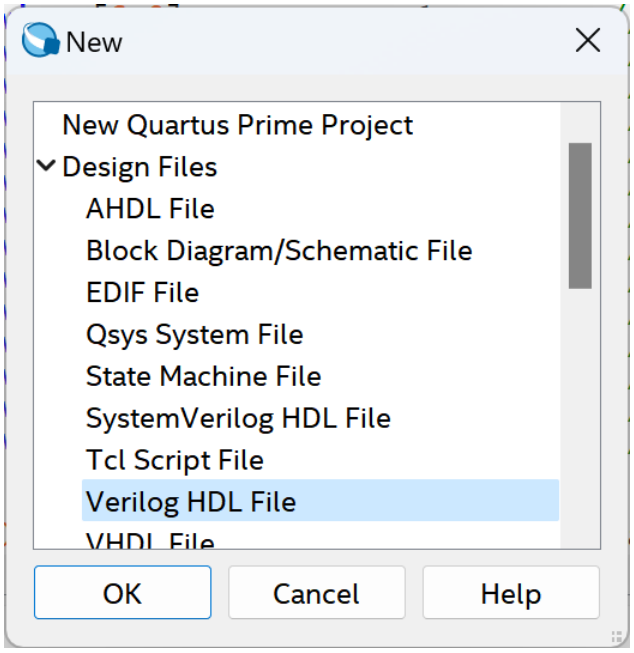

Figure 38: Add verilog file.

Create a workspace directory for where your project will be stored. This directory should be present project_name/software/workspace. Now open the “Nios II Embedded Development SBT for Eclipse” and link it to your workspace directory and click next. Figure 1 shows this step.

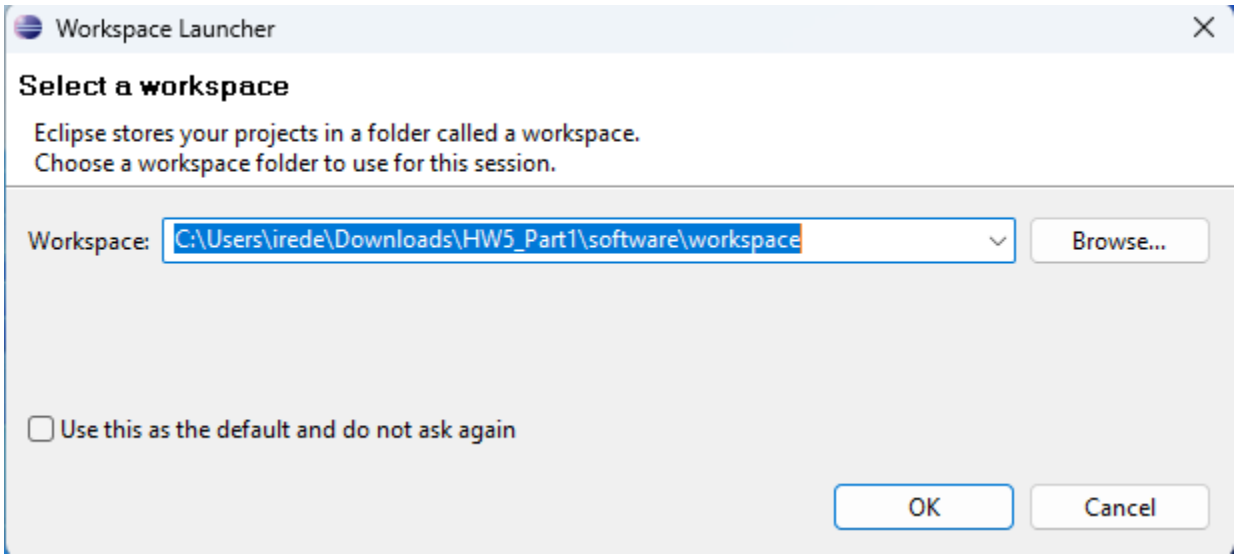

Figure 39: SBT workspace directory.

In the top right corner click File > New > Nios II Application and BSP from Template and click next.

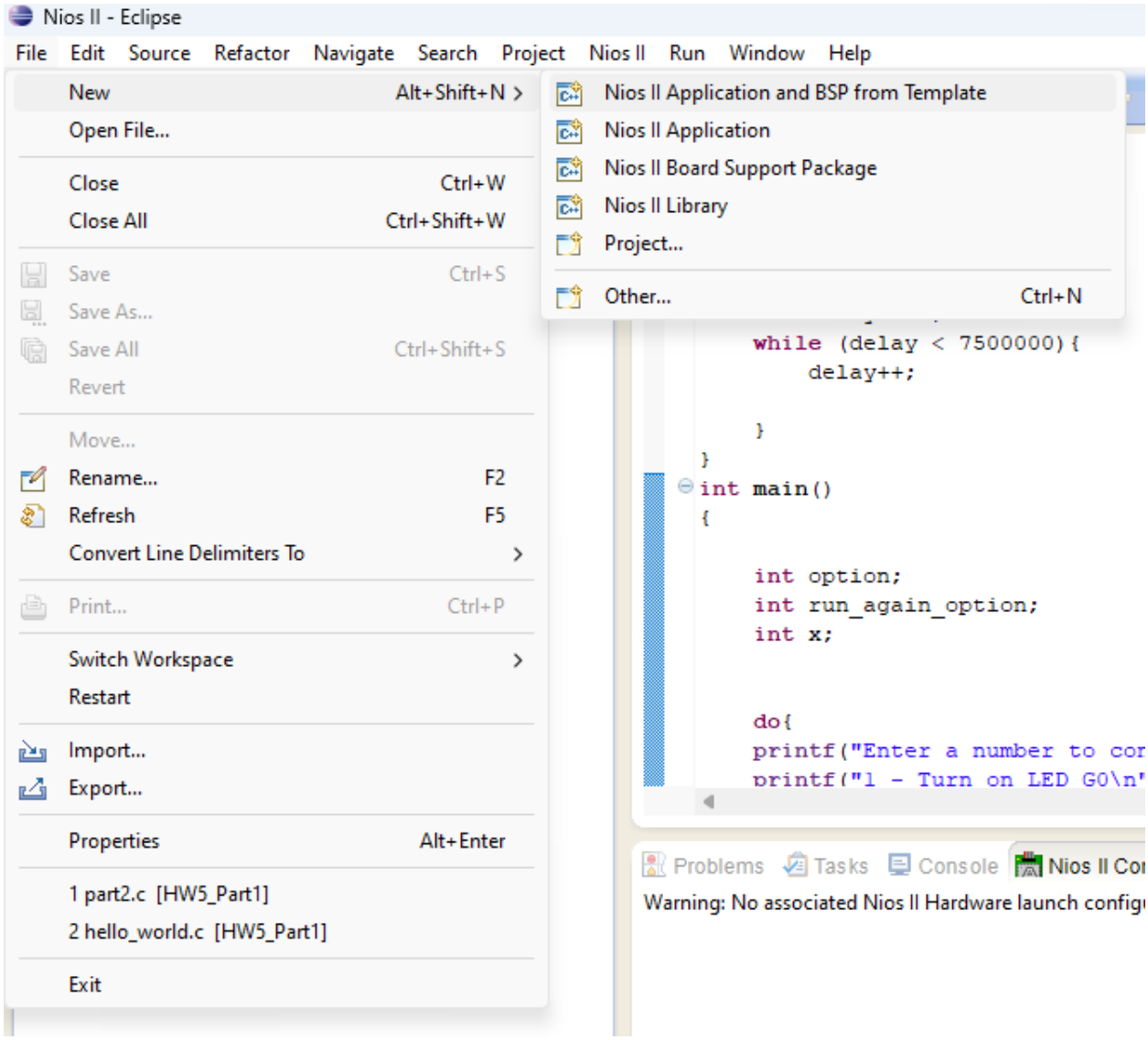

Figure 40: SBT project setup window.

In the Nios II Software Examples window, select your sopc file, and give the project a name. Under the project template, select the ‘Hello World’ template and click next.

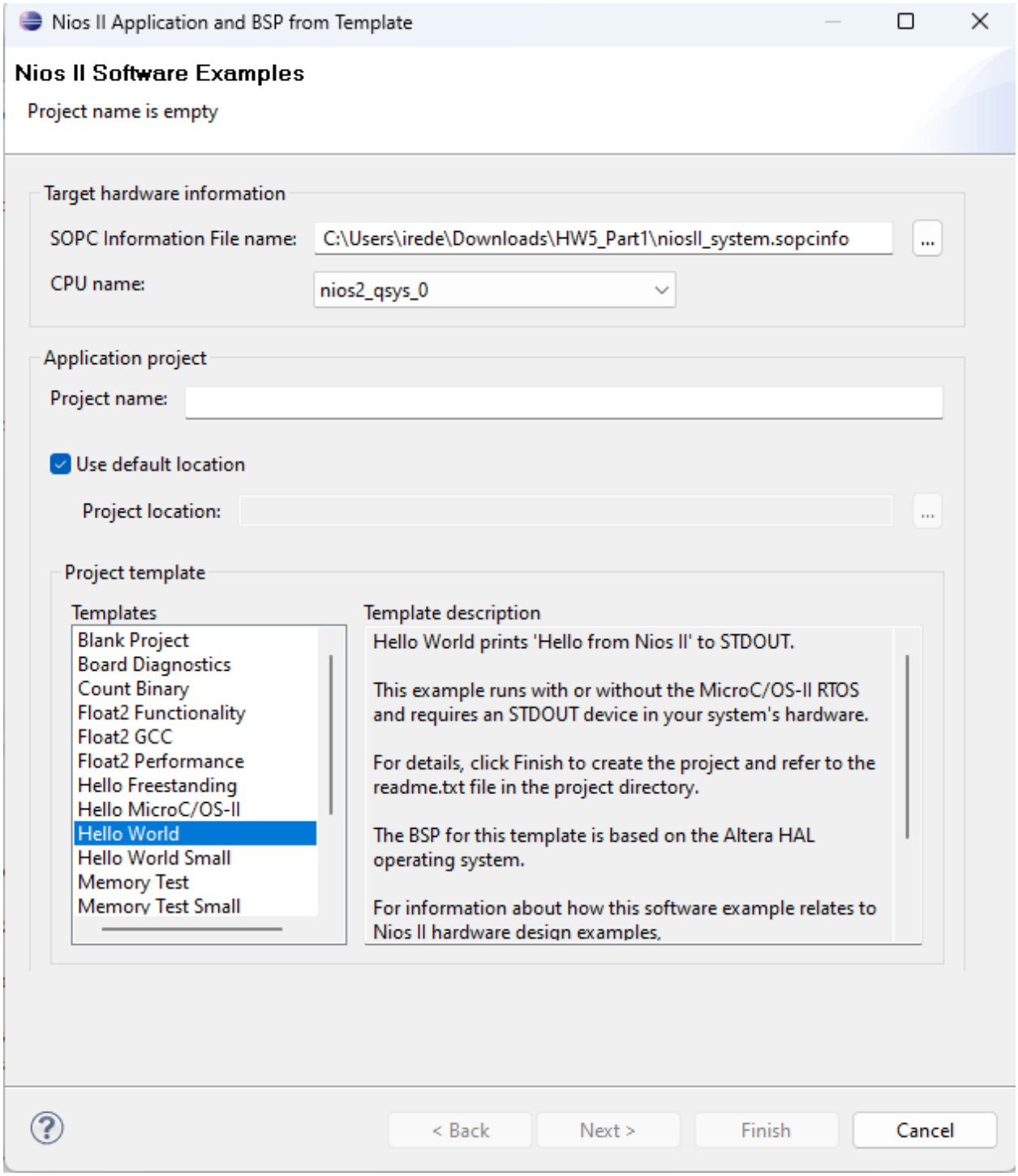

Figure 41: Nios II Software Examples window.

In the next window, select the ‘Create a new BSP project based on the application project template’ option and give it a project name. Click finish.

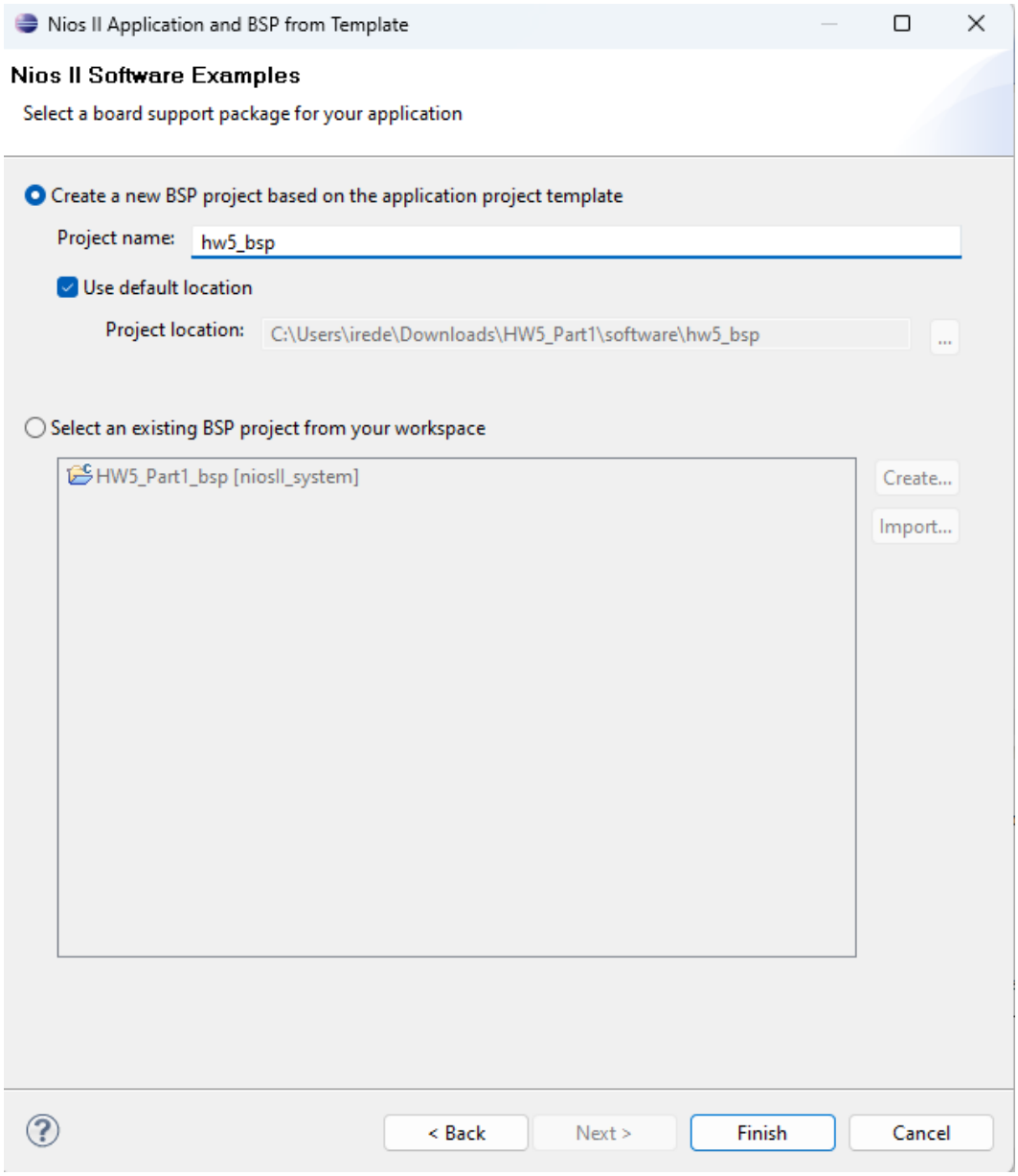

Figure 42: BSP project setup window.

Next right click on the project BSP folder and select the properties option, then under the 'Nios II BSP Properties' option click the BSP Editor button.

Under the Main window, select the *system timer* option under the sys_clk_timer dropdown.

Click the generate option and Exit button when the generation is complete.

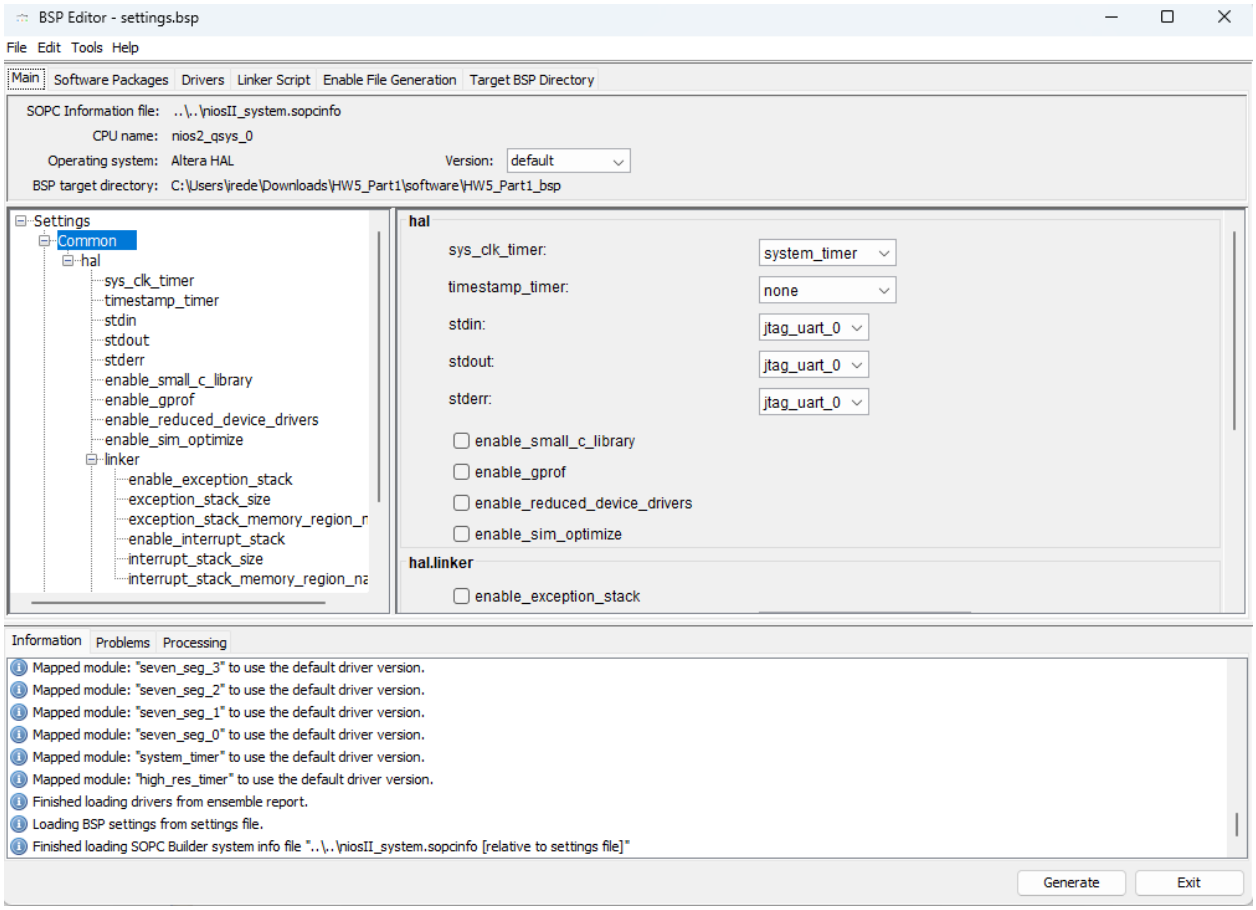

Figure 43: BSP Editor window.

Now open the Quartus II Programmer which is under the Nios II option in the top menu bar. Click the Add File button and include your sof file.

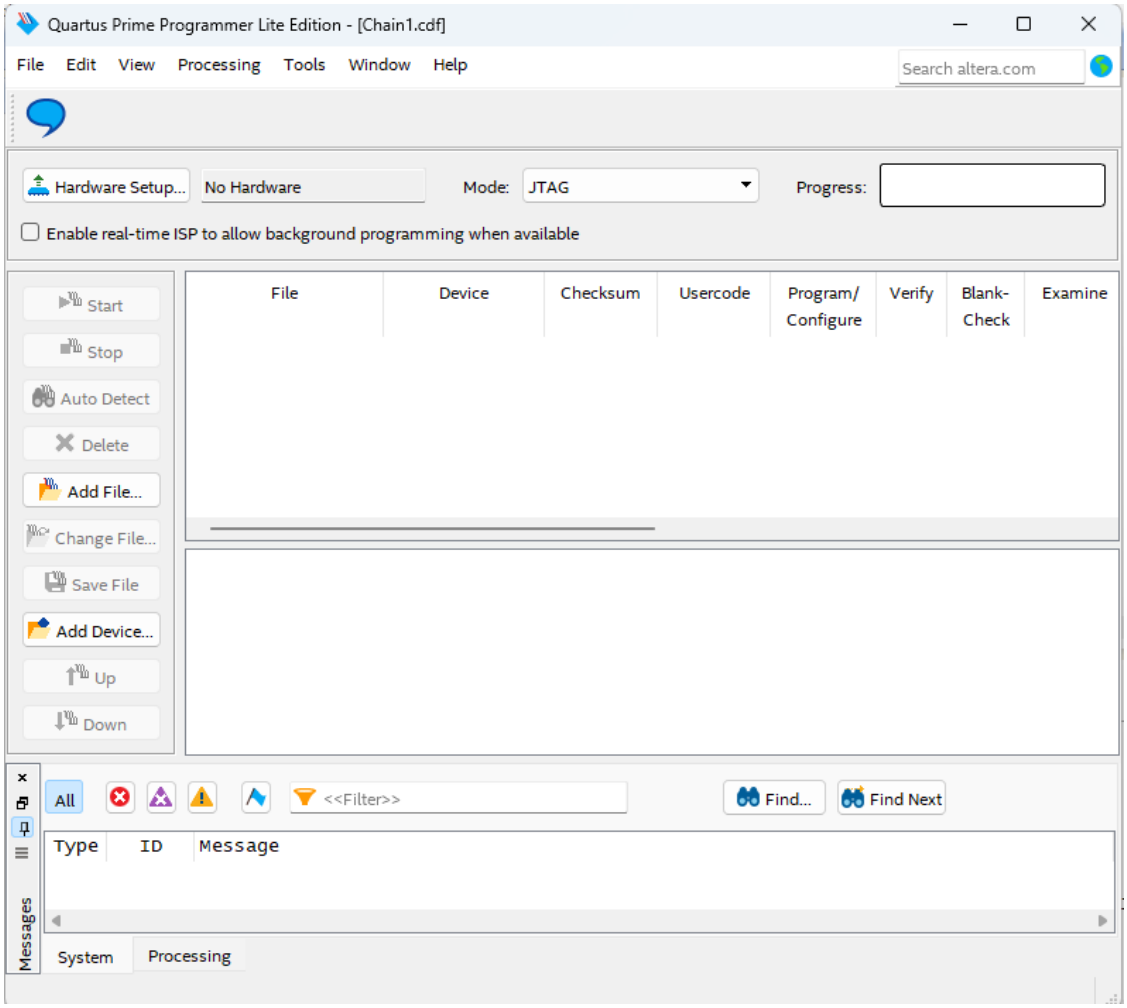

Figure 44: Quartus II Programmer window.

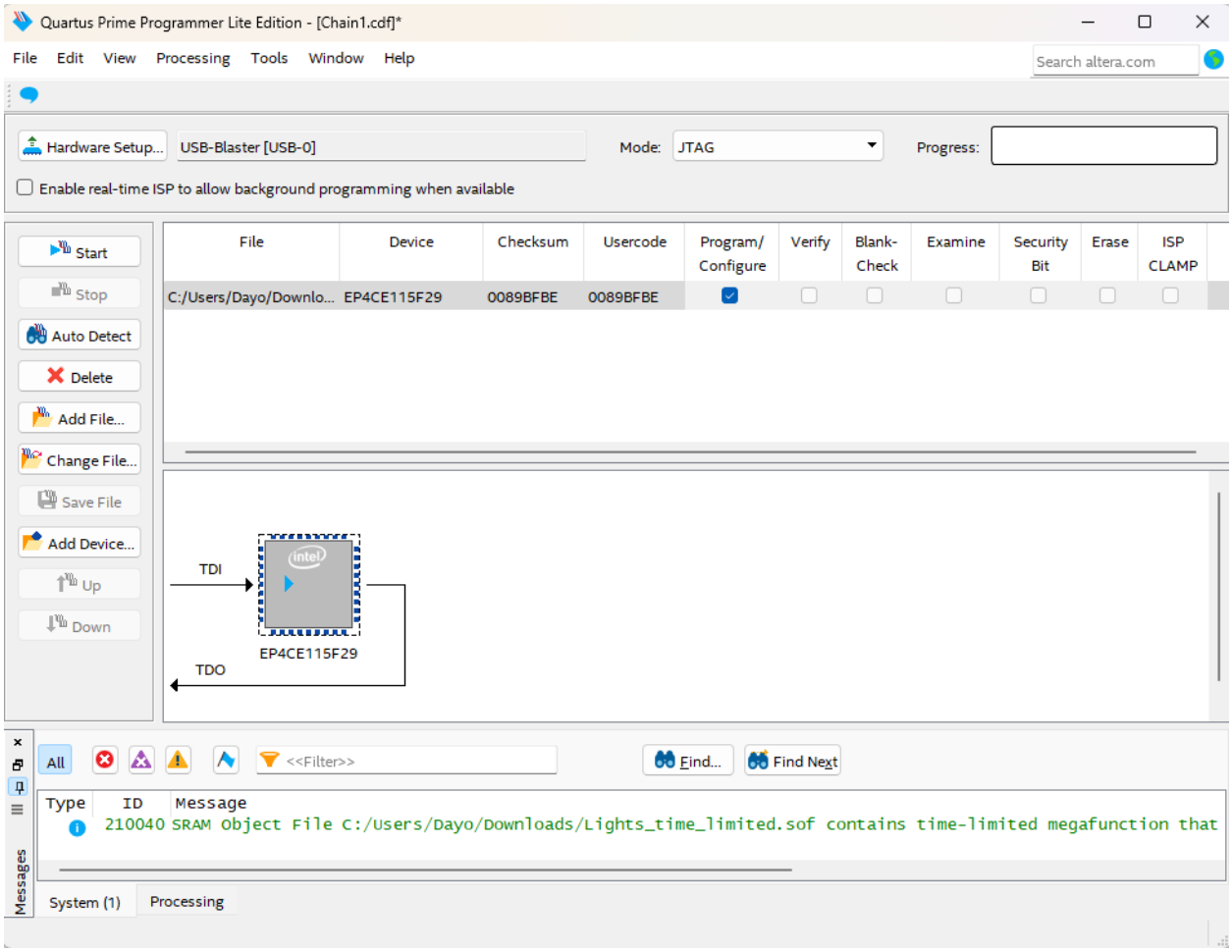

Figure 45: Quartus II Programmer window with loaded sof.

Make sure the DE2-115 board is powered on and connected to the computer then click the Hardware Setup button and select the USB-Blaster [USB-0] option.

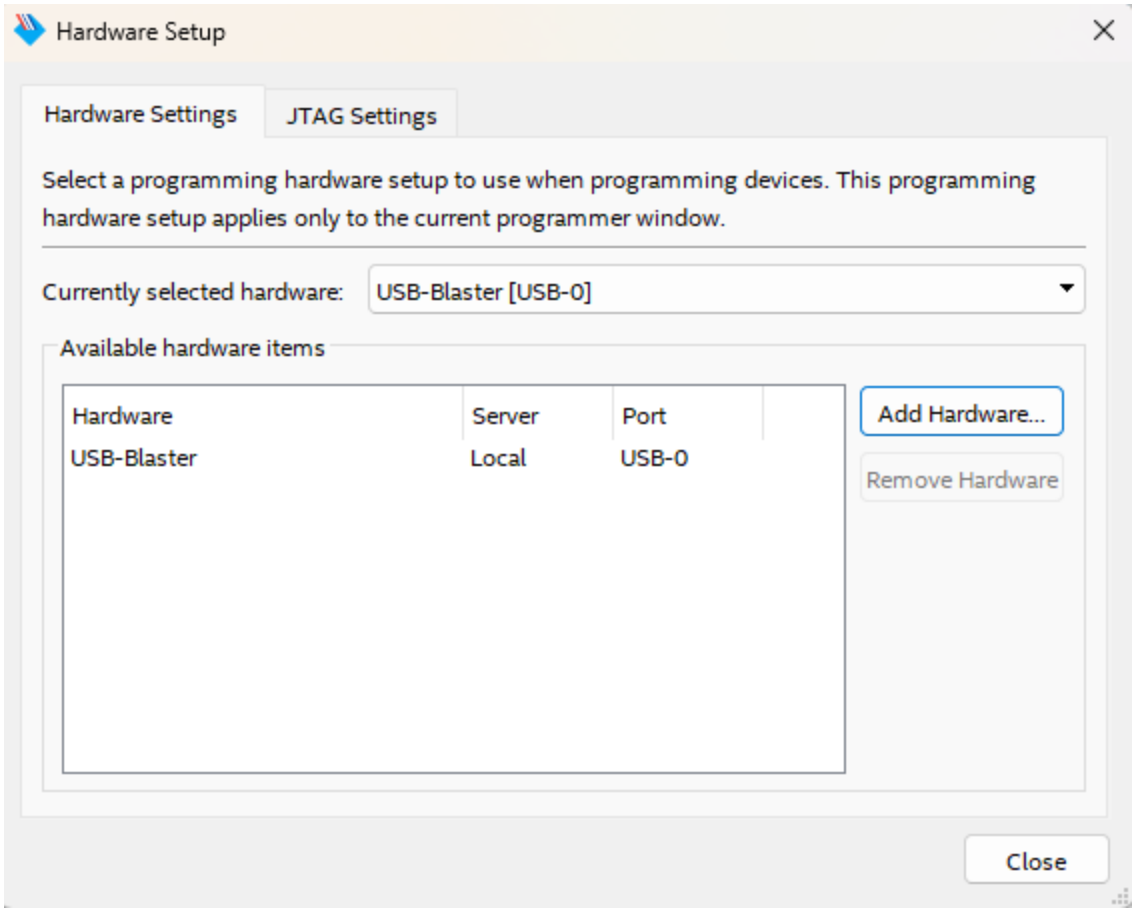

Figure 46: Hardware Setup window.

Now that everything is loaded up, click the Start button. The OpenCore Plus Status window will pop up, ignore it and switch back to eclipse. Do not click cancel.

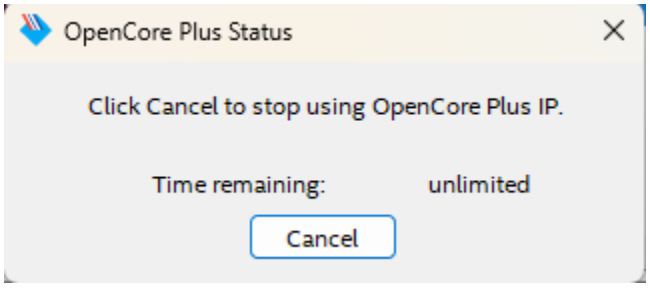

Figure 47: OpenCore Plus Status window.

Once back on Eclipse, right click on your project folder and select the Build Project option.

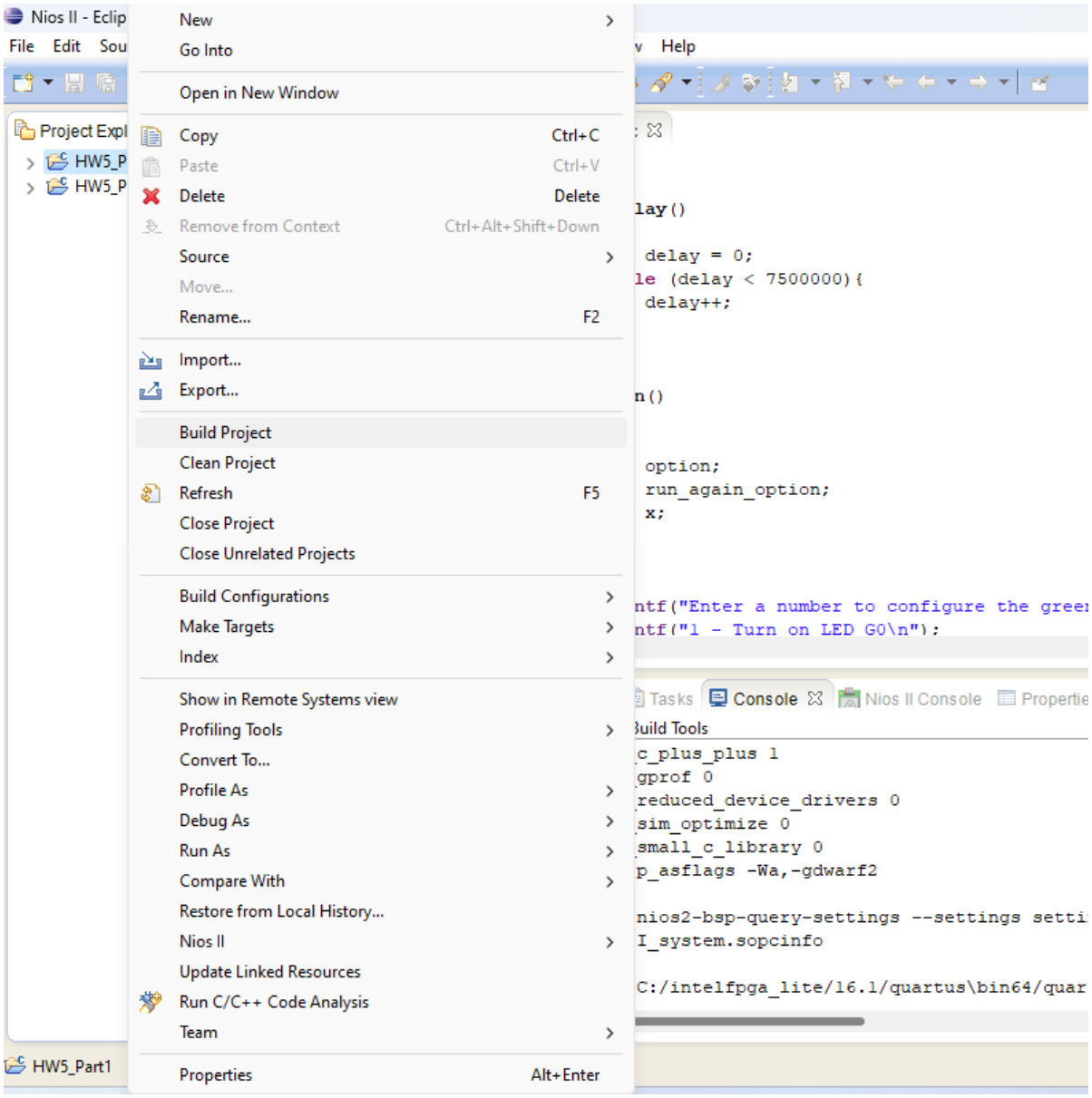

Figure 48: Build project.

Once the project has finished building, right click on the project again and select Run As > Nios II Hardware.

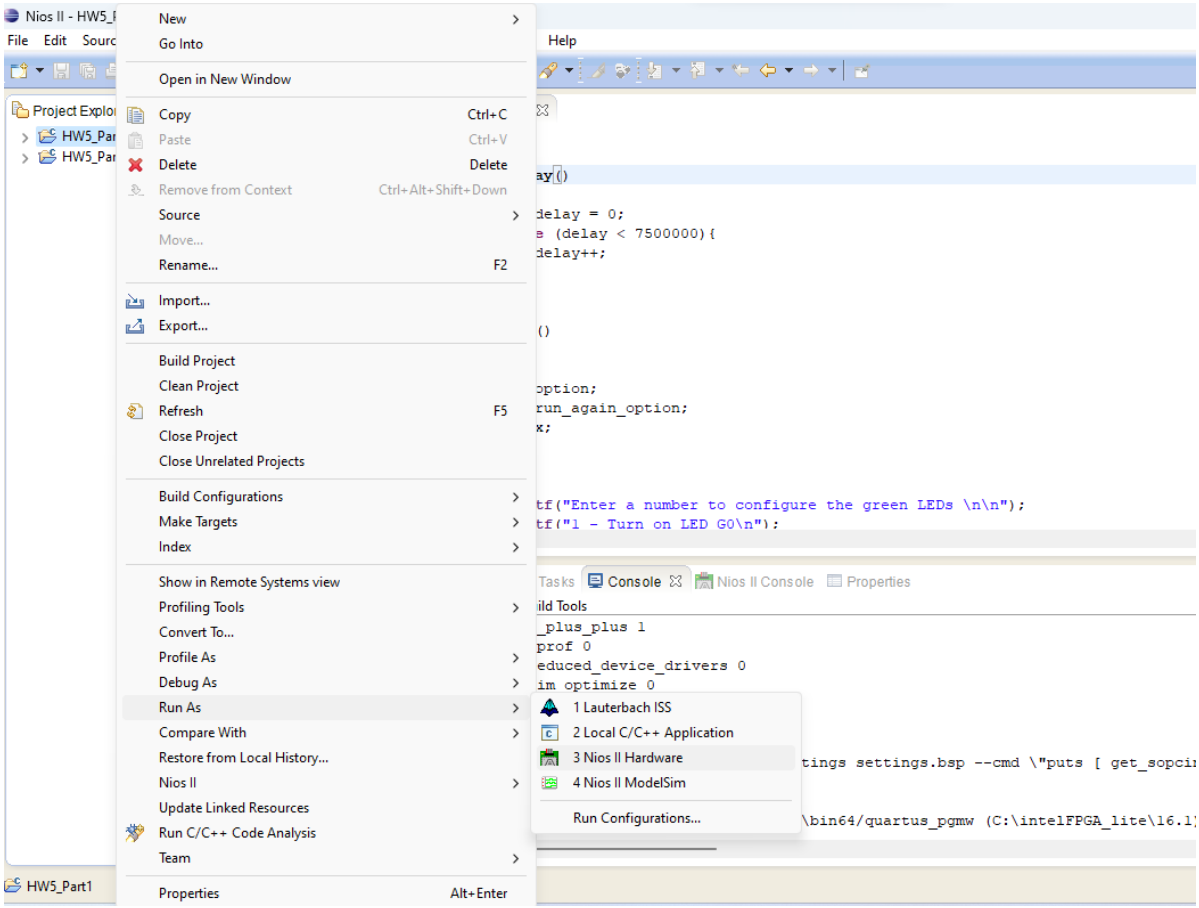

Figure 49: Running the project.

The Run Configuration window will pop up, select the Target Connection window. Now under System ID checks select the two options, “Ignore mismatched system ID” and “Ignore mismatched system timestamp”. Then click the Refresh Connection button and press Apply. Click Run and the program should run with no errors.

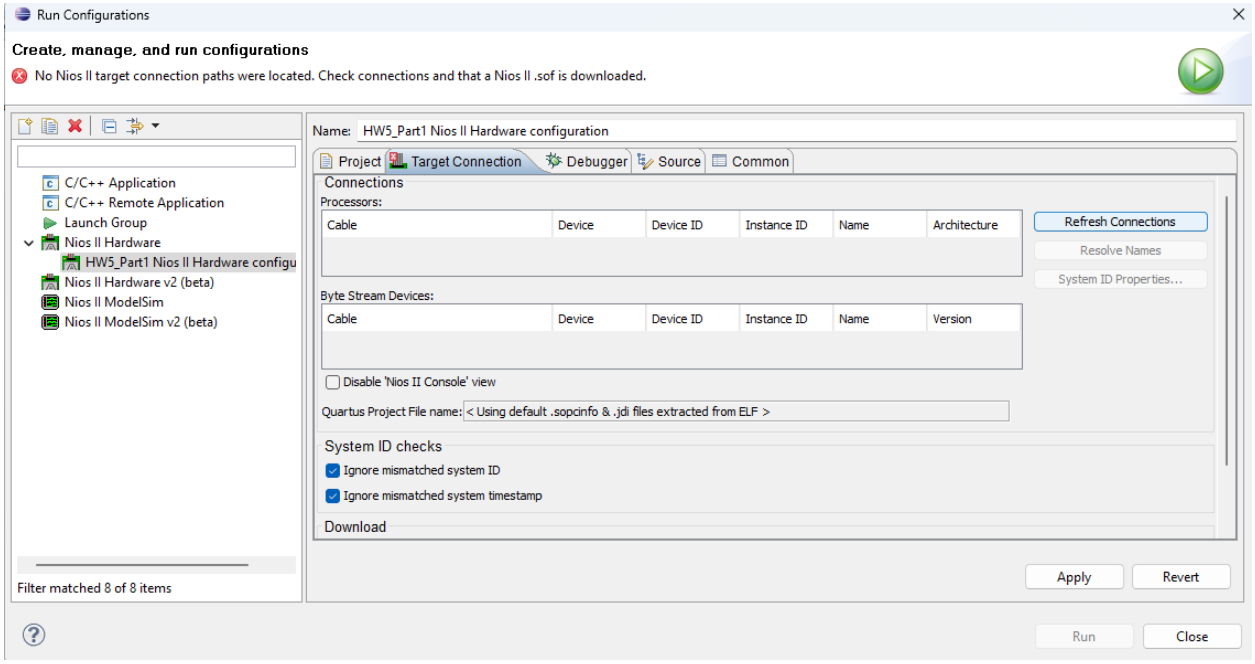

Figure 50: Run Configuration window.

6. Analysis

The project demonstration has three parts that demonstrate the different functionality that was developed. The first program demonstrates RTOS, the second program involves reading/writing to the SD card and the last demonstration displays image data from an SD card to a monitor using the VGA output.

RTOS program:
The first demonstration involves using RTOS to schedule two tasks with different priority levels. The first task is responsible for lighting up the red LEDs and printing to the console “Task 1 executes”. Task 2 is responsible for lighting up, clearing the red LEDs and lighting up the green LEDs for three seconds while also printing “Task 2 executes” to the console. In terms of priority, task 1 is given a higher priority than task 2.

Both tasks execute at the same time, when the program runs the red LEDs light up first because it has a higher priority than task 2, and after task 1 is done task 2 runs which is responsible for clearing the red LEDs and lighting up the green LEDs. The program halts

for 3 seconds and the process repeats itself.

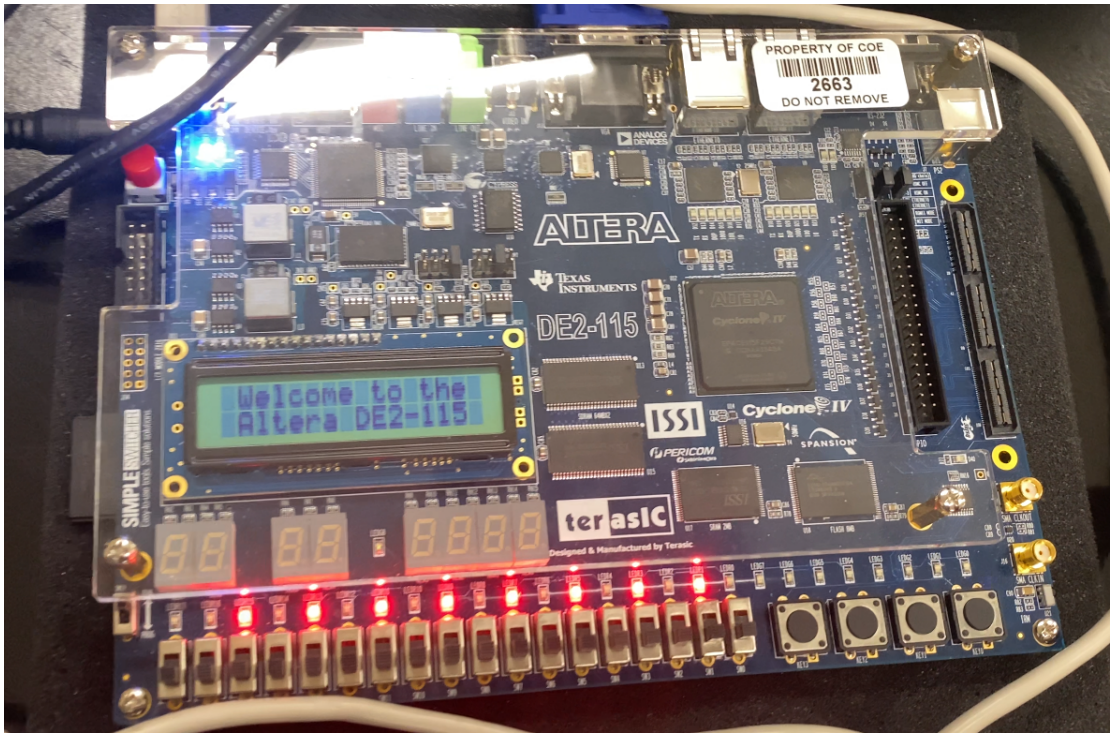

Figure 51: Task 1 is running.

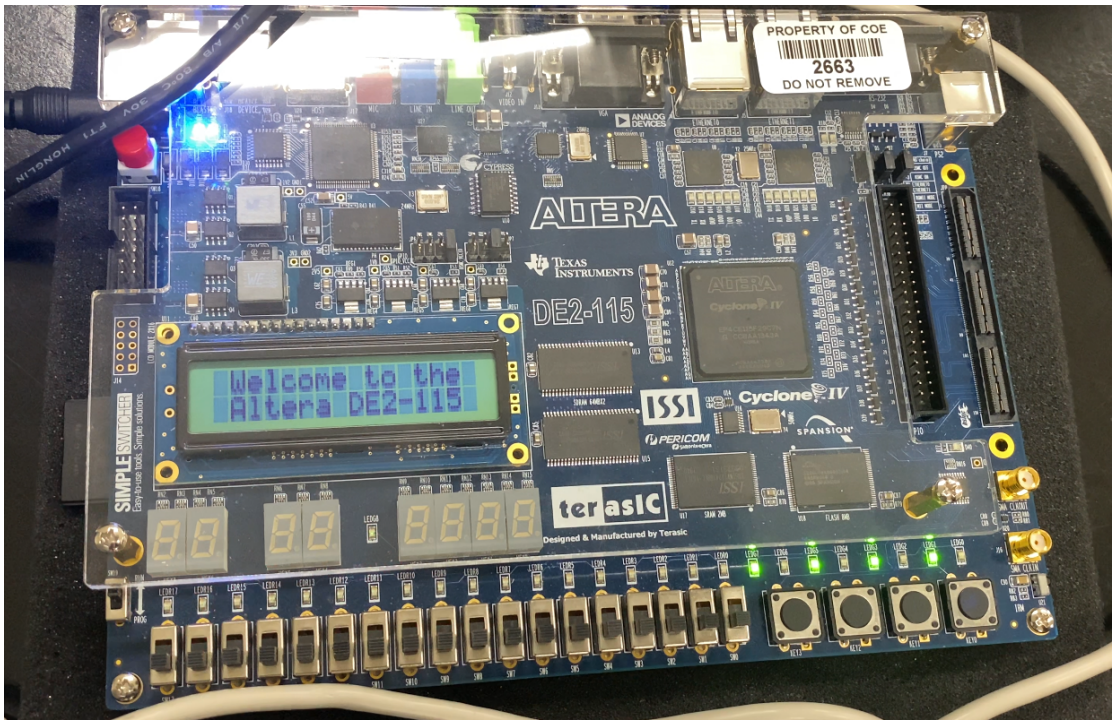

Figure 52: Task 2 running.

SD card:
The second program demonstrates the use of the SD card, reading and writing to the SD card file system. Using the “Altera_UP_SD_Card_Avalon_Interface.h” library we were able to interface with the SD card.

When the program runs it gives the user a chance to select one of three choices using the keyboard. Entering a ‘1’ will display a list of all the files saved onto the SD card, ‘2’ gives the option to write a predetermined message to a predetermined file on the SD card, and ‘3’ gives the user the option to read a file from the SD card.

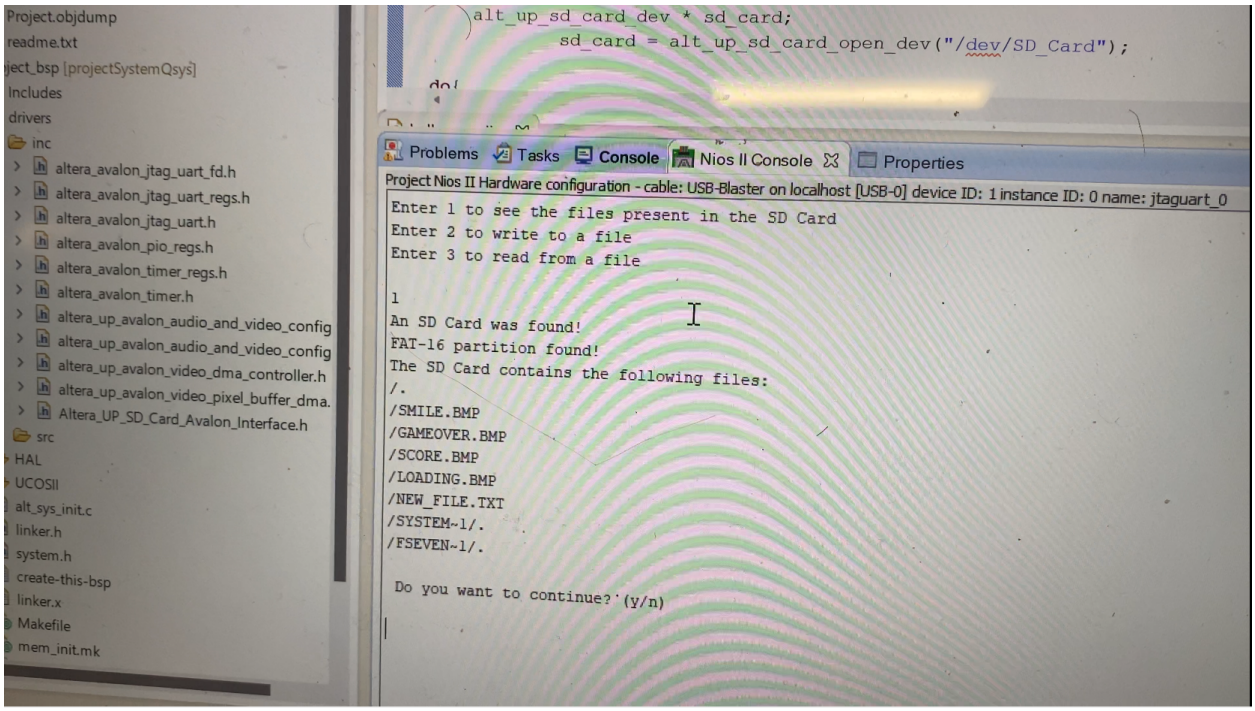

Figure 53: Option 1 displaying the already existing files on the SD card.

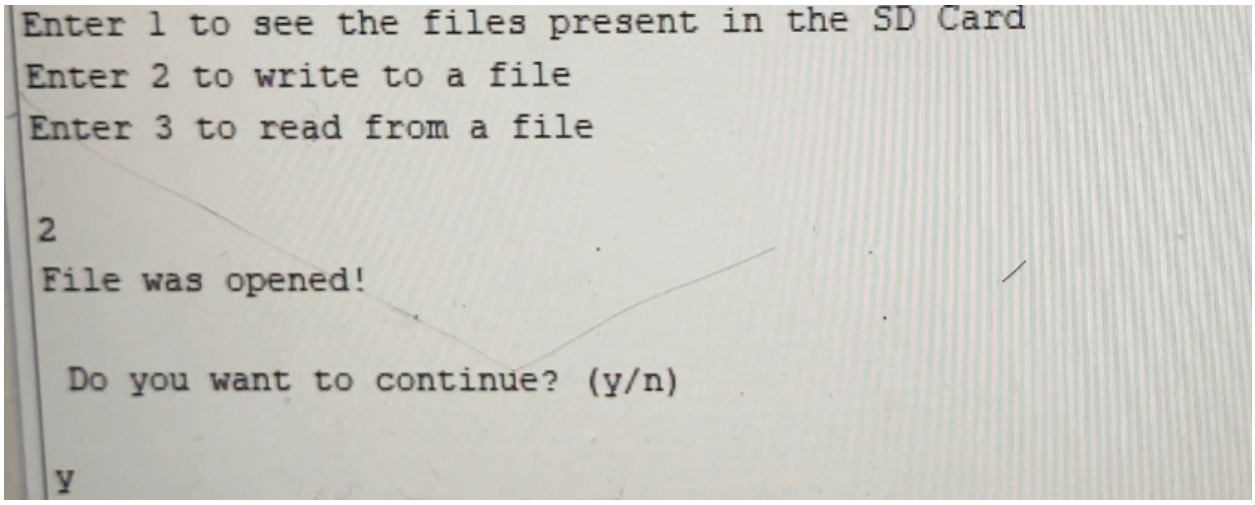

Figure 54: Option 2 writing to a file on the SD card.

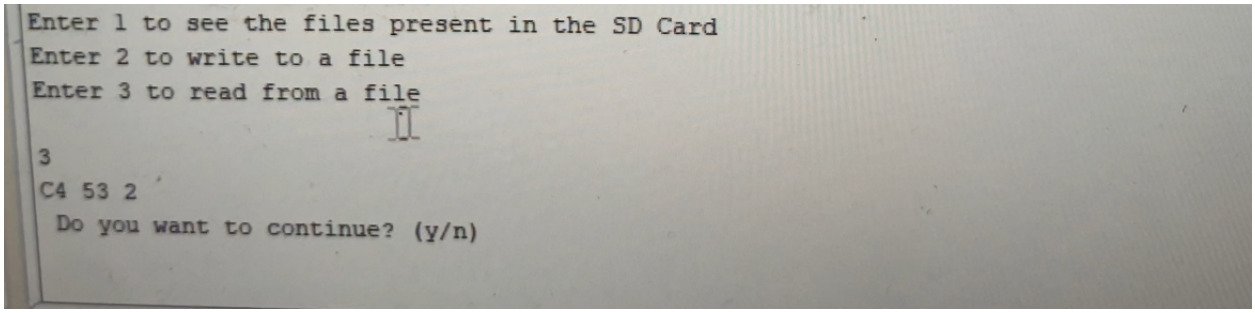

Figure 55: Option 3 reading from a file on the SD card.

When option 1 runs it displays seven file directories with their file name and extension. Most of the files have the extension .BMP because these are bitmap image files that will be used for the final program demonstration that involves the VGA.

Option 2 writes a predetermined message which is the character “C” to the predetermined file named “NEW_FILE.txt”.

Option 3 opens the predetermined file “NEW_FILE.txt” and displays the contents of the file to the console. The resulting display shows the recently added “C” character added to the file amongst the previously stored message.

VGA display:
This final program builds upon the previous one since it will be reading image files from an SD card to display to the monitor. We were able to access the pixel buffer and the DMA controller using the “altera_up_avalon_video_pixel_buffer_dma.h” and the “altera_up_avalon_video_dma_controller.h” library respectively.

The first function in the program is responsible for drawing characters to the screen that arrange in a formation to spell “ECE 178”. Using the built in *alt_up_pixel_buffer_dma_draw_box()* function, each letter in the word has its own separate block with its necessary pixel information.

The next function displayVGA() is responsible for reading all four .BMP files on the SD card and displaying it to the monitor. This function uses the same ones from the previous program to read and write to the SD card. The final screen to be displayed will be a blank screen to indicate the end.

Figure 56: ECE 178 drawn to the screen.

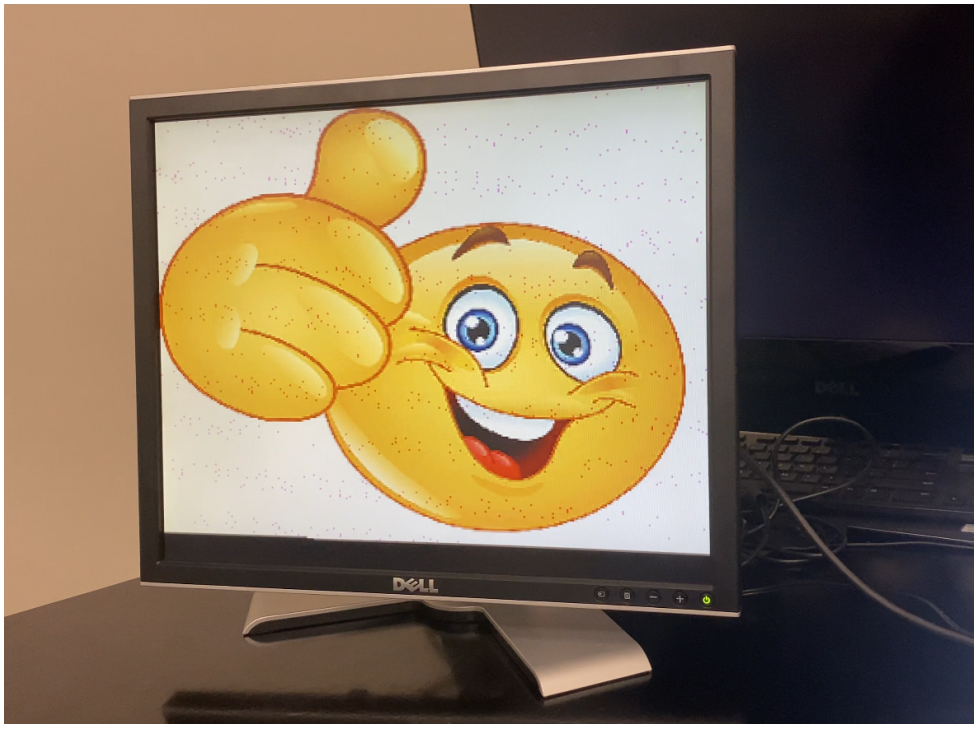

Figure 57: SMILE.BMP screen.

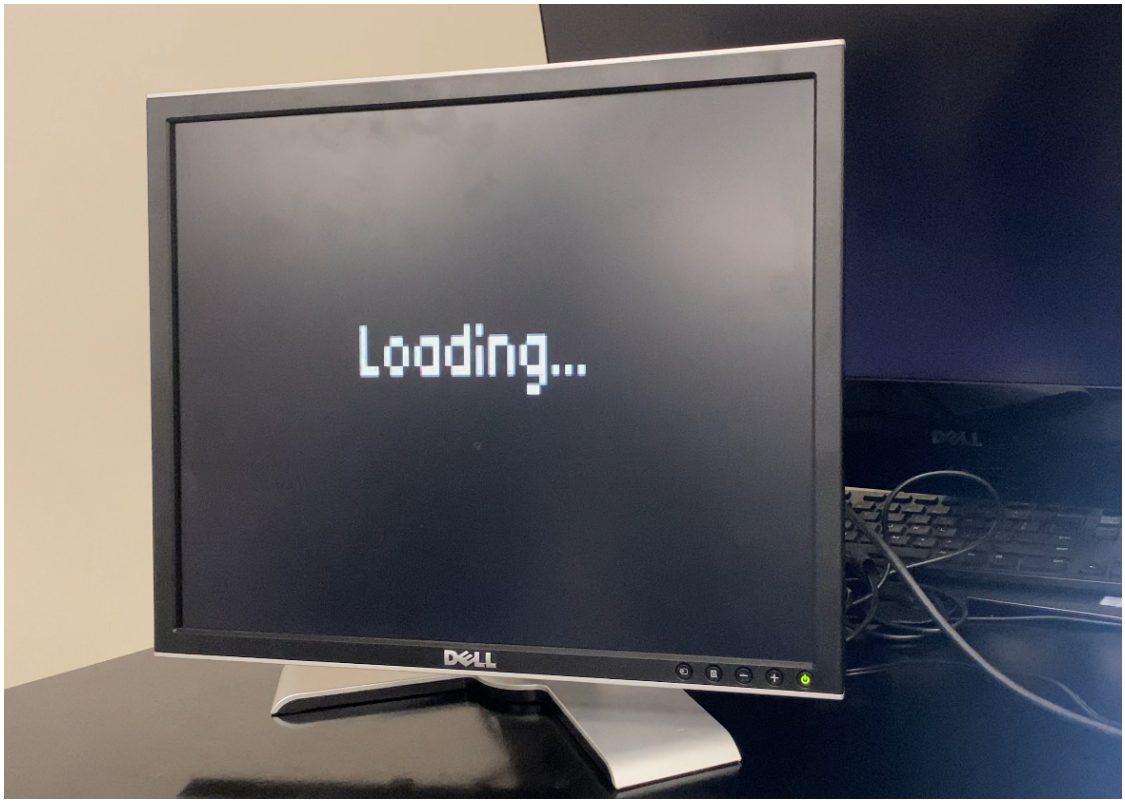

Figure 58: LOADING.BMP screen.

Figure 59: GAME_OVER.BMP screen.

Figure 60: SCORE.BMP screen.

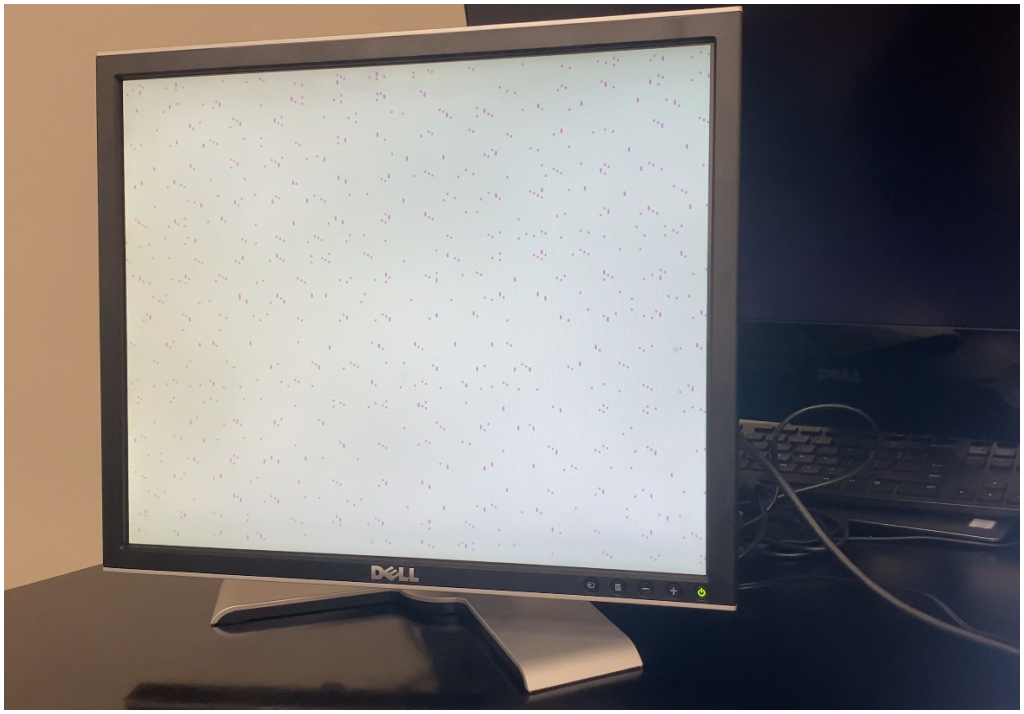

Figure 61: Blank screen.

7. Conclusion

In conclusion, this final project was used to test students' ability to develop useful software by integrating different aspects of what was taught in class. Also exposing students to new components and learning how to integrate it to the projects. For this project we were able to successfully implement RTOS, SD card interface and VGA control to display images to a monitor.

Appendix A

RTOS c program

```
#include <stdio.h>
#include <unistd.h>
#include <io.h>
#include <system.h>
#include <math.h>
#include "altera_avalon_pio_regs.h"
#include <stdio.h>
#include "includes.h"
#include "Altera_UP_SD_Card_Avalon_Interface.h"


/* Definition of Task Stacks */
#define   TASK_STACKSIZE       2048
OS_STK    task1_stk[TASK_STACKSIZE];
OS_STK    task2_stk[TASK_STACKSIZE];
OS_STK    task3_stk[TASK_STACKSIZE];
/* Definition of Task Priorities */

#define TASK1_PRIORITY      1
#define TASK2_PRIORITY      2


#define MAX_SUBDIRECTORIES 20


void clear_all_leds();
void clear_hex();
void find_files (char* path);
void delay(int d);

int switches_in;
int tens_hex;
int ones_hex;
/* Prints "Hello World" and sleeps for three seconds */
void task1(void* pdata)
{
  while (1)
  {
          clear_all_leds();
          clear_hex();
          IOWR_ALTERA_AVALON_PIO_DATA(RED_LEDS_BASE, 0xaaaa);
          printf("Task 1 executes\n");
          delay(500000);

          OSTimeDlyHMSM(0, 0, 3, 0);


  }
}


/* Prints "Hello World" and sleeps for three seconds */
void task2(void* pdata)
```

```
{
  while (1)
  {
          clear_all_leds();
          clear_hex();

          IOWR_ALTERA_AVALON_PIO_DATA(GREEN_LEDS_BASE, 0b10101010);
          printf("Task 2 executes\n");
          delay(500000);
          OSTimeDlyHMSM(0, 0,3 , 0);

  }
}
/* The main function creates two task and starts multi-tasking */
int main(void)
{

  OSTaskCreateExt(task1,
                  NULL,
                  (void *)&task1_stk[TASK_STACKSIZE-1],
                  TASK1_PRIORITY,
                  TASK1_PRIORITY,
                  task1_stk,
                  TASK_STACKSIZE,
                  NULL,
                  0);

  OSTaskCreateExt(task2,
                  NULL,
                  (void *)&task2_stk[TASK_STACKSIZE-1],
                  TASK2_PRIORITY,
                  TASK2_PRIORITY,
                  task2_stk,
                  TASK_STACKSIZE,
                  NULL,
                  0);


  OSStart();
  return 0;
}


void clear_hex()
{
        IOWR_ALTERA_AVALON_PIO_DATA(SEVEN_SEG_0_BASE, 0xFF);
        IOWR_ALTERA_AVALON_PIO_DATA(SEVEN_SEG_1_BASE, 0xFF);
        IOWR_ALTERA_AVALON_PIO_DATA(SEVEN_SEG_2_BASE, 0xFF);
        IOWR_ALTERA_AVALON_PIO_DATA(SEVEN_SEG_3_BASE, 0xFF);
        IOWR_ALTERA_AVALON_PIO_DATA(SEVEN_SEG_4_BASE, 0xFF);
        IOWR_ALTERA_AVALON_PIO_DATA(SEVEN_SEG_5_BASE, 0xFF);
        IOWR_ALTERA_AVALON_PIO_DATA(SEVEN_SEG_6_BASE, 0xFF);
        IOWR_ALTERA_AVALON_PIO_DATA(SEVEN_SEG_7_BASE, 0xFF);
}

void clear_all_leds()
{
        IOWR_ALTERA_AVALON_PIO_DATA(RED_LEDS_BASE, 0);
        IOWR_ALTERA_AVALON_PIO_DATA(GREEN_LEDS_BASE, 0);
```

```
}

void delay(int d)
{
        while (d > 0){

                d = d -1;
        }
}
```

SD card c program

```
#include <io.h>
#include "system.h"
#include "altera_avalon_timer_regs.h"
#include "altera_avalon_timer.h"
#include "altera_avalon_pio_regs.h"
#include "sys/alt_irq.h"
#include "sys/alt_stdio.h"
#include "alt_types.h"
#include <stdio.h>
#include "includes.h"
#include <stdlib.h>
#include <string.h>
#include <stdbool.h>
#include "Altera_UP_SD_Card_Avalon_Interface.h"
#include "altera_up_avalon_video_dma_controller.h"
#include "altera_up_avalon_video_pixel_buffer_dma.h"
#include "sys/alt_stdio.h"

#define MAX_SUBDIRECTORIES 20

void displayVGA(char fn[]);
void VGA_box (int,int,int,int,int);


void delay(int ms){

        int i;
        for (i=ms;i>0;i--)
        {}

}

void draw_ECE_178(alt_up_pixel_buffer_dma_dev *pixel_buffer_dev )
{       delay(500000);
        int background_color = 0x001F;  // Blue in a 16-bit color format
         alt_up_pixel_buffer_dma_draw_box(pixel_buffer_dev, 0, 0, 639, 479, background_color, 0);

        alt_up_pixel_buffer_dma_draw_box(pixel_buffer_dev, 10, 10, 20, 88, 0xffff, 0x001F);
            alt_up_pixel_buffer_dma_draw_box(pixel_buffer_dev, 10, 10, 60, 20, 0xffff, 0x001F);
            alt_up_pixel_buffer_dma_draw_box(pixel_buffer_dev, 10, 50, 50, 60, 0xffff, 0x001F);
            alt_up_pixel_buffer_dma_draw_box(pixel_buffer_dev, 10, 80, 60, 88, 0xffff, 0x001F);

            // Draw "C"
            alt_up_pixel_buffer_dma_draw_box(pixel_buffer_dev, 70, 10, 80, 88, 0xffff, 0x001F);
```

```
    alt_up_pixel_buffer_dma_draw_box(pixel_buffer_dev, 80, 10, 100, 20, 0xffff, 0x001F);
            alt_up_pixel_buffer_dma_draw_box(pixel_buffer_dev, 80, 80, 100, 88, 0xffff, 0x001F);

            // Draw "E"
           alt_up_pixel_buffer_dma_draw_box(pixel_buffer_dev, 110, 10, 120, 88, 0xffff, 0x001F);
            alt_up_pixel_buffer_dma_draw_box(pixel_buffer_dev, 110, 10, 160, 20, 0xffff, 0x001F);
            alt_up_pixel_buffer_dma_draw_box(pixel_buffer_dev, 110, 50, 150, 60, 0xffff, 0x001F);
            alt_up_pixel_buffer_dma_draw_box(pixel_buffer_dev, 110, 80, 160, 88, 0xffff, 0x001F);

            // Draw "178"
            alt_up_pixel_buffer_dma_draw_box(pixel_buffer_dev, 170, 110, 180, 188, 0xffff, 0x001F);  //
1



            alt_up_pixel_buffer_dma_draw_box(pixel_buffer_dev, 210, 110, 220, 188, 0xffff, 0x001F);  //
7
            alt_up_pixel_buffer_dma_draw_box(pixel_buffer_dev, 190, 110, 220, 120, 0xffff, 0x001F);

            alt_up_pixel_buffer_dma_draw_box(pixel_buffer_dev, 230, 110, 270, 120, 0xffff, 0x001F);  //
8
            alt_up_pixel_buffer_dma_draw_box(pixel_buffer_dev, 230, 110, 240, 188, 0xffff, 0x001F);
            alt_up_pixel_buffer_dma_draw_box(pixel_buffer_dev, 260, 110, 270, 188, 0xffff, 0x001F);
            alt_up_pixel_buffer_dma_draw_box(pixel_buffer_dev, 230, 145, 270, 155, 0xffff, 0x001F);
            alt_up_pixel_buffer_dma_draw_box(pixel_buffer_dev, 230, 180, 270, 188, 0xffff, 0x001F);


}

void VGA_box (int x1, int y1, int x2, int y2, int pixel_color){
int offset, row, column;
volatile short * pixel_buffer = (short *) SRAM_BASE;//VGA pixel buffer address
for (row = y1-1; row <= y2-1; row++)
{
column = x1;
while (column <= x2)
{
offset = (row << 9) + column;
//Computer halfword address, set pixel
*(pixel_buffer + offset) = pixel_color;
++column;
}// END of WHILE-LOOP
}// END of FOR-LOOP
}// END of VGA_box

void displayVGA(char fn[]){
int i,j;
int offset, row, column;
        short int handler;
short att3 = 0, att2 = 0, att1 = 0;
int pixel;
short int *pixel_buffer = (short int *)SRAM_BASE; // VGA pixel buffer address

if (strcmp(fn,"blank")==0){
//Clear 640X480 screen
        printf("Screen cleared \n");
VGA_box (0,0,639,479,0xFFFF);
} else {
//Read Image from SD-Card
```

```
handler = alt_up_sd_card_fopen(fn, false);
//Skip Header
for (j = 0; j < 54; j++){
att1 = alt_up_sd_card_read(handler);
}
for (i = 240; i >= 0; i = i-1){
for (j = 0; j < 320; j++){
//Read each byte
att1 = (unsigned char)alt_up_sd_card_read(handler);
att2 = (unsigned char)alt_up_sd_card_read(handler);
att3 = (unsigned char)alt_up_sd_card_read(handler);
//24bit to 16-bit conversion
pixel = ((att3>>3)<<11) | ((att2>>2)<< 5) | (att1>>3);
for (row = i-1; row <= i-1; row++)
{
column = j;
while (column <= j)
{
offset = (row << 9) + column;
//Computer halfword address, set pixel
*(pixel_buffer + offset) = pixel;
++column;
}// END of WHILE-LOOP
}// END of FOR-LOOP
//pixel_buffer[i * 320 + j] = pixel;
//VGA_box (j, i, j, i, pixel);
}
}
alt_up_sd_card_fclose(handler);
}
}//END of displayVGA()

int main(void)
{int delaytime;
char ch;

alt_up_pixel_buffer_dma_dev *pixel_buffer_dev;
//      alt_up_video_dma_dev *char_dma_dev;


        pixel_buffer_dev = alt_up_pixel_buffer_dma_open_dev ("/dev/VGA_Subsystem_VGA_Pixel_DMA");
if ( pixel_buffer_dev == NULL)
        alt_printf ("Error: could not open VGA pixel buffer device\n");
else
        alt_printf ("BufferOpen \n");

/* clear the graphics screen */
alt_up_pixel_buffer_dma_clear_screen(pixel_buffer_dev, 0);

delay(3000000);
        draw_ECE_178 (pixel_buffer_dev);
delay(900000);
alt_up_sd_card_dev * sd_card;
                                sd_card = alt_up_sd_card_open_dev("/dev/SD_Card");
        //Local Variables
        int connected = 0;
        //Enable SD-Card Interface
        //If SD-Card Interface is enabled successfully
        if (sd_card != NULL){
```

```
        //If SD-Card is present on the DE2-115 Board
        if ((connected == 0) && (alt_up_sd_card_is_Present())){
        printf("Card Connected! \n ");
        //Check FAT16 File System
        if (alt_up_sd_card_is_FAT16()){
        printf("FAT16 File System Detected \n ");
        // Initialize the Start Menu
                displayVGA("\SMILE.BMP");
                delay(1000);
                displayVGA("\LOADING.BMP");
                delay(1000);
                displayVGA("\GAMEOVER.BMP");
                delay(1000);
                displayVGA("\SCORE.BMP");
                delay(1000000);
                displayVGA("blank");
        }}}
  return 0;
}
```

Verilog file

```
module Lights (CLOCK_50,CLOCK2_50, SW, KEY, LEDR, LEDG, HEX0, HEX1, HEX2, HEX3,
               HEX4, HEX5, HEX6,HEX7, DRAM_ADDR, DRAM_BA,DRAM_CAS_N, DRAM_RAS_N,DRAM_CLK,
                                      DRAM_CKE, DRAM_CS_N,DRAM_WE_N, DRAM_DQM, DRAM_DQ,
SD_CLK,SD_CMD,
                                      SD_DAT, VGA_CLK, VGA_BLANK_N, VGA_HS, VGA_VS, VGA_SYNC_N,
VGA_B,
                                      VGA_R, VGA_G, I2C_SCLK, I2C_SDAT,SRAM_ADDR,
        SRAM_DQ,
        SRAM_CE_N,
        SRAM_WE_N,
        SRAM_OE_N,
        SRAM_UB_N,
        SRAM_LB_N
                );
    input wire CLOCK_50;
         input wire CLOCK2_50;
    input wire [17:0] SW;
    input wire [3:0] KEY;
    output wire [17:0] LEDR;
    output wire [7:0] LEDG;
    output wire [6:0] HEX0;
    output wire [6:0] HEX1;
    output wire [6:0] HEX2;
    output wire [6:0] HEX3;
    output wire [6:0] HEX4;
    output wire [6:0] HEX5;
    output wire [6:0] HEX6;
         output wire [6:0] HEX7;
    output [12:0] DRAM_ADDR;
         output [1:0] DRAM_BA;
         output DRAM_CAS_N, DRAM_RAS_N,DRAM_CLK;
         output DRAM_CKE, DRAM_CS_N,DRAM_WE_N;
         output [3:0] DRAM_DQM;
         inout [31:0] DRAM_DQ;
         output SD_CLK;
        inout           SD_CMD;
```

```
        inout                   [ 3: 0]  SD_DAT;
        output [7:0] VGA_B, VGA_G,VGA_R;
        output VGA_CLK, VGA_BLANK_N, VGA_HS, VGA_VS, VGA_SYNC_N;
        output I2C_SCLK;
        inout   I2C_SDAT;

        // SRAM
output          [19: 0] SRAM_ADDR;
inout                   [15: 0] SRAM_DQ;
output                                  SRAM_CE_N;
output                                  SRAM_WE_N;
output                                  SRAM_OE_N;
output                                  SRAM_UB_N;
output                                  SRAM_LB_N;

projectSystemQsys NiosII(
                .av_config_SDAT(I2C_SDAT),              //            av_config.SDAT
                .av_config_SCLK(I2C_SCLK),
                .system_pll_ref_clk_clk(CLOCK_50),            //         clk.clk
                .green_leds_export(LEDG),  //  green_leds.export
                .red_leds_export(LEDR),
                .keys_export(KEY[3:1]),
                .system_pll_ref_reset_reset(1'b0),      //       reset.reset_n
                .seven_seg_0_export(HEX0), // seven_seg_0.export
                .seven_seg_1_export(HEX1), // seven_seg_1.export
                .seven_seg_2_export(HEX2), // seven_seg_2.export
                .seven_seg_3_export(HEX3), // seven_seg_3.export
                .seven_seg_4_export(HEX4), // seven_seg_4.export
                .seven_seg_5_export(HEX5), // seven_seg_5.export
                .seven_seg_6_export(HEX6), // seven_seg_6.export
                .seven_seg_7_export(HEX7),
                .switches_export(SW),     //    switches.export
        .sdram_clk_clk(DRAM_CLK),
   .sdram_addr                                                                  (DRAM_ADDR),
        .sdram_ba                                                                       (DRAM_BA),
        .sdram_cas_n                                                            (DRAM_CAS_N),
        .sdram_cke                                                                      (DRAM_CKE),
        .sdram_cs_n                                                                     (DRAM_CS_N),
        .sdram_dq                                                                       (DRAM_DQ),
        .sdram_dqm                                                                      (DRAM_DQM),
        .sdram_ras_n                                                            (DRAM_RAS_N),
        .sdram_we_n                                                                     (DRAM_WE_N),
                .sd_card_b_SD_cmd(SD_CMD),    //     sd_card.b_SD_cmd
                .sd_card_b_SD_dat(SD_DAT[0]),   //            .b_SD_dat
                .sd_card_b_SD_dat3(SD_DAT[3]),  //            .b_SD_dat3
                .sd_card_o_SD_clock(SD_CLK),
                .vga_CLK(VGA_CLK),   // vga_controller.CLK
                .vga_HS(VGA_HS),    //               .HS
                .vga_VS(VGA_VS),    //               .VS
                .vga_BLANK(VGA_BLANK_N), //               .BLANK
                .vga_SYNC(VGA_SYNC_N),  //              .SYNC
                .vga_R(VGA_R),     //               .R
                .vga_G(VGA_G),     //               .G
                .vga_B(VGA_B),      //               .B
                .vga_pll_ref_clk_clk(CLOCK2_50),        //      vga_pll_ref_clk.clk
                .vga_pll_ref_reset_reset(1'b0),
                        .sram_DQ(SRAM_DQ),
        .sram_ADDR(SRAM_ADDR),
        .sram_LB_N(SRAM_LB_N),
```

```
.sram_UB_N(SRAM_UB_N),
.sram_CE_N(SRAM_CE_N),
.sram_OE_N(SRAM_OE_N),
.sram_WE_N(SRAM_WE_N)
);

endmodule
```