1 Objective

.

This report documents the design, implementation and testing of a custom 8-channel Pulse Width Modulation (PWM) system on the Altera DE1-SoC FPGA board. The core objective was to control the brightness of on-board LEDs by modulating the duty cycle of a PWM signal. This was achieved by integrating a custom-designed PWM IP core into a Qsys system and programming a Nios II processor to control its parameters. The project relates key embedded systems concepts, including the use of Qsys for component integration, the Avalon-MM interface for memory-mapped communication and a polling-based method for handling user input from push buttons. The overall system was verified using the JTAG UART for debugging and by observing the varying light intensity on the LEDs.

2 Hardware Required

- Platform Designer Tool on Quartus
- Computer installed Quartus Prime 18.1v
- DE1-SoC FPGA Board
- USB Cable
- DE1-SoC 12V DC Power Adapter for board

3 Software Required

- Quartus Prime version 18.1
- Nios II SBT for Eclipse
- Qsys, Programmer
- Notepad/ text editor
- Word Processor

Documents Required: Pin Assignment - qsf file, DE1-SoC FPGA manual

4 Background

4.1 Platform Designer Tool

Platform Designer, a tool in Intel Quartus Prime which is generally used to visually build the custom hardware systems by connecting the respective IP blocks like processors memory and I/O peripherals. In this assignment, it was used to create Nios II-based system by adding components such as a CPU, on-chip memory, SDRAM, PLL, PIOs for LEDs and buttons and connecting them through an Avalon bus.

4.2 Quartus

Quartus Prime is Intel's all-in-one software bunch used for FPGA design, simulation and implementation. It supports writing the hardware descriptions in Verilog integrating systems using Platform Designer, setting up pin assignments, compiling designs and also programming the FPGA with the generated configuration file (.sof). In this assignment, Quartus Prime was used to build a complete system-on-chip (SoC) design with the Nios II processor and peripherals. The tool also handled synthesizing the hardware, checking timing and downloading the design onto the DE1-SoC board through the USB-Blaster interface. Quartus made it possible to turn our hardware design and logic into a working implementation on the FPGA.

4.3 Nios II SBT for Eclipse

Nios II Software Build Tools (SBT) for Eclipse is basically an IDE provided by Intel for writing, compiling and running embedded C/C++ programs on the processor. It allows developers to build applications that run on the custom hardware system created in Platform Designer. In this assignment, Nios II SBT was used to write the main C code, access I/O devices like LEDs and keys through memory-mapped addresses and load the compiled .elf file onto the DE1-SoC board. The tool also helps manage the Board Support Package (BSP), which contains drivers and HAL functions needed to interact with the hardware.

5 Assignment Details

Followed this steps for finishing the project.

5.1 Quartus

To begin the assignment, a new project was set up in Quartus Prime. A Verilog file was created using the same name as the top-level design module to ensure the consistency. From there, the Platform Designer (Qsys) tool was launched to construct the system and named it bindhw5

Within Platform Designer, several essential components were added to form the embedded system:

- A Nios II processor configured so vectors are pointing to sdram
- On-chip memory configured for tightly coupled access
- An SDRAM controller for external memory access
- System ID peripheral for verification
- JTAG UART for serial communication and debugging
- PIO (Parallel I/O) modules for:
    - Push-buttons(keys)
    - LEDs

Custom PWM IP Core (pwm.v): This is the important in the design. It's an 8-channel PWM generator that creates the modulated signals. The Nios II processor can configure the period and duty cycle for each of the eight channels by writing to its memory-mapped registers via the Avalon-MM interface.

Once all components were included, they were properly connected to share the same clock and reset lines. HDL files were then generated from Platform Designer. The .qip and corresponding Verilog output files were added back into the main Quartus project.

Afterward, the Qsys-generated Verilog module was instantiated in the top-level file. The .qsf pin assignment file was then loaded to match the I/O with the DE1-SoC board. The project was then synthesized and compiled successfully.

5.2 Nios II SBT for Eclipse

The software was developed using Nios II Software Build Tools (SBT) for Eclipse. The Board Support Package (BSP), generated from the Platform Designer (.qsys) file, provided the necessary Hardware Abstraction Layer (HAL) drivers and peripheral base addresses.

The C program implements the following core functionalities:

PWM Initialization & Configuration:

- **Action:** Sets the target PWM frequency and initializes all eight PWM channels to a default brightness.
- **Mechanism:** Defines constants for a 2 kHz PWM frequency based on the 50 MHz system clock. The set_all_channels() function is used to write the calculated period and duty cycle values to the custom PWM IP's memory-mapped registers via the IOWR_32DIRECT macro.
- **Output:** The LEDs initially display a medium brightness level.

Polling-based Input Handling:

- **Action:** Continuously monitors the state of the push buttons (KEY[1], KEY[2], KEY[3]) to detect a button press.
- **Mechanism:** A while(1) loop actively polls the PIO input register. Logic is implemented to detect a rising edge (the moment of a button press) and includes a small delay (usleep) to handle button debouncing, preventing multiple signals from a single physical press.
- **Output:** Triggers an update to the PWM duty cycle when a new button press is confirmed.

PWM Control Logic & Feedback:

- **Action:** Modifies the duty cycle of the PWM signal for all channels based on the detected button press.
- **Mechanism:** Based on which key is pressed, the program calls set_all_channels() with a predefined duty cycle value (high, medium or low). The logic is structured so that KEY[1] produces the highest brightness, KEY[2] a medium brightness and KEY[3] the lowest brightness

- **Output:** Real-time feedback is provided via the JTAG UART, which displays the key pressed and the corresponding period and duty cycle values. The visual output is the change in light intensity on the LEDs.

Upon compilation and loading onto the DE1-SoC board, all program functions were successfully verified.

6 Procedure

The procedure has all details of the homework.

6.1 Setting up Quartus Project

The hardware portion of this project was developed using Quartus Prime Lite Edition, where a new FPGA design project was set up from base. The process began by launching the Quartus software from the desktop environment.

Figure 1: Opening Quartus Prime and Intel FPGA Monitor Program from the desktop.

Once Quartus was launched, the main workspace appeared. The New Project Wizard was selected from the File menu to begin creating the FPGA project.

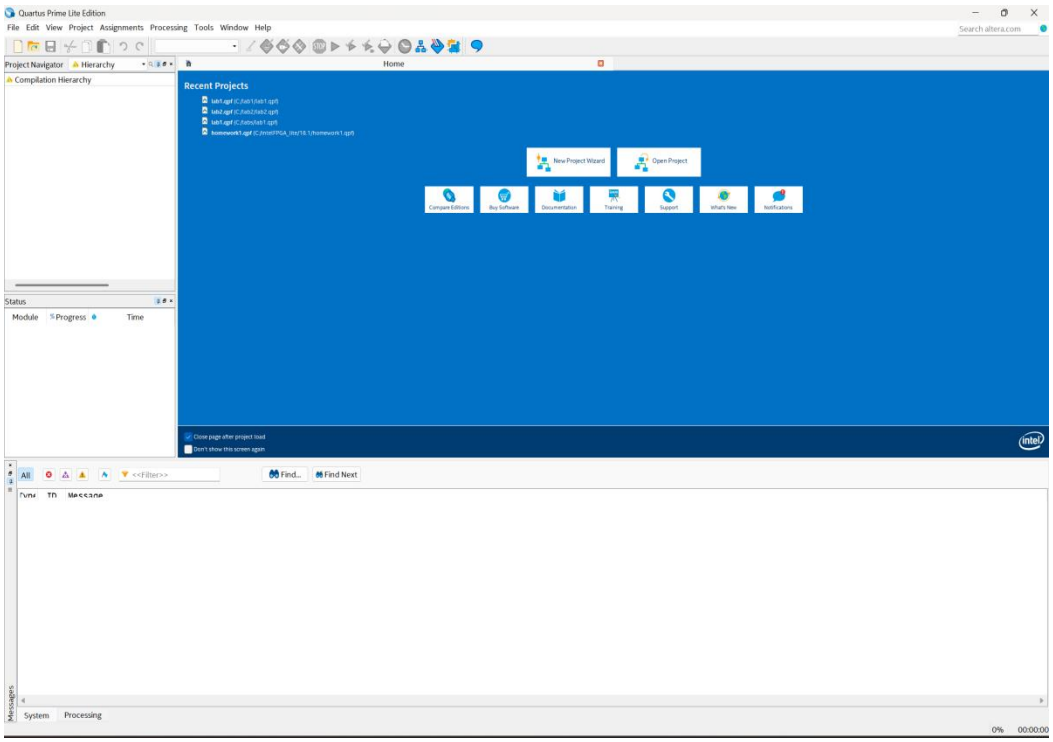


Figure 2: Accessing the New Project Wizard from the Quartus File menu.

The software displayed the welcome screen of the New Project Wizard, outlining the configuration steps to be completed before project creation.

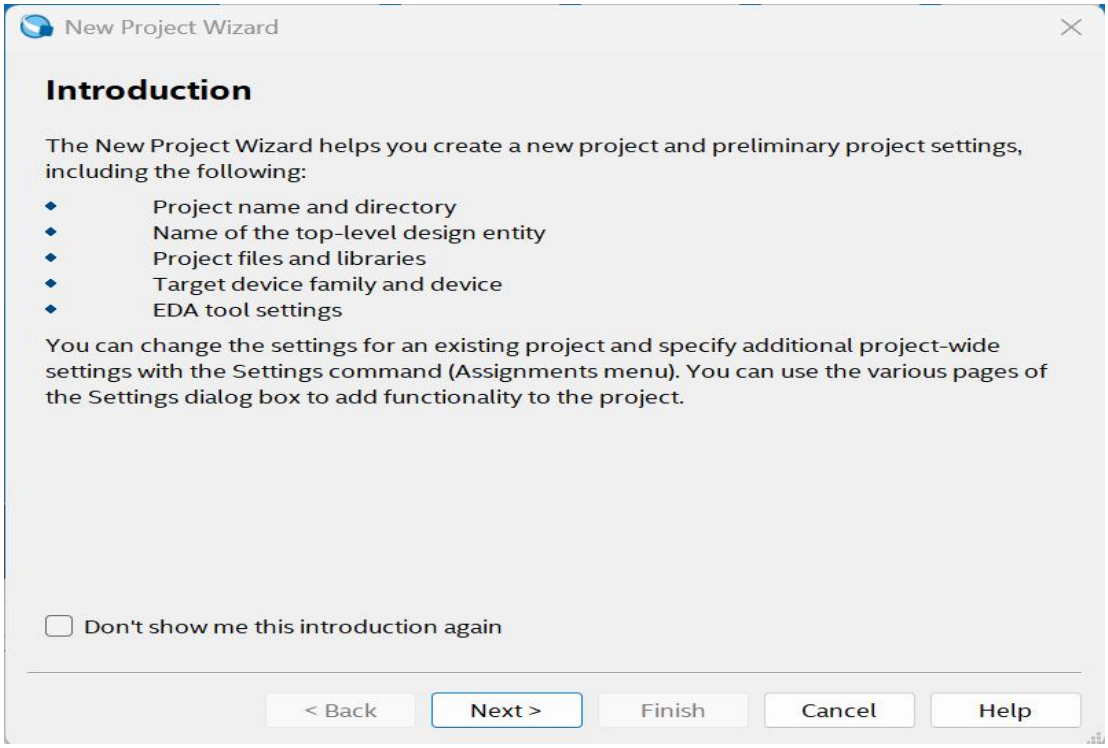


Figure 3: Introduction screen of the New Project Wizard, outlining key setup parameters.

The working directory, project name and top-level module name were specified. In this case, the project was named and the top-level design entity matched the same name to maintain consistency with the Verilog module.

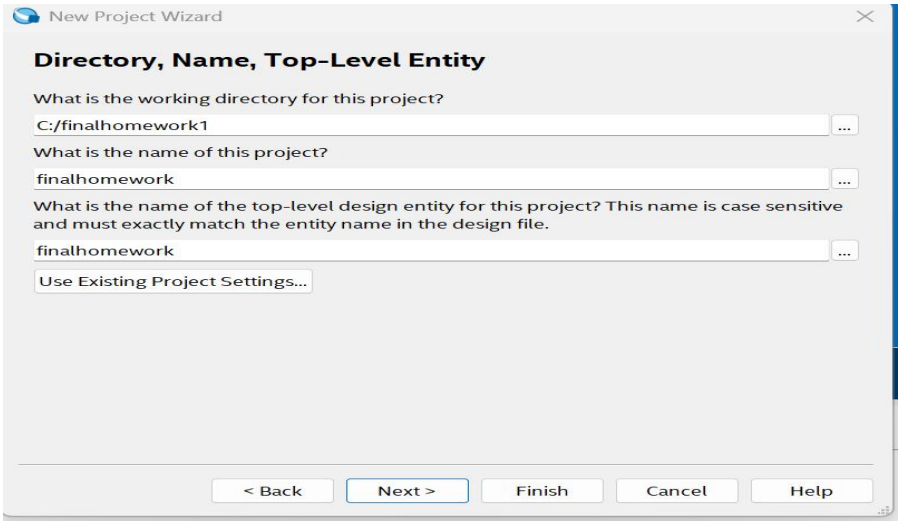


Figure 4: Defining the project directory, name and top-level design entity.

Next, the project type was selected. "Empty Project" was chosen to allow manual configuration of all components and files throughout the design process.

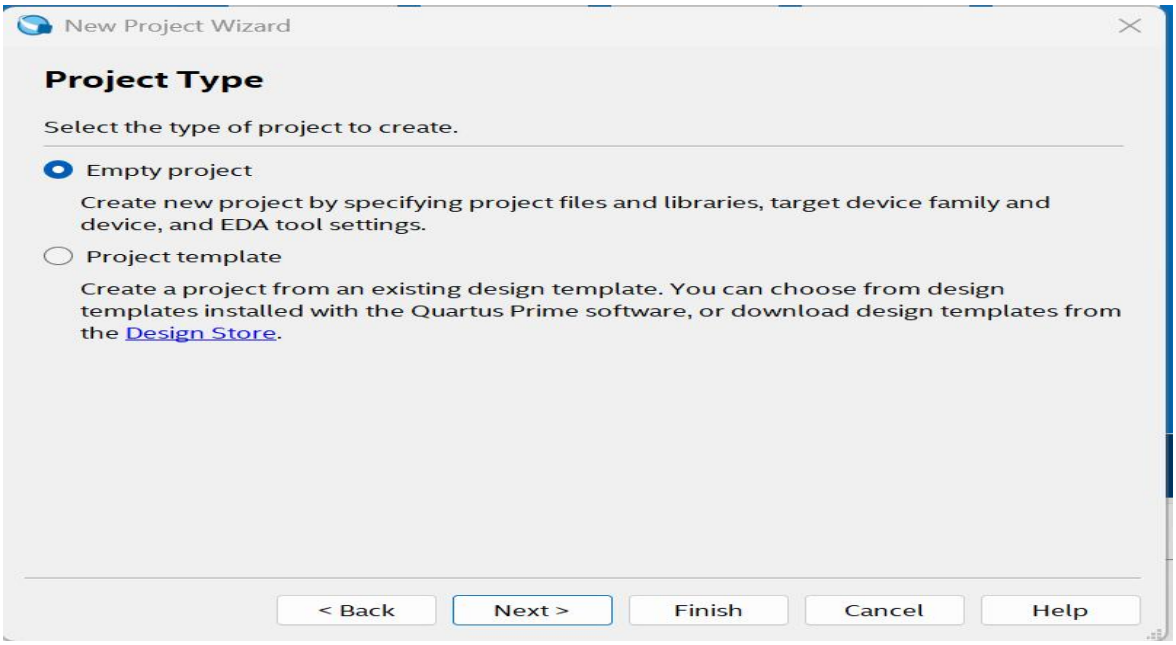


Figure 5: Choosing the "Empty Project" option for a custom setup.

No files were added at this stage since the Verilog files and Qsys outputs were to be generated and added later. The wizard allows files to be added at any point after project creation.

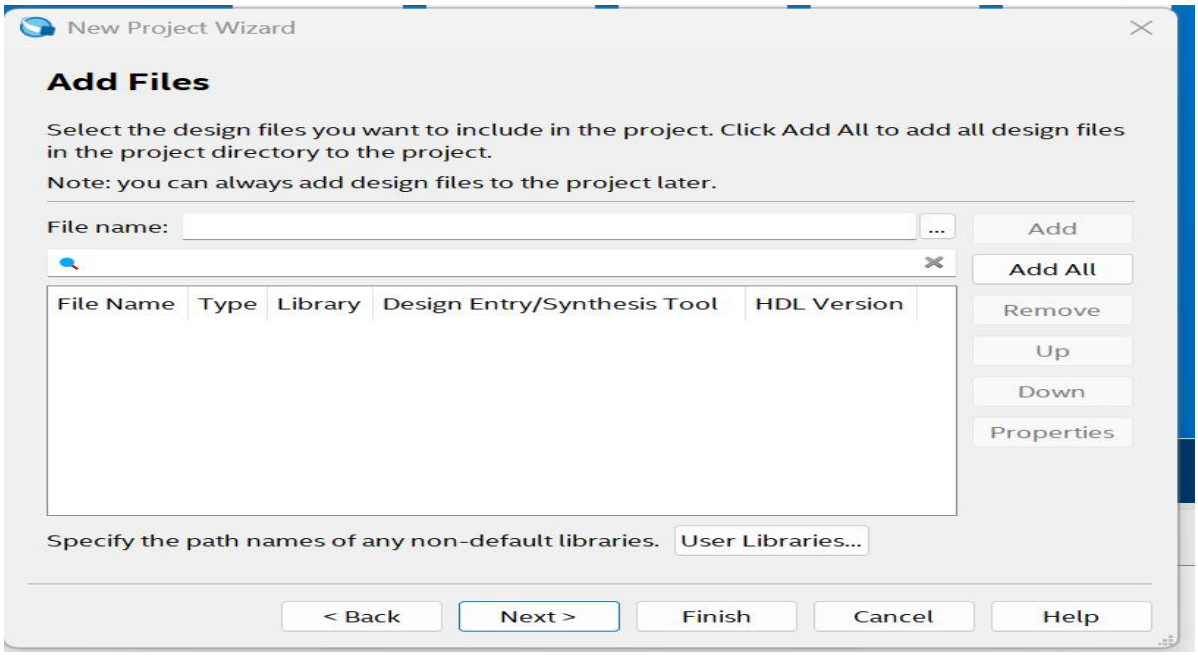


Figure 6: Skipping the file addition step for now, to be handled manually after Qsys generation.

The target FPGA device used on the DE1-SoC board, **Cyclone V – 5CSEMA5F31C6**, was selected from the list of available devices. This ensures the generated bitstream is compatible with the development board.

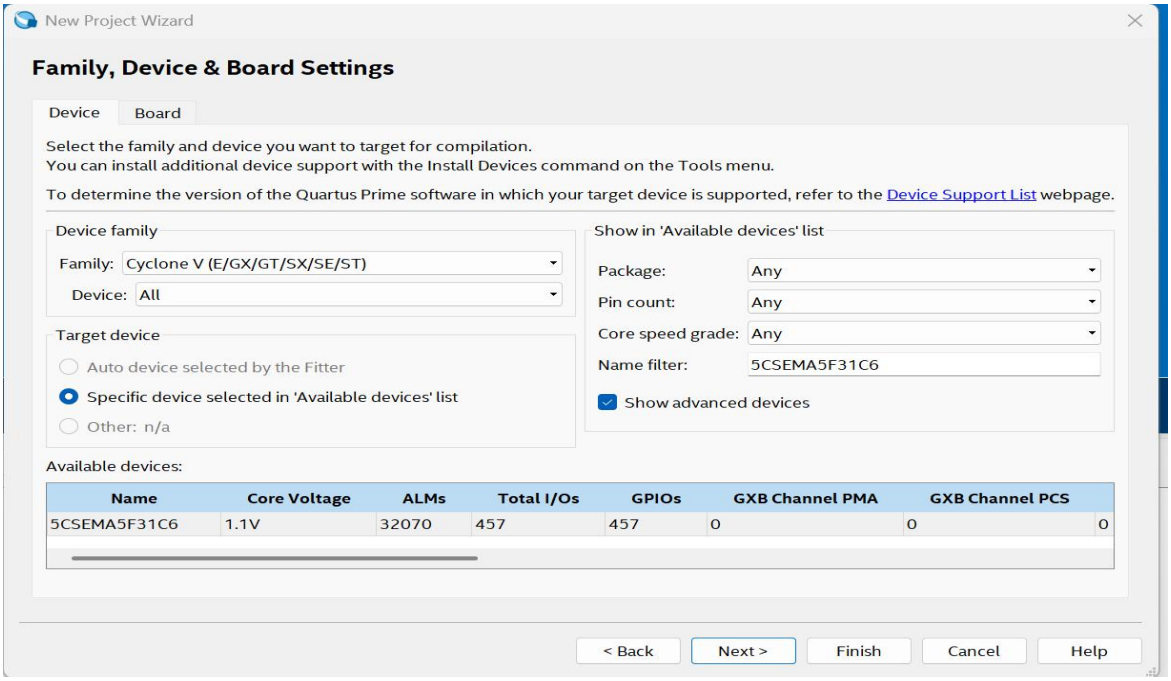


Figure 7: Selecting the Cyclone V device used on the DE1-SoC FPGA board.

EDA tool integration settings were left at their default values, as no simulation or third-party tools were needed in this assignment.

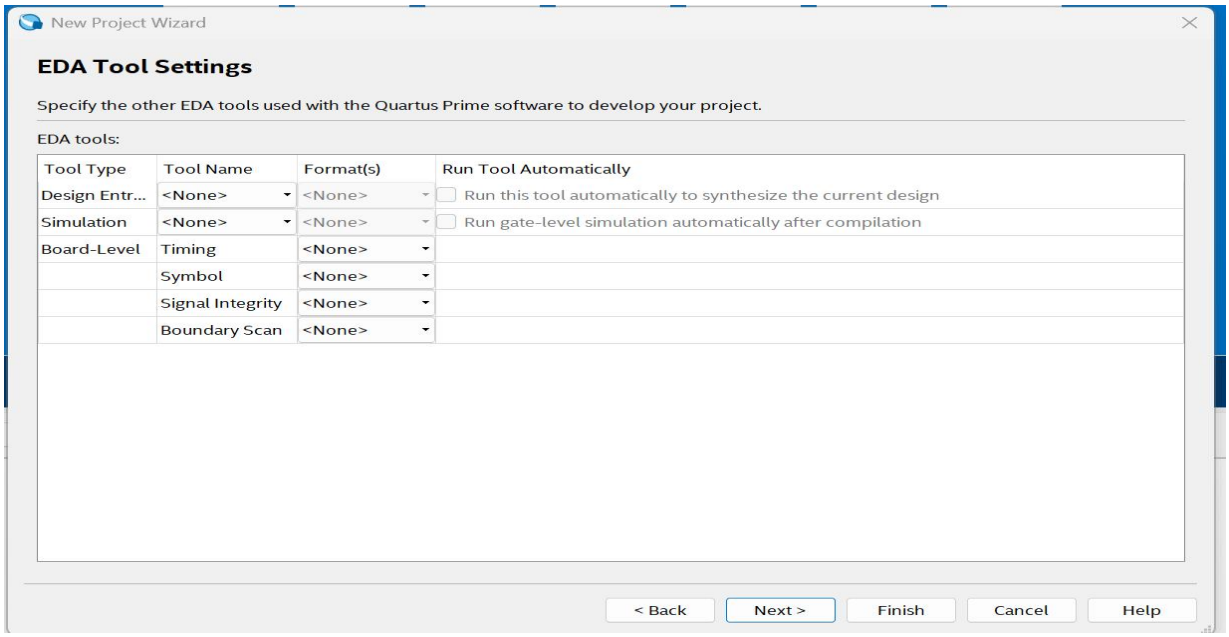


Figure 8: Keeping default settings for EDA tool configuration.

Finally, the project summary was displayed, showing all selected configurations including device family, top-level entity name and board details. Once reviewed, the project was created by clicking the "Finish" button.

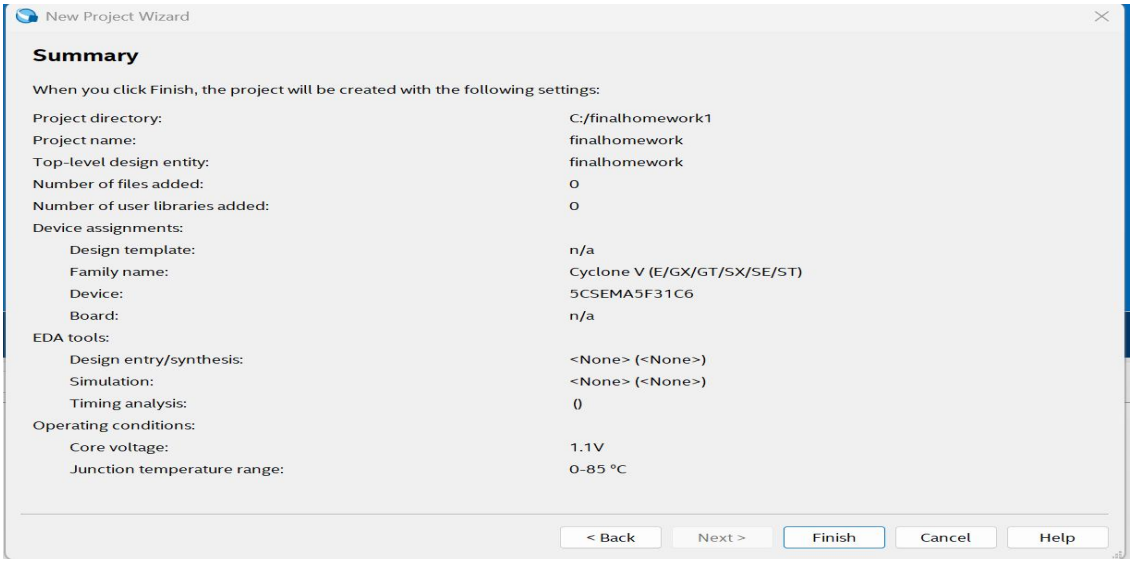


Figure 9: Summary screen confirming all project configuration settings before final creation.

With the project created, Quartus returned to the main workspace. From here, additional design files could be added and the Platform Designer (Qsys) tool could be launched to begin building the custom embedded system for the assignment.

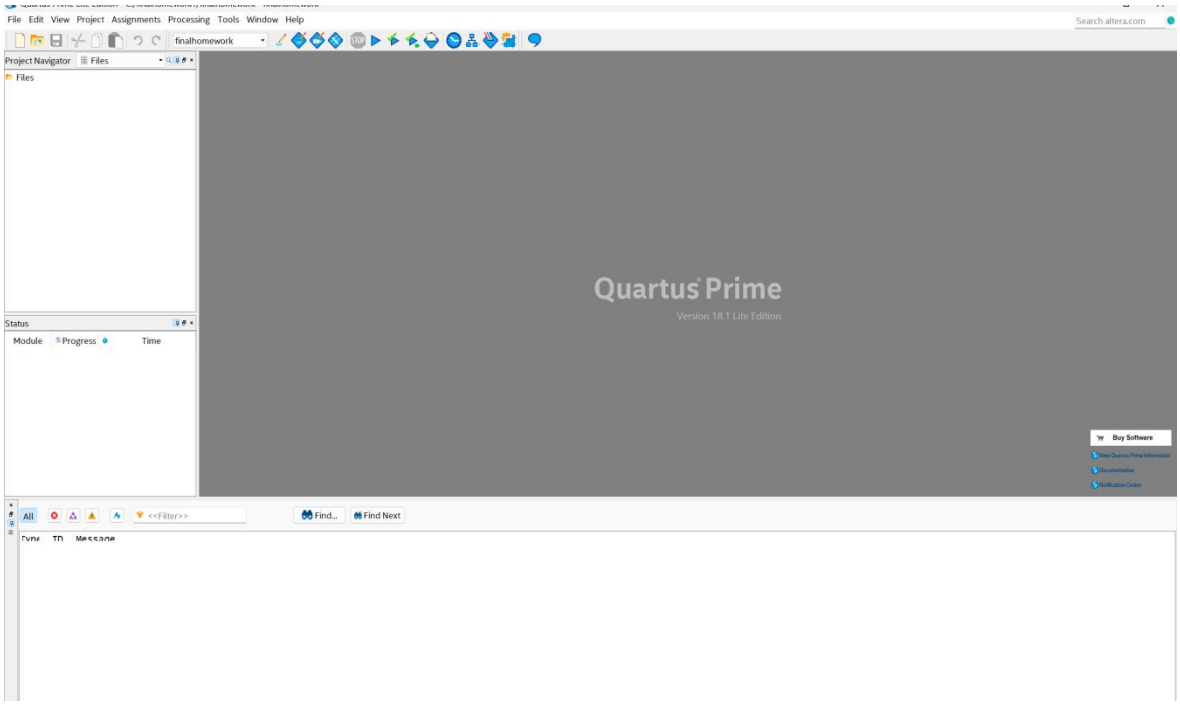


Figure 10: Quartus Prime main interface after project creation, ready for system integration and further development.

6.2 Qsys

To build the system, I first opened **Platform Designer** (also called Qsys) from the **Tools** menu in Quartus. This tool helps in adding and connecting different components like the processor, memory, timers and input/output devices.

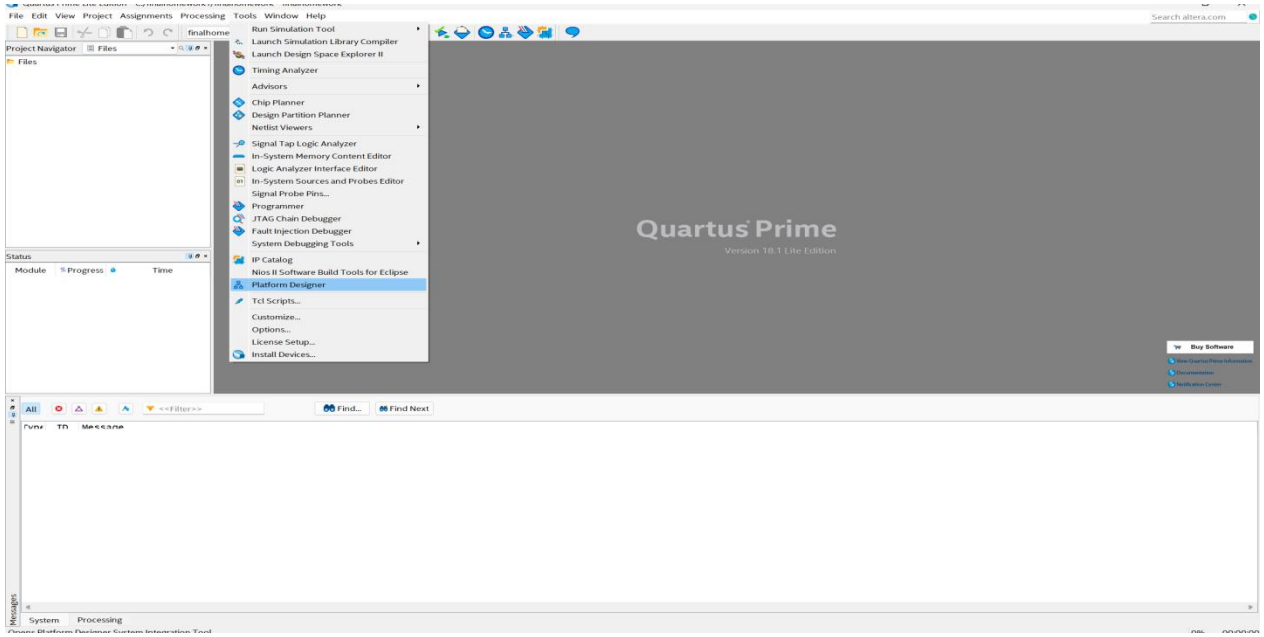


Figure 11: Opening Platform Designer from the Quartus Tools menu.

After Platform Designer launched, this is default window opens in canvas space with a default clock Source. In this homework, I used pll clock, so removed this from space.

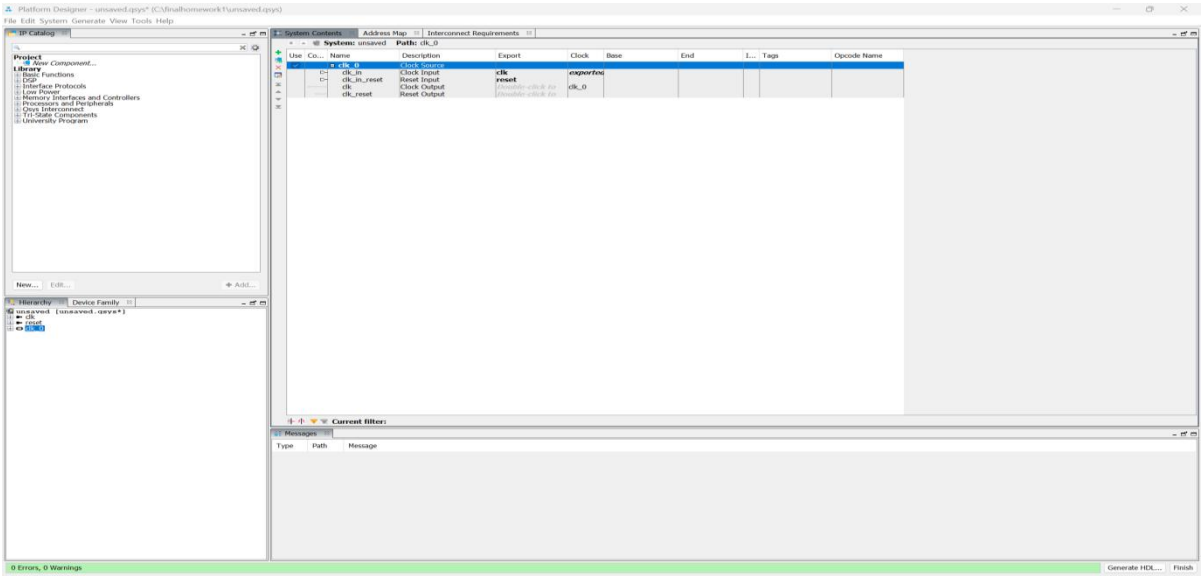


Figure 12: canvas space in qsys, Clock Source component added to the system. Removed this later

This reference clock runs at 50 MHz and desired system clock is 100 MHz and selected DE-1 SOC is required for other components like the processor and timers to work properly.

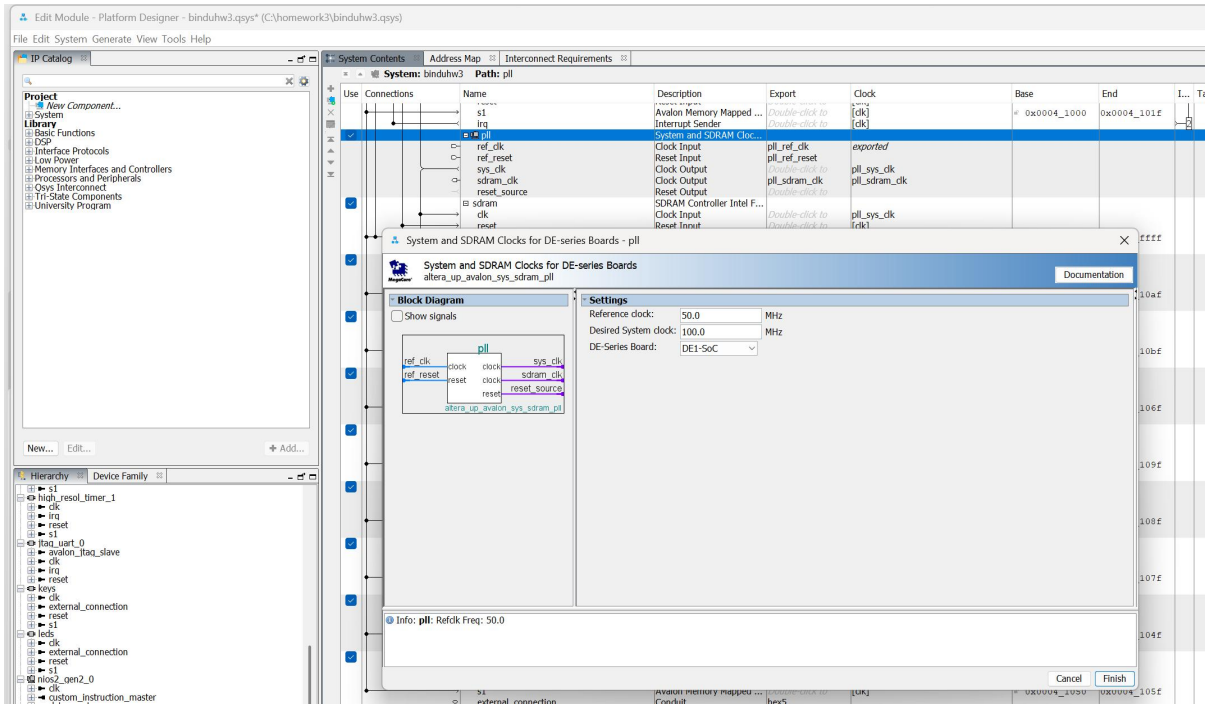


Figure 13: Setting the pll Clock frequency.

Added the Nios II processor to our Platform Designer system by selecting it from the "Embedded Processors" category under "Processors and Peripherals."

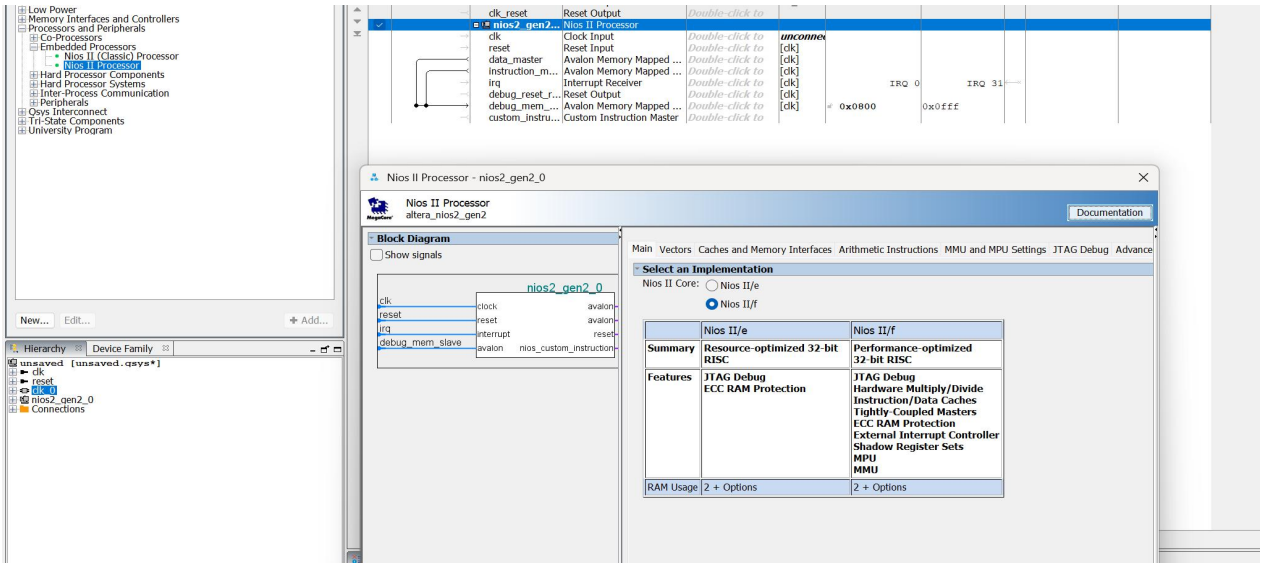


Figure 14: Selecting the Nios II Processor from the IP Catalog and adding into design

The On-Chip Memory component was added and its size was set to large bytes. The memory was configured to be initialized at system startup. Dual port access is enabled.

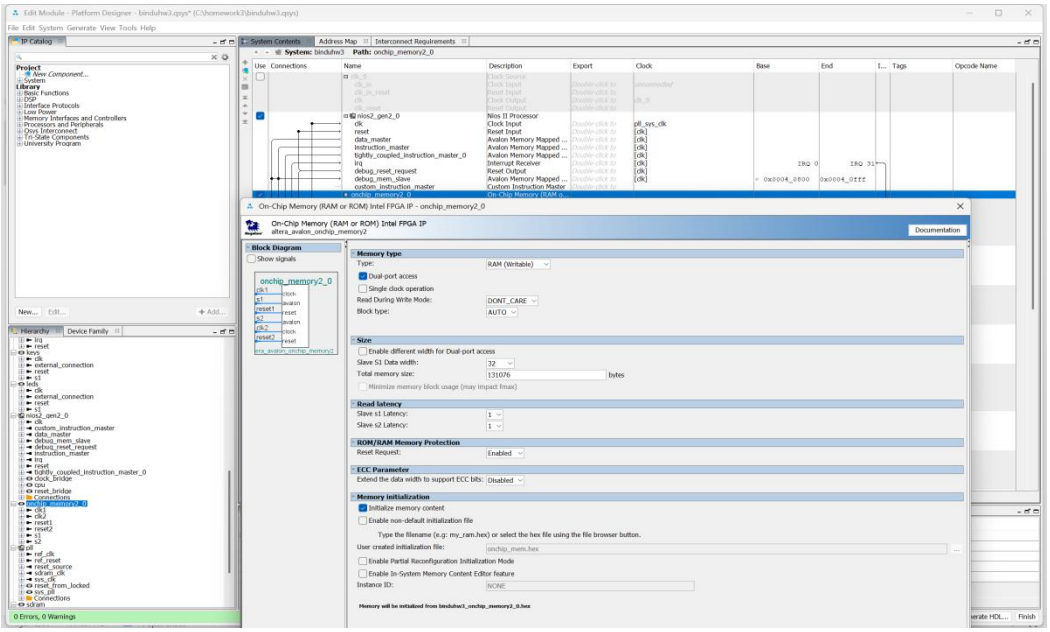


Figure 15: On-Chip Memory Configuration

Instruction and data cache were enabled to improve performance. A 4 KB instruction cache and a 2 KB data cache were selected. Also gave 1 to tightly coupled master ports for data and instruction ports.

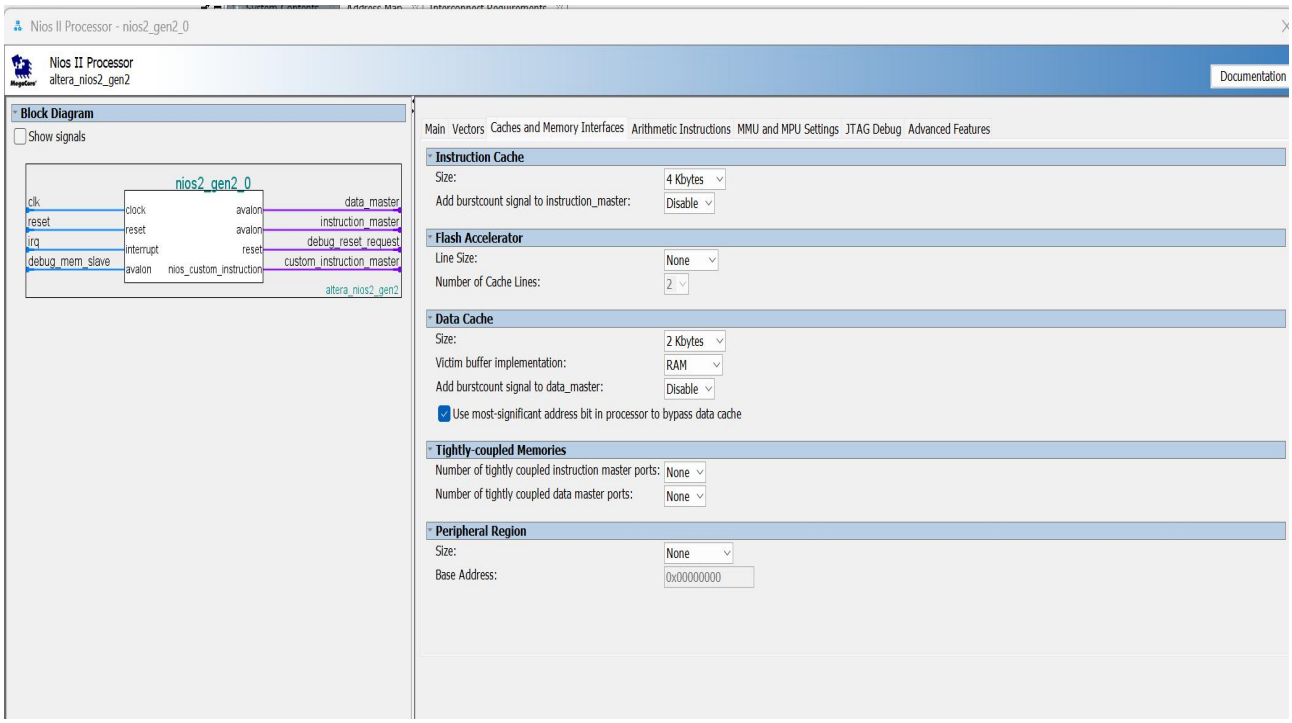


Figure 16: Configuring Cache for Nios II Processor

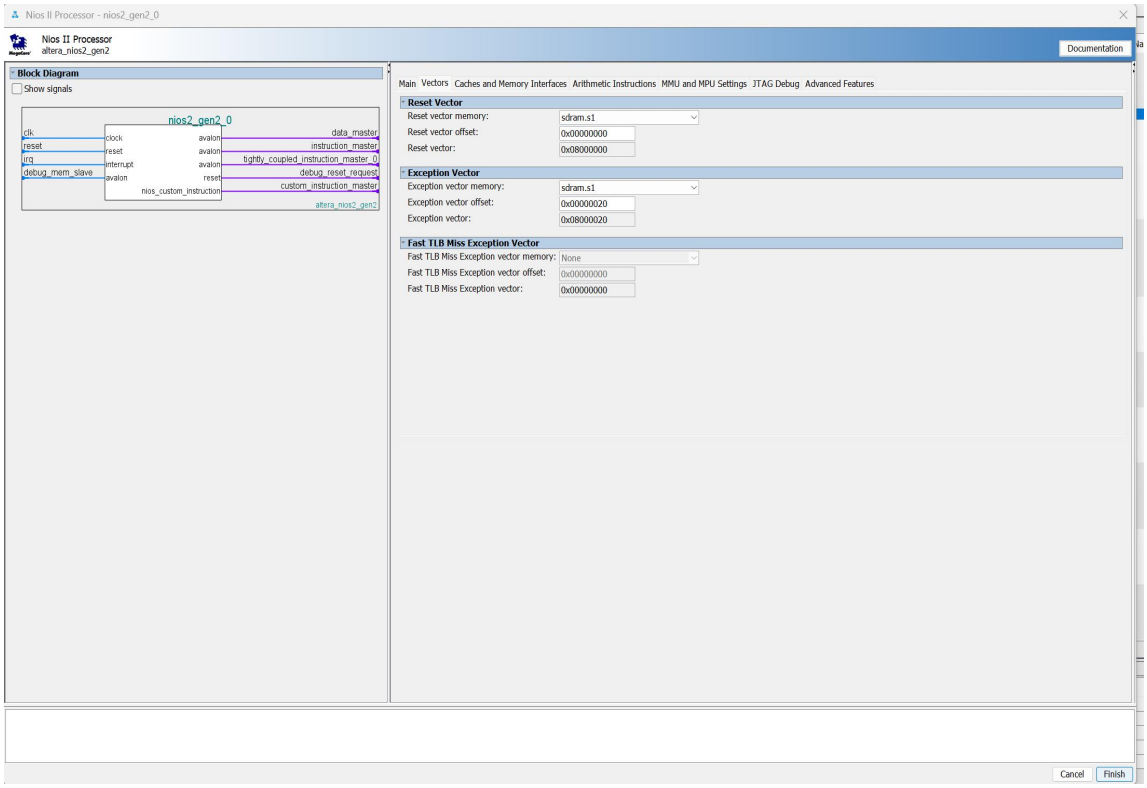


Figure 17: Setting Reset and Exception Vectors

The reset and exception vectors for the processor were both mapped to the sdram. This ensures the processor starts execution from the correct memory location after reset or interrupt.

In Platform Designer, the “System ID Peripheral Intel FPGA IP” is added from the IP Catalog under "Simulation; Debug and Verification." This block helps ensure software compatibility with the hardware system.

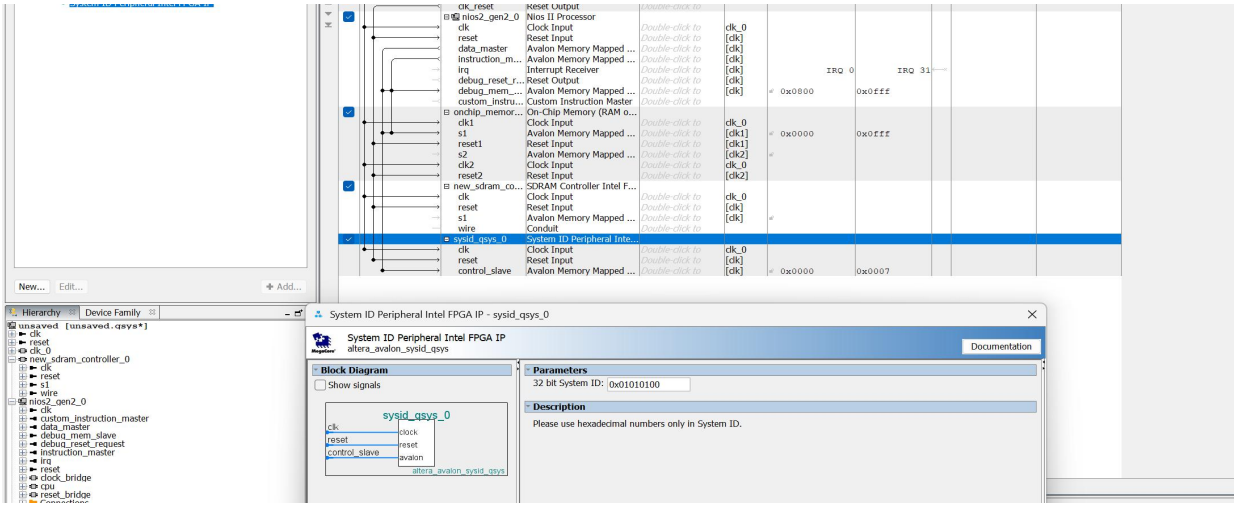


Figure 18: Configuring system ID peripheral

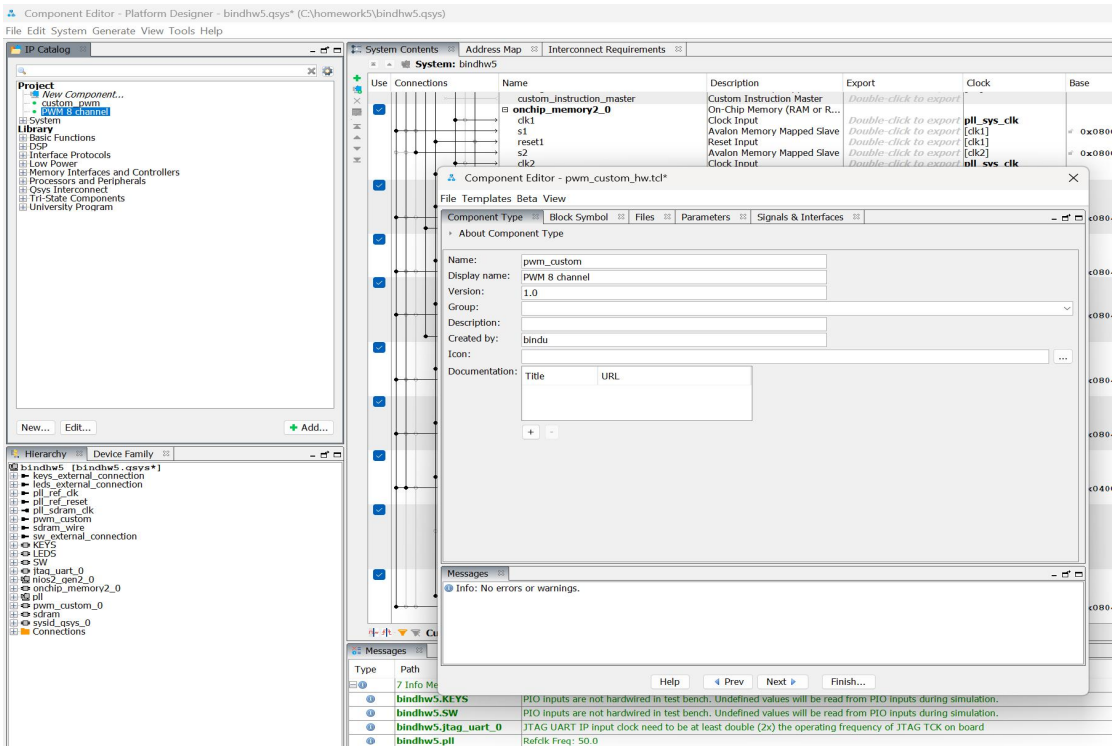


Figure 19: New component need to be added

Under project in IP catalog, could see the new component, from there added into my existing design. Once it opens, name and a display name was given and then under files tab, added the pwm.v file into it and click on analyze synthesis files. It automatically selects this file as top level module and

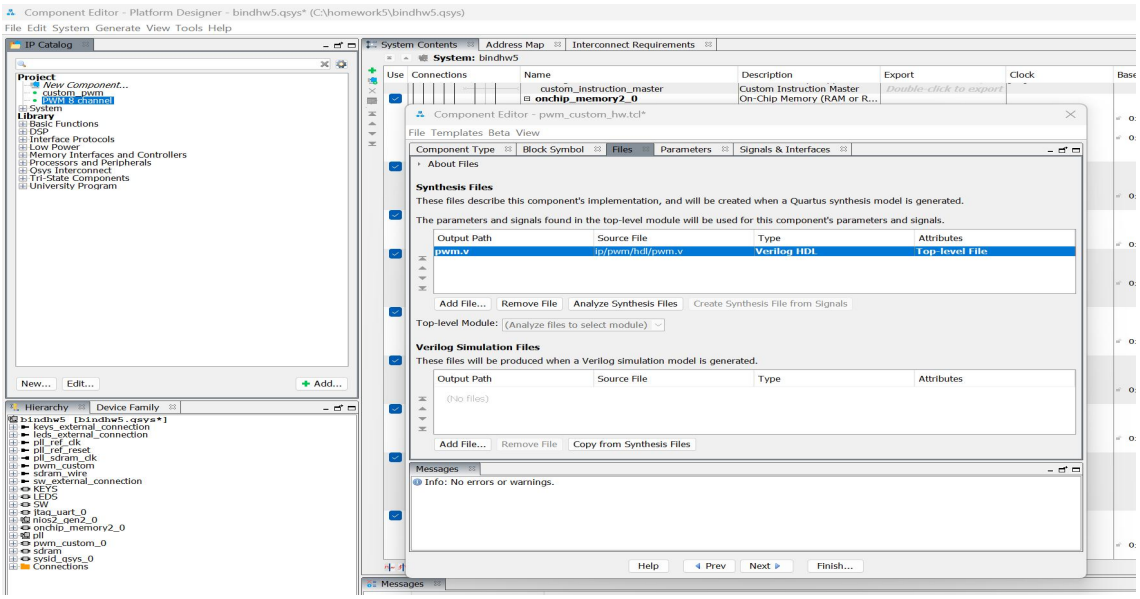
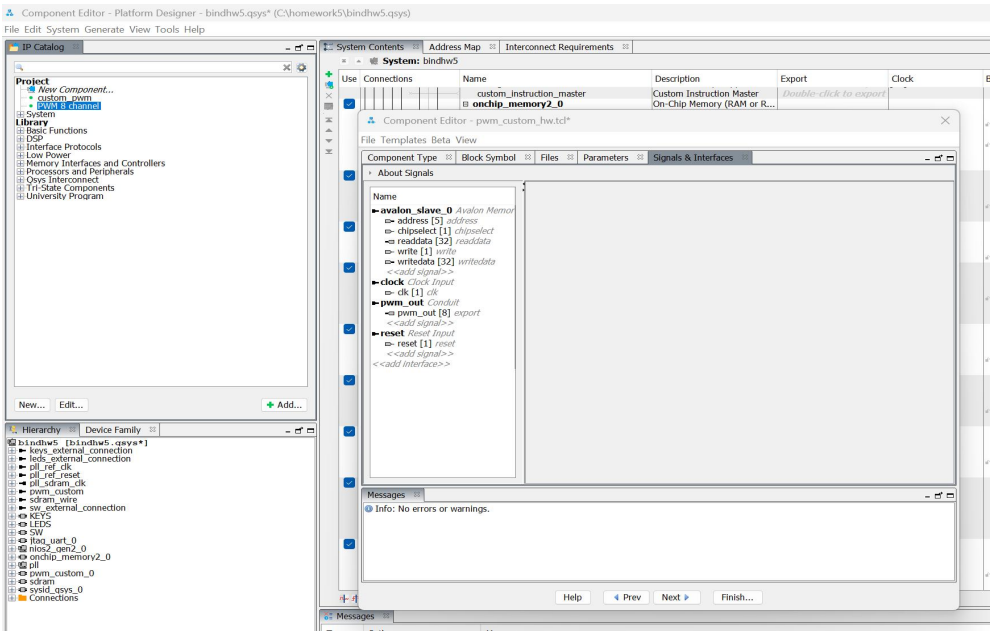


Figure 20: Add verilog file into the synthesis file tab

Once after adding file, I saw some errors at message section. To resolve this I moved on to the signals and interfaces tab. Assigned and kept the correct interfaces and added reset signals and removed unwanted signals. This looks like below.



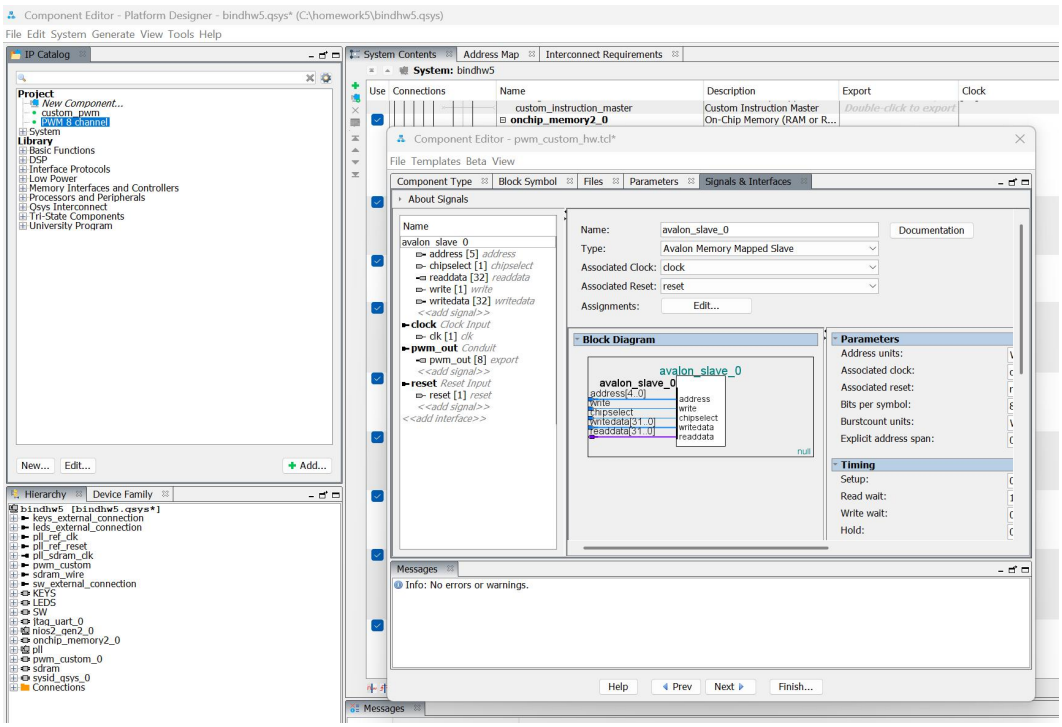


Figure 21: signals and interfaces, interfaces

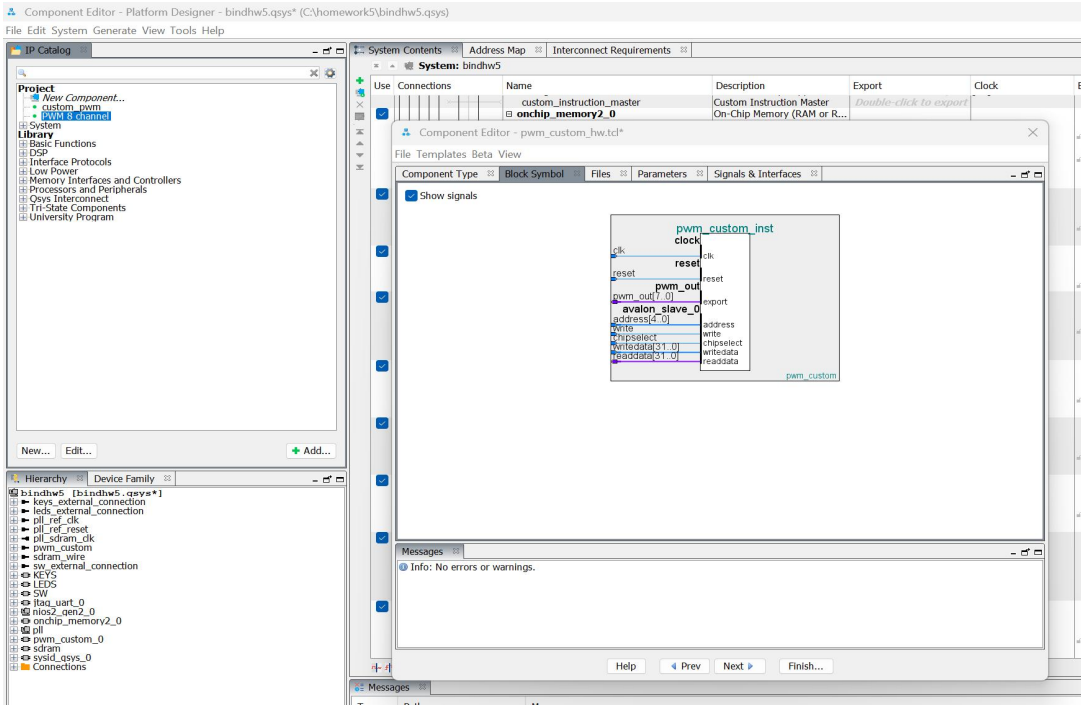


Figure 22: block diagram of pwn component

The Switch PIO (Parallel I/O) component is added and configured as an input with specific width. It will be used to read switch values from the DE1-SoC board.

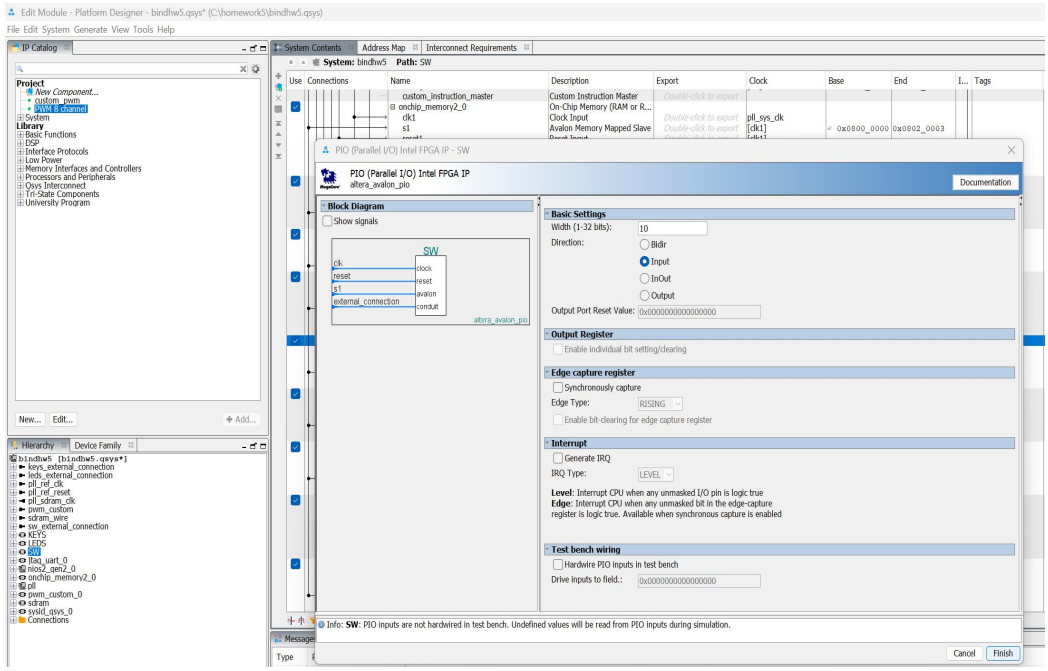


Figure 23: Configuring switches

The LEDR PIO is also added and configured as an output. This component will send data to the red LEDs. The width is set to 10 bits to match the number of available LEDs.

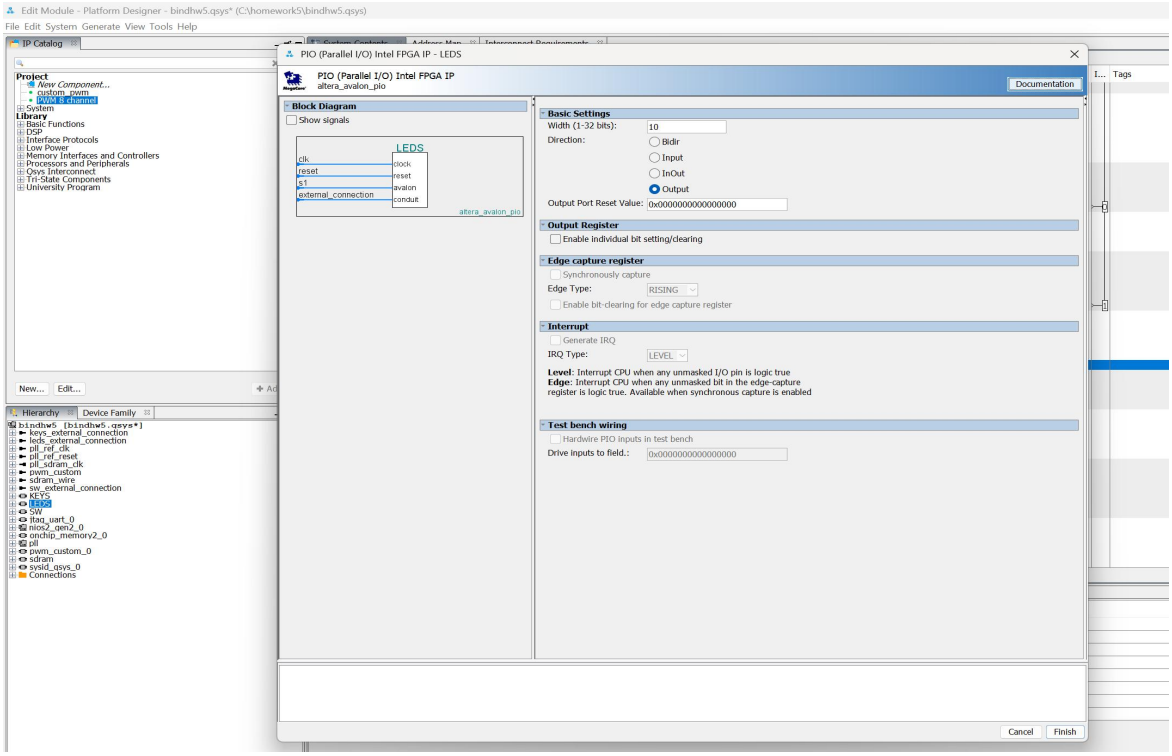


Figure 24: Configuring LEDR PIO

A PIO named KEYS was added for push-button input. It was set to 4-bit input and interrupt generation was enabled, allowing the processor to respond to button presses. Connected all the clocks, resets and master ports, irq and assigned priorities. Also connected the external conduits of the peripherals.

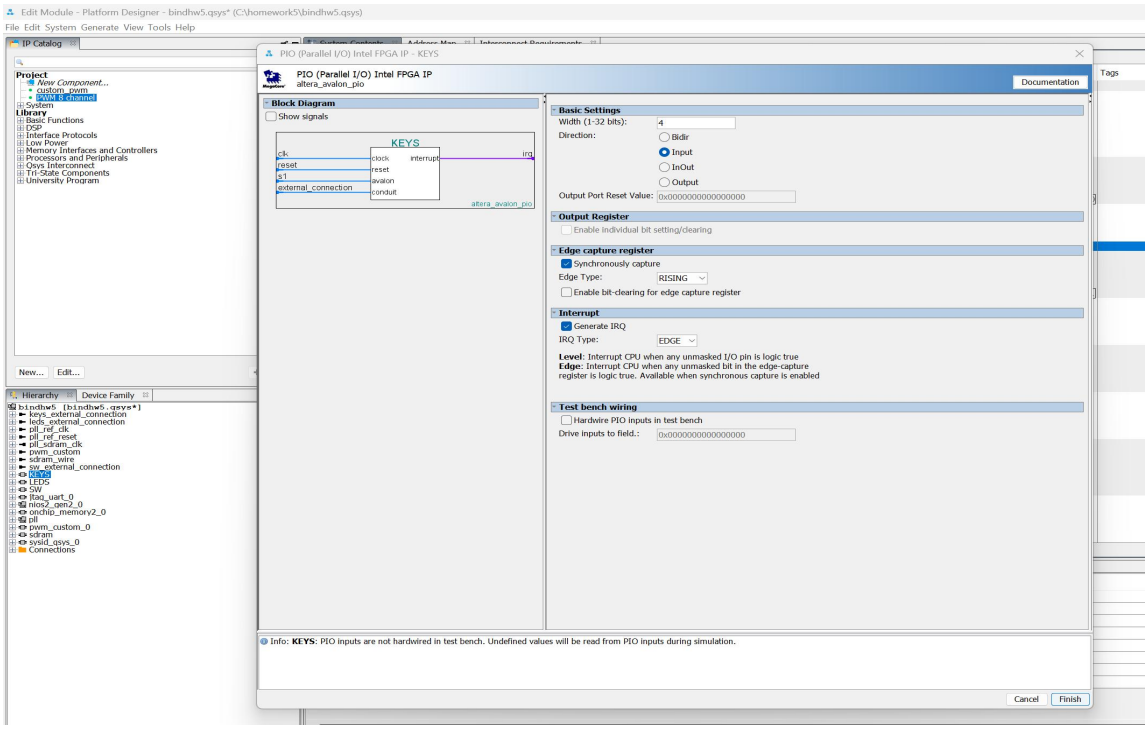


Figure 24: Configuring keys as input (PIO)

Once all the connections are made and complete, next step is to assign base address and avoid overlapping ranges. So clicked on auto assign base address and this resolved many errors.

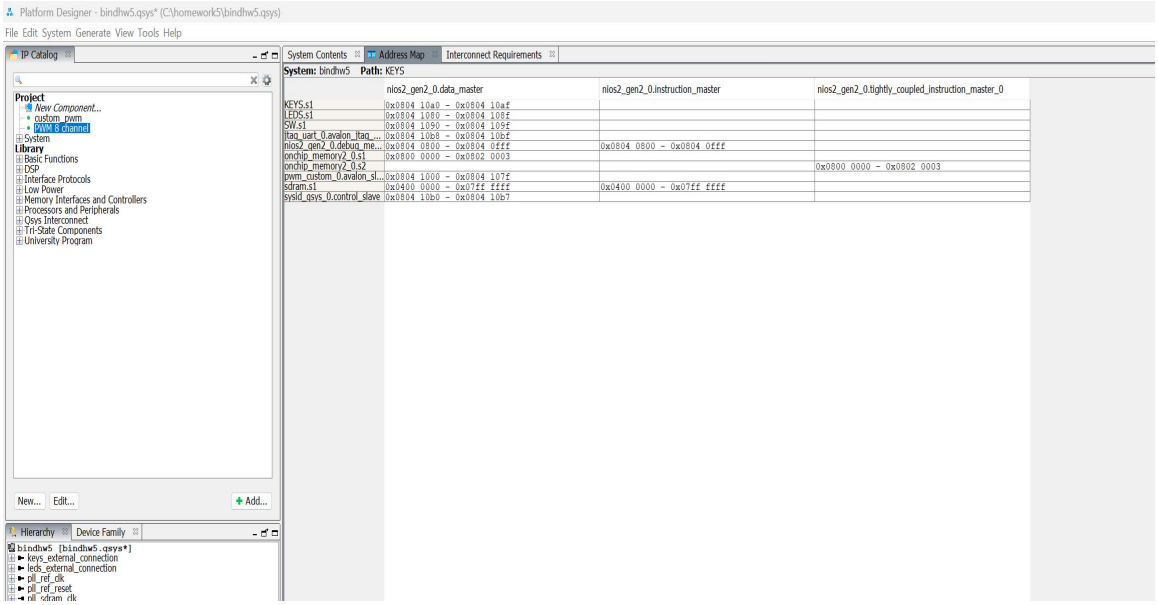


Figure 25: Memory allocation doesn't overlap. Auto assign base address

Final system layout qsys,displaying all IP blocks and their connections along with address mappings.

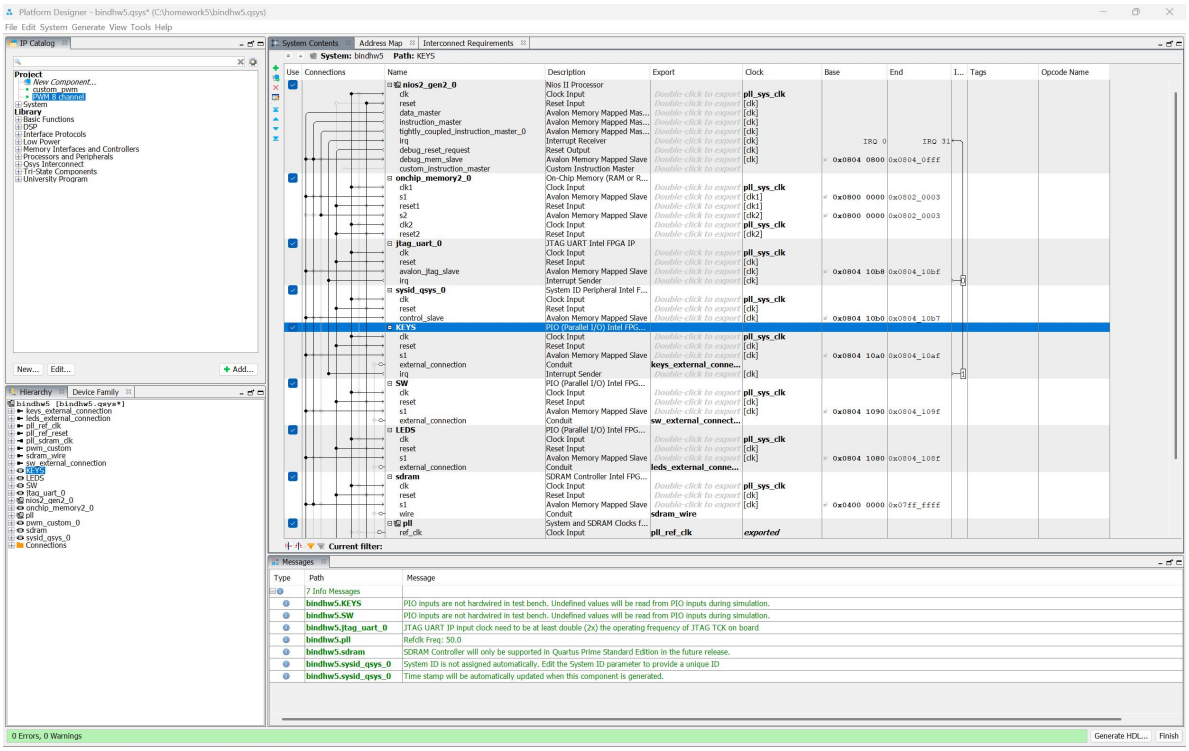


Figure 26: qsys design layout

This figure captures the system connections screen, where all modules like the seven segment, LED, keys, JTAG UART, pll, SDRAM, on chip memory and timers are wired properly. It ensures that the data, address and clock signals are connected before HDL generation.

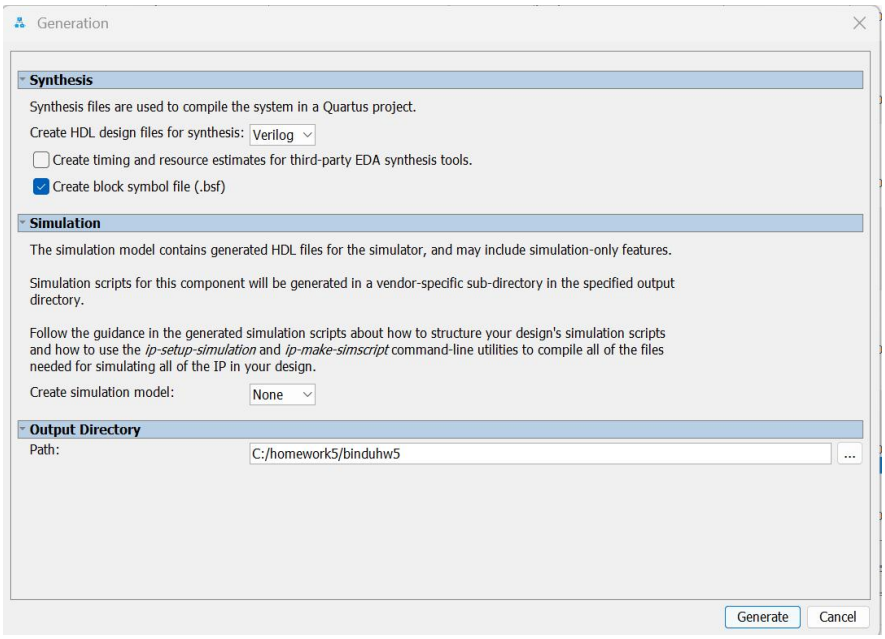


Figure 28: Generate HDL

This window appears when we click Generate HDL in Platform Designer, selected Verilog as the HDL language This .bsf file helps us add the system to the main project easily. The output path shows where the generated files will be saved.

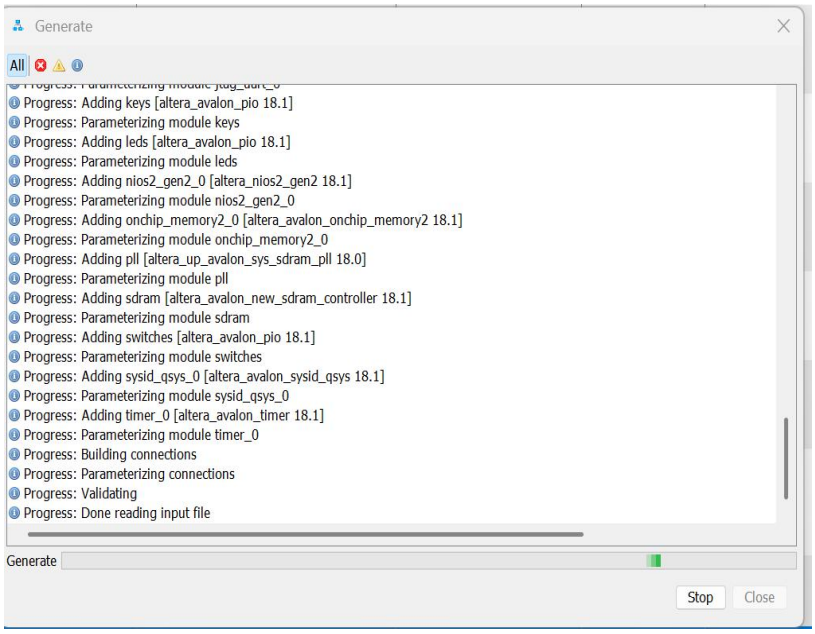


Figure 29: Generation process in progress

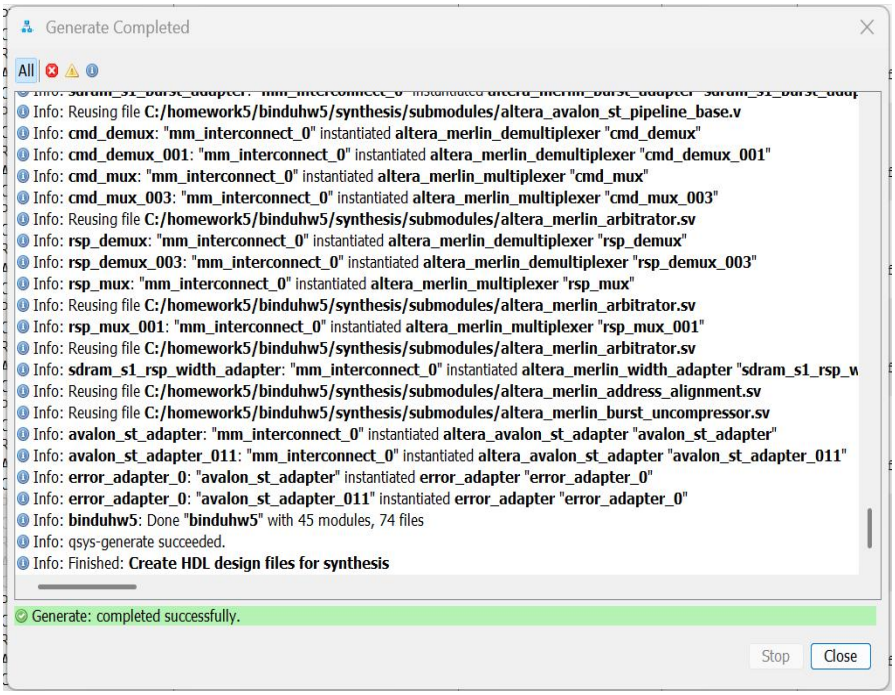


Figure 30: Generation completed successfully

This is the final step of HDL generation. The message “Generation completed successfully” confirms that the Verilog files for our Platform Designer system were created without any errors. We can now use these files in our main Quartus project.

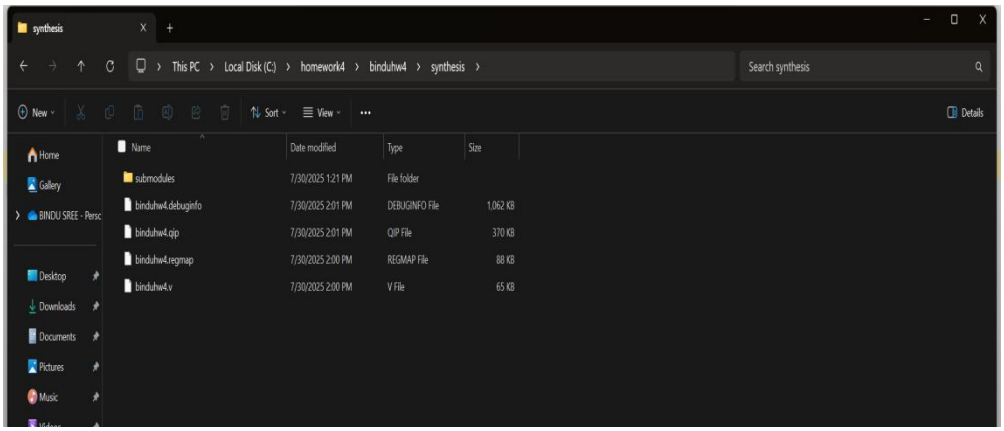


Figure 31: Files created in project file directory

6.3 Quartus top level setup

Back in quartus, created a verilog top level module file and imported pin assignments. And then, compiled the file.

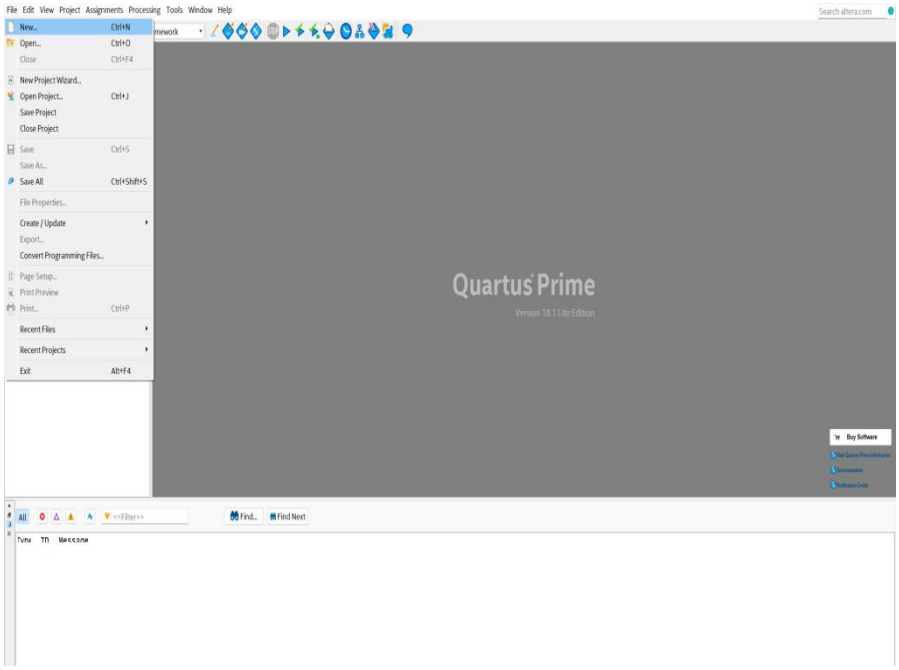


Figure 32: Creating new file from Quartus File Menu

This screenshot shows how to access Platform Designer in Quartus Prime by navigating to Tools > Platform Designer. This step launches the system integration tool required to build the hardware system.

This window appears when creating a new file in Quartus. The dialog allows users to choose the type of design file to create, including Verilog, VHDL, Block Diagram and Qsys System Files. Selected the Verilog VHDL file type here and saved. Make this file as top level entity.

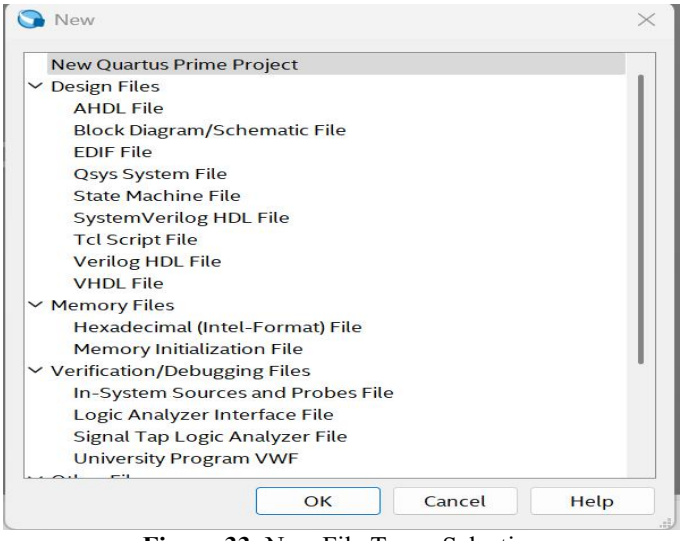


Figure 33: New File Types Selection

Next step is to add design files to the Quartus project. The user navigates to Project > Add/Remove Files in Project to include Qsys-generated files into the current project.

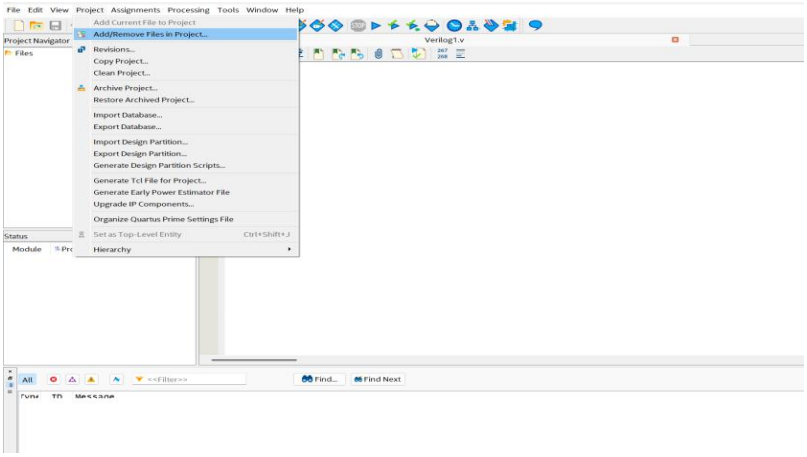


Figure 34: Adding Files to the Quartus Project

Adding Verilog design file to the project by selecting Add/Remove Files in Project from the Project menu.

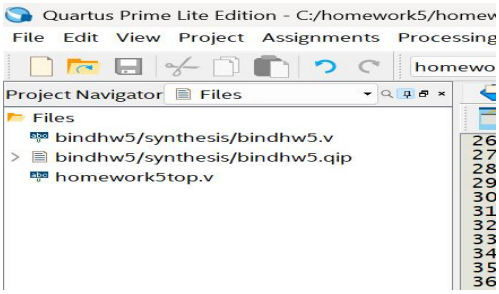
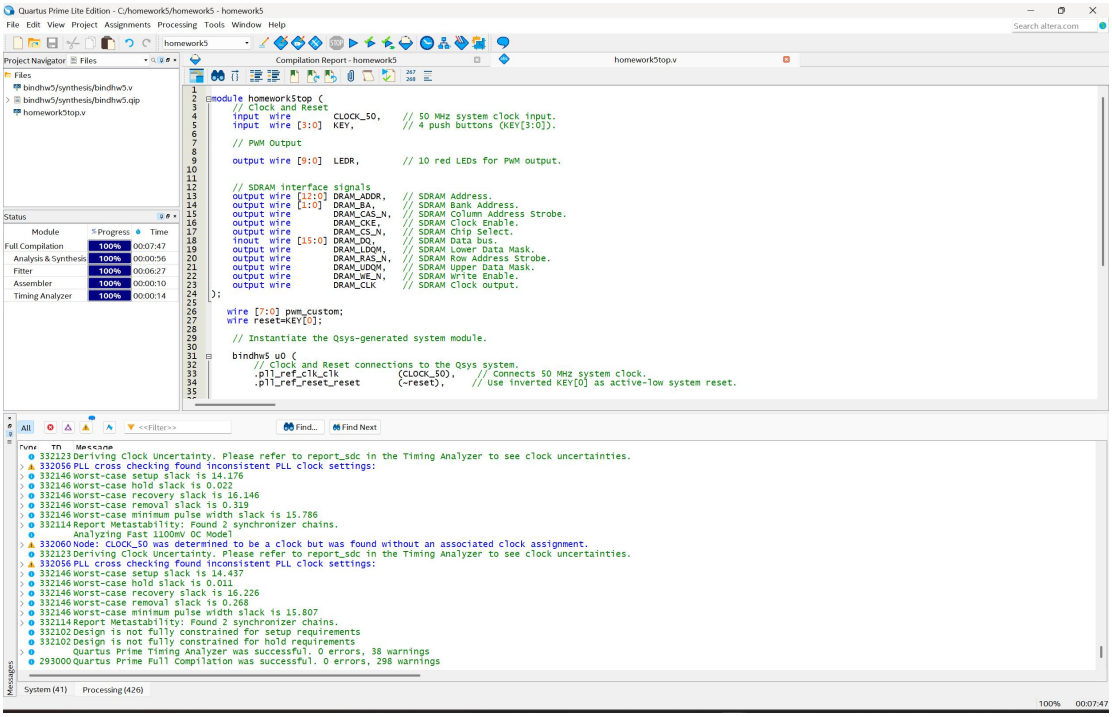


Figure 35: In project navigator, these loaded files can be seen



This figure shows the finalhw1 top-level Verilog module declaration for integrating the Platform Designer Qsys system in Quartus Prime.

```
module homework4top (
    input  wire        CLOCK_50,    // 50 MHz clock
    input  wire [3:0]  KEY,         // Push buttons
    input wire  [9:0]  SW,
    output wire [9:0]  LEDR,        // Red LEDs
    output wire [6:0]  HEX0,        // Seven segment displays
    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,
    // SDRAM interface (connected to physical SDRAM chip)
    output wire [12:0] DRAM_ADDR,
    output wire [1:0]  DRAM_BA,
    output wire        DRAM_CAS_N,
    output wire        DRAM_CKE,
    output wire        DRAM_CS_N,
    inout  wire [15:0] DRAM_DQ,
    output wire        DRAM_LDQM,
    output wire        DRAM_RAS_N,
    output wire        DRAM_UDQM,
    output wire        DRAM_WE_N,
    output wire        DRAM_CLK
);
    wire pll_locked;

    binduhw4 u0 (
        .pll_ref_clk_clk              (CLOCK_50),              // Clock input to PLL
        .pll_ref_reset_reset   (~KEY[0]),                   // Active-low reset

        // I/O
        .keys_external_connection_export         (KEY),
        .leds_external_connection_export      (LEDR),
        .hex0_export          (HEX0),
        .hex1_export          (HEX1),
        .hex2_export          (HEX2),
        .hex3_export          (HEX3),
        .hex4_export          (HEX4),
        .hex5_export          (HEX5),
        .switches_export      (SW),
        // SDRAM connections
        .sdram_wire_addr      (DRAM_ADDR),
        .sdram_wire_ba        (DRAM_BA),
        .sdram_wire_cas_n     (DRAM_CAS_N),
        .sdram_wire_cke       (DRAM_CKE),
        .sdram_wire_cs_n      (DRAM_CS_N),
        .sdram_wire_dq        (DRAM_DQ),
        .sdram_wire_dqm       ({DRAM_UDQM, DRAM_LDQM}),
        .sdram_wire_ras_n     (DRAM_RAS_N),
        .sdram_wire_we_n      (DRAM_WE_N),
        .pll_sdram_clk_clk    (DRAM_CLK)
       // .sdram_pll_locked_export (pll_locked)
    );

endmodule
```

Figure 36: Top-Level Module Declaration in Verilog

The top-level Verilog module is compiled successfully for analysis and Elaboration and shown with 0 errors.

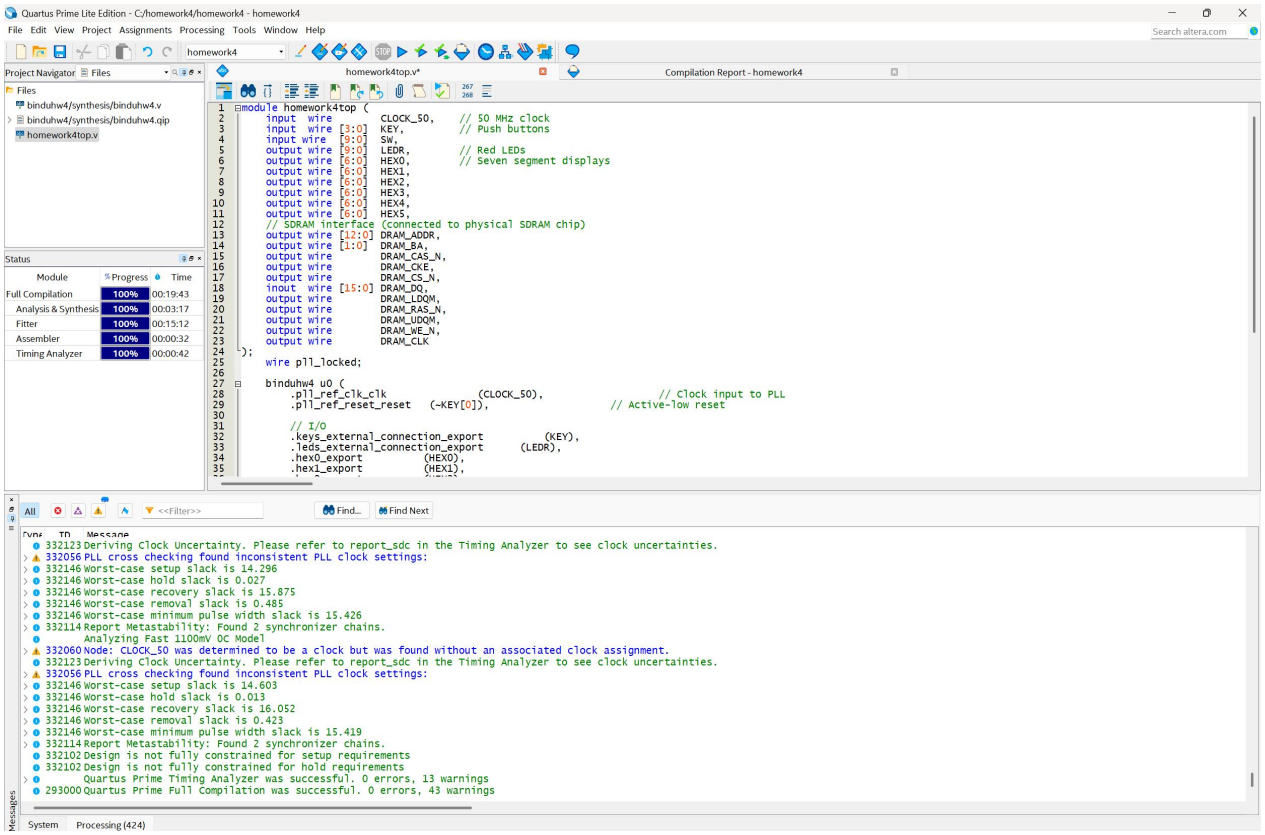


Figure 37: Success compilation for analysis block, no syntax errors

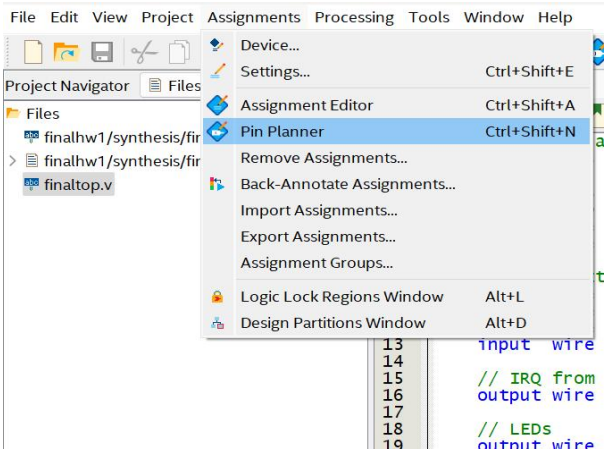


Figure 38: Accessing Pin Planner via Assignments Menu

Shows the Assignments menu with “Pin Planner” selected. This option is used to manually assign FPGA I/O pins corresponding to Verilog top-level ports and Qsys-generated signals.

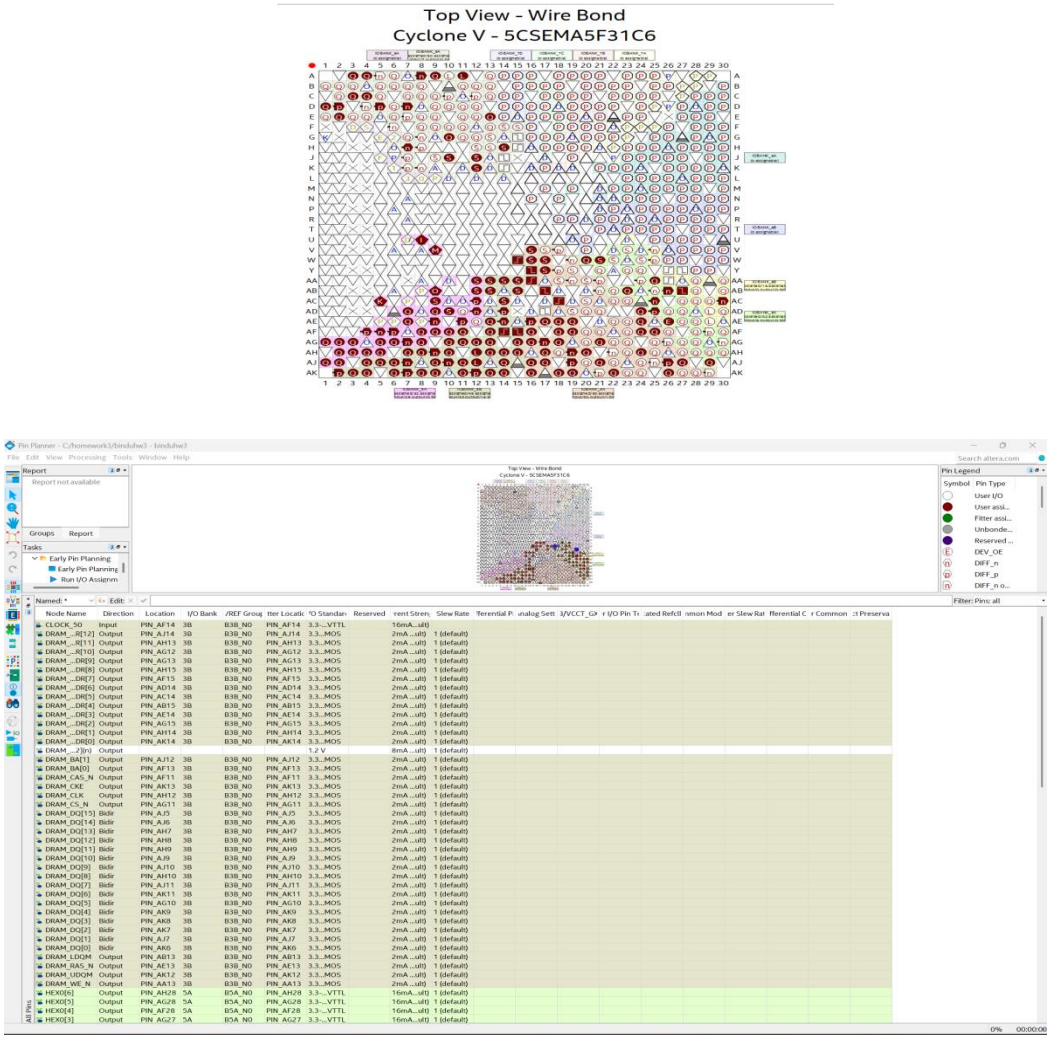


Figure 39: Pin Planner with Assigned I/O Pins

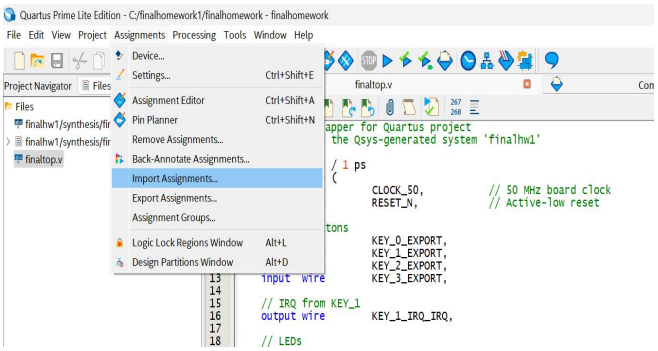


Figure 40: Import Assignments for pins

Also, can assign Pin from the Assignments menu to import assigning pins from pins file.

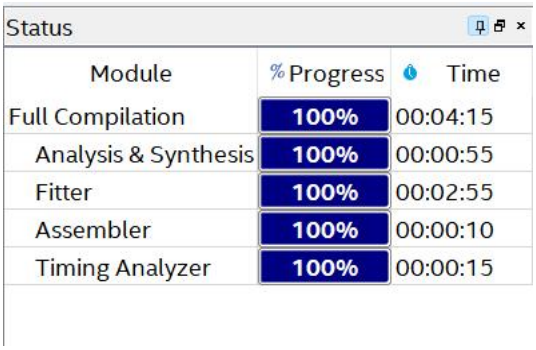


Figure 41: Status panel shows 100%

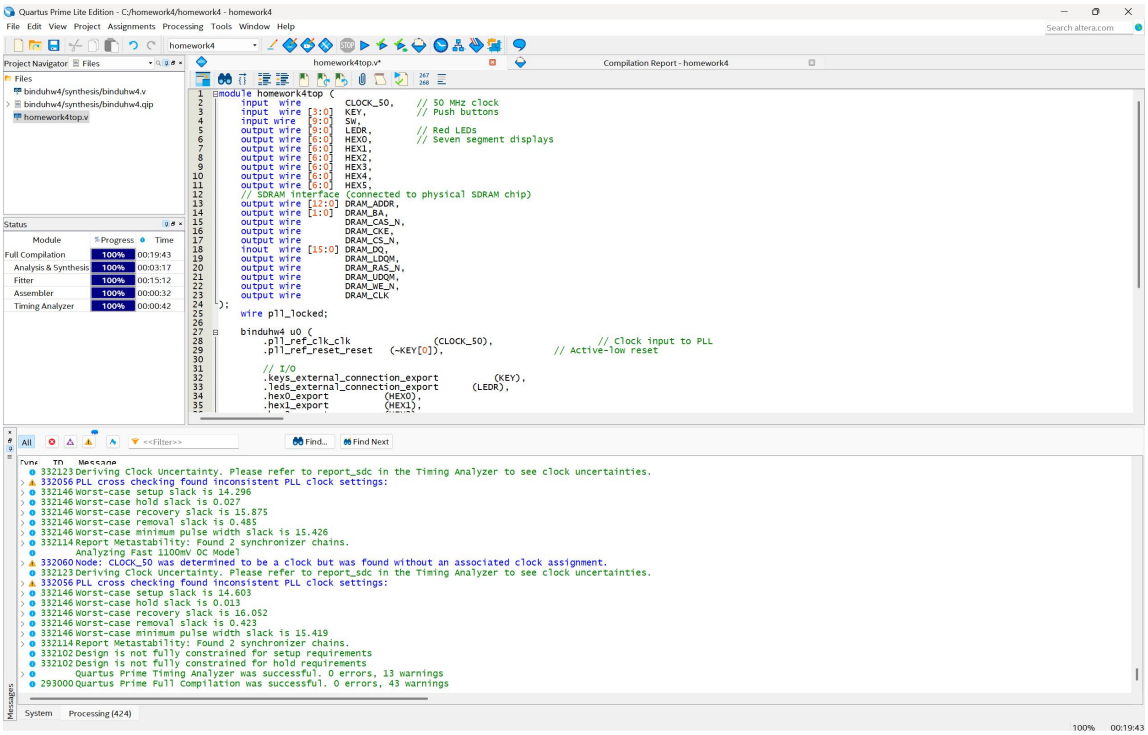


Figure 42: Successful Compilation and Analysis in Quartus Prime

This figure displays the successful compilation report of the project in Quartus Prime Lite. All compilation stages -->Analysis & Synthesis, Fitter, Assembler and Timing Analyzer, all completed with 100% success. The Status panel shows the time taken by each stage, with full compilation. The Message panel confirms that the design was correctly synthesized and fit into the target FPGA, with zero errors and only minor warnings. This validates that the design is fully functional and ready for programming onto the DE1-SoC board.

Once after successful compilation, .sof files got created in the project directory in file manager folder.

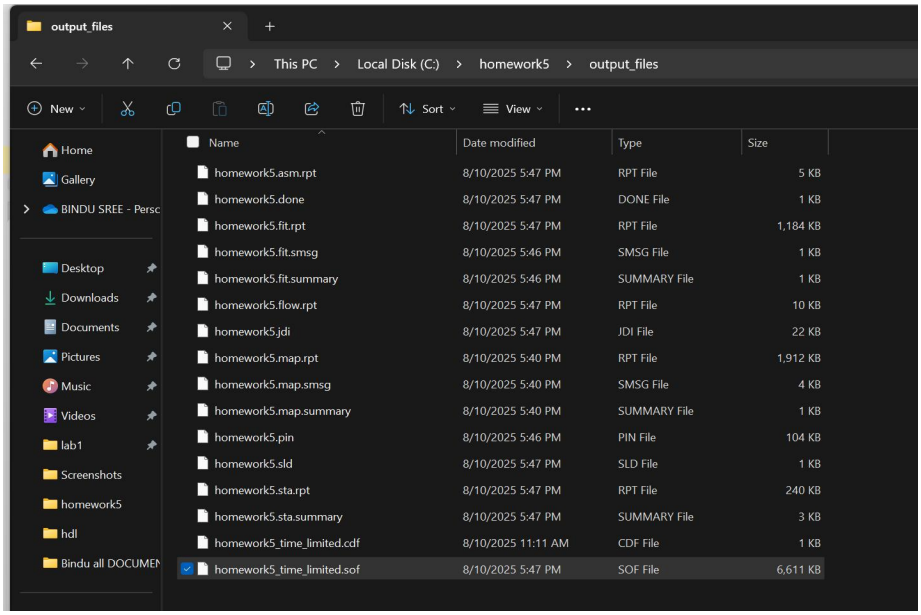


Figure 43: .sof files generated in file manager

6.4 Quartus

Next step is quartus programmer, for launching the Programmer tool in Quartus Prime Lite Edition. From the top menu, the user navigates to Tools > Programmer, which opens the device programming interface required to configure the FPGA. Also, Programmer can be open from Nios II tools.This step is essential to initiate the programming process using the .sof file, allowing the configured logic to be uploaded to the FPGA device on the DE1-SoC board.

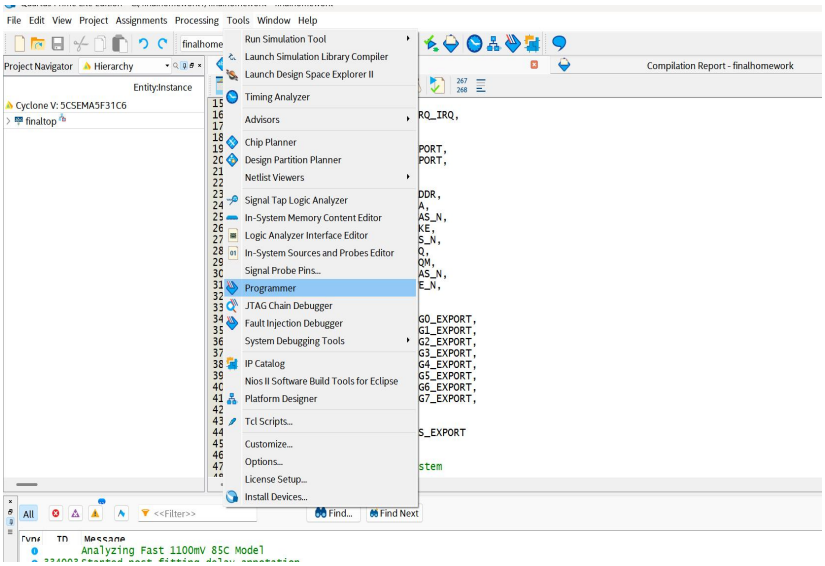


Figure 44: Launching the Programmer Tool in Quartus Prime

This shows the Hardware Setup window in Quartus Prime Programmer Lite Edition. The DE1-SoC board is successfully detected.. The hardware is selected using the dropdown list under “Currently selected hardware.” This step ensures the programmer software communicates correctly with the physical FPGA board via the USB-Blaster interface.

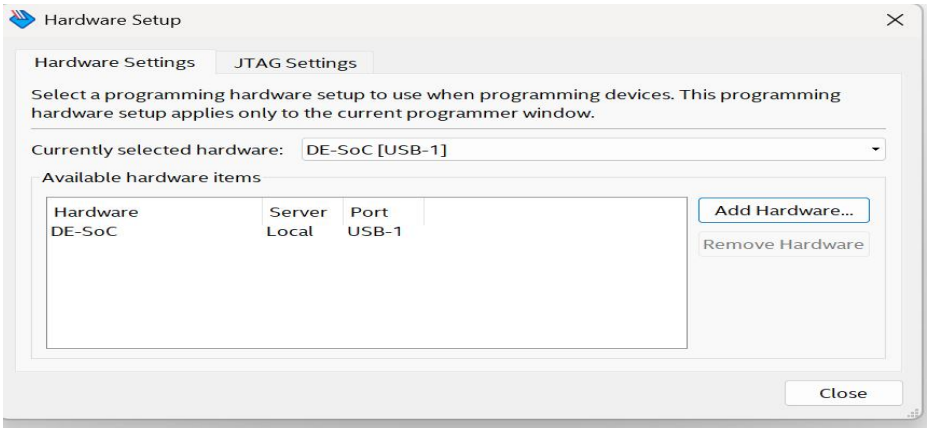


Figure 45: Hardware Setup Selection in Quartus Prime Programmer

Click on Auto detect for FPGA board, opens up dialog box for type device selection. Select as shown in image below and click OK.

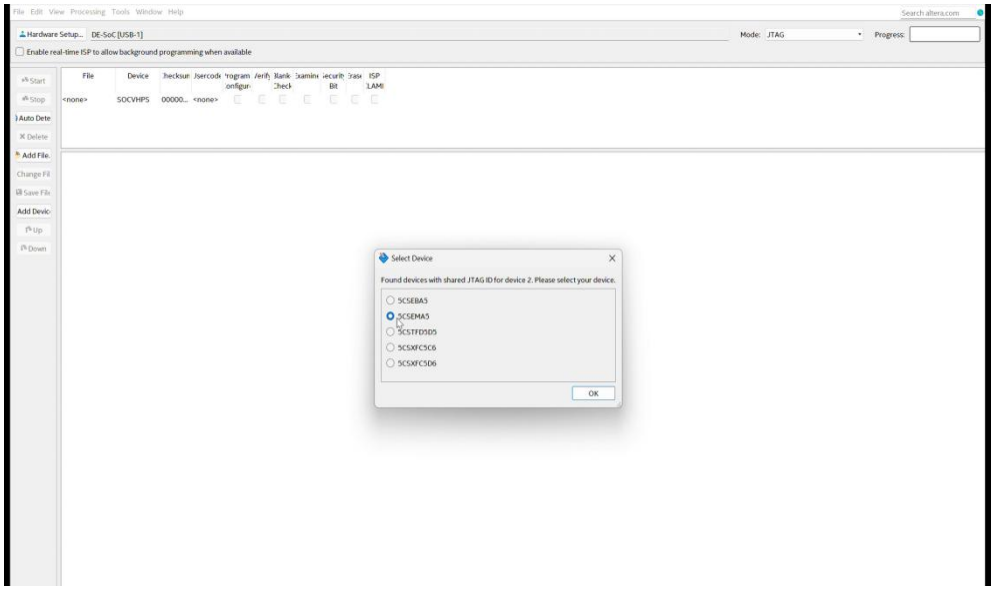


Figure 46: Device selection in Quartus Prime Programmer

After this, Add .sof file into this programmer, by “ADD files” option. And then click on “start”. This figure shows the Quartus Prime Programmer window after the .sof file has been successfully downloaded to the FPGA device. The progress bar at the top right shows "100% (Successful)", indicating that the configuration bitstream was correctly loaded onto the 5CSEMA5F31C6 - FPGA on the DE1-SoC board via the JTAG interface.

At the bottom-left corner of the window, the OpenCore Plus Status dialog box is visible. It confirms that the FPGA is operating in time-limited mode using OpenCore Plus licensing and I kept it open till homework completion.

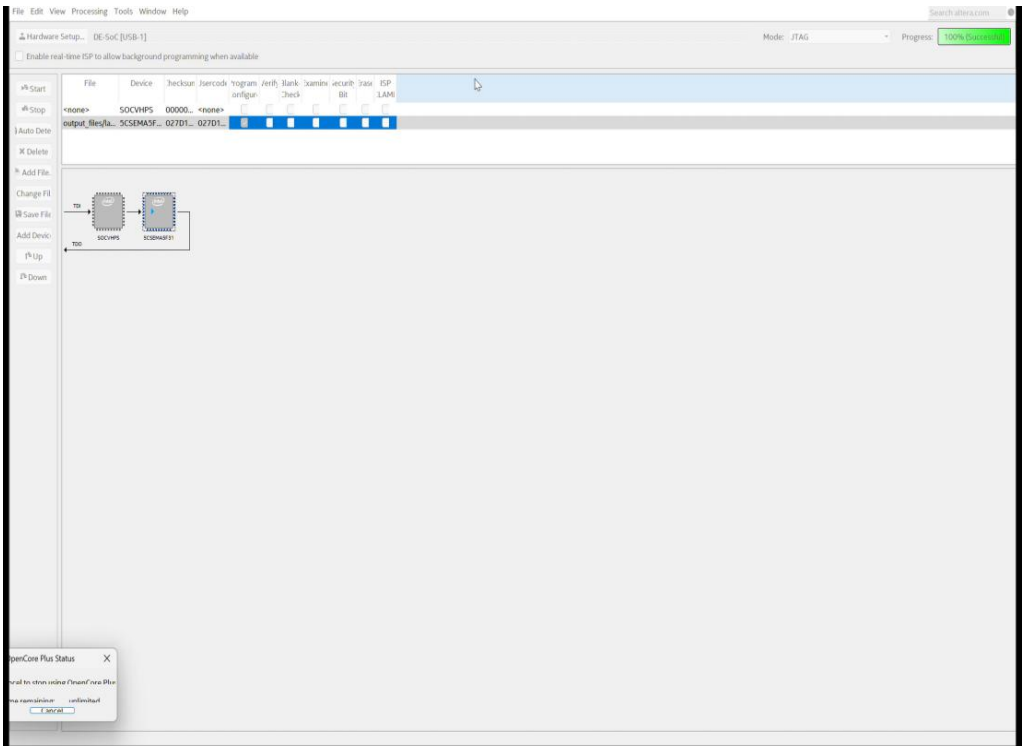


Figure 47: Successful FPGA Programming in Quartus Programmer

After programming the board with .sof file, this is how the board looks like below figure

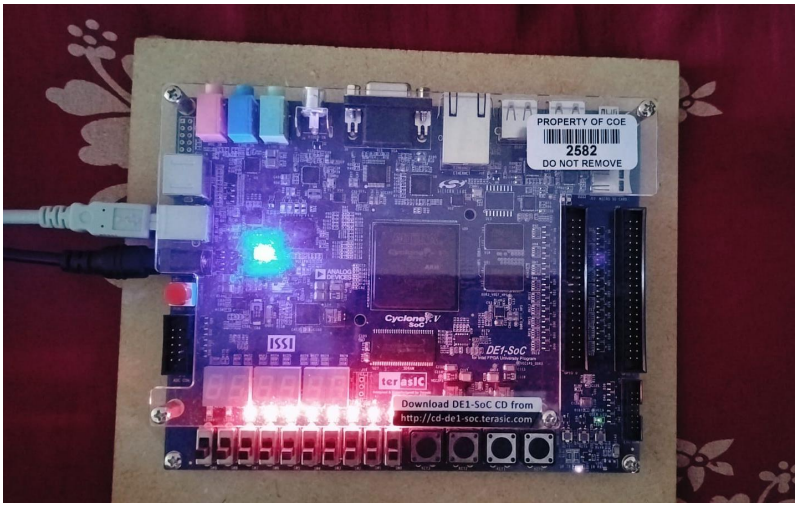


Figure 48: FPGA board after programming .sof file

6.5 Nios II SBT For Eclipse IDE

To write the C code for our custom hardware system (Qsys), we need a proper development environment. So, in this step, I went to Tools > Nios II Software Build Tools for Eclipse from the Quartus Prime window. This launches the Eclipse IDE that is configured for Nios II development. This step is important because it allows us to build and load C applications that will run on the Nios II processor we created in Platform Designer.

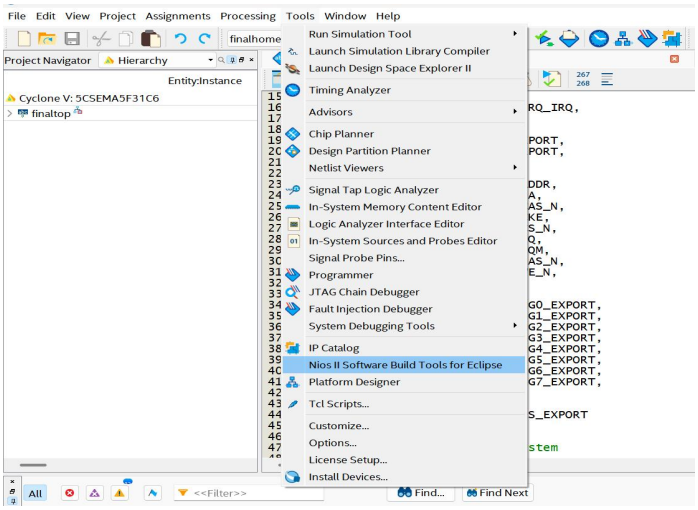


Figure 49: Opening Nios II Software Build Tools for Eclipse

After Eclipse opens, need to create two projects application and bsp projects.

-> **Application project** – where we write our custom code

->Board Support Package (BSP) – which is auto-generated based on the hardware .sopcinfo file we provide while creating.

This step sets up the software, so we can program and test our design on the DE1-SoC board. In this homework, I created application and bsp projects.

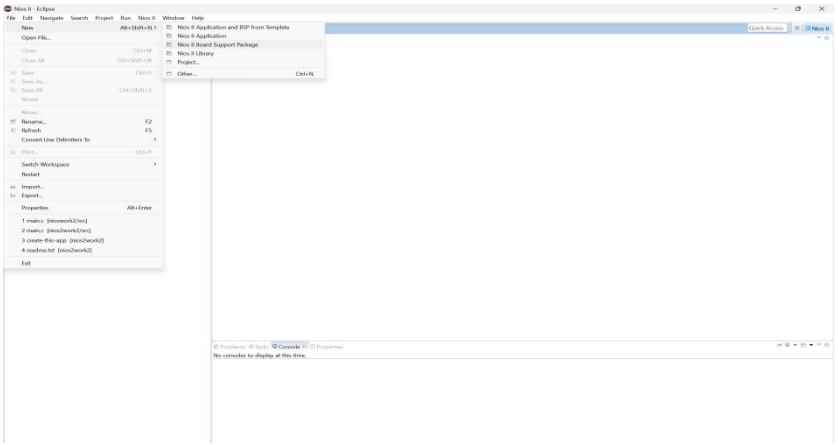


Figure 50: Creating new application project and the BSP

For testing purpose, I created a hello world default project and run the proejcts. I could see everything is working correctly and I could see expected printed in nios console.

Then I moved on to original homework5 implementation.

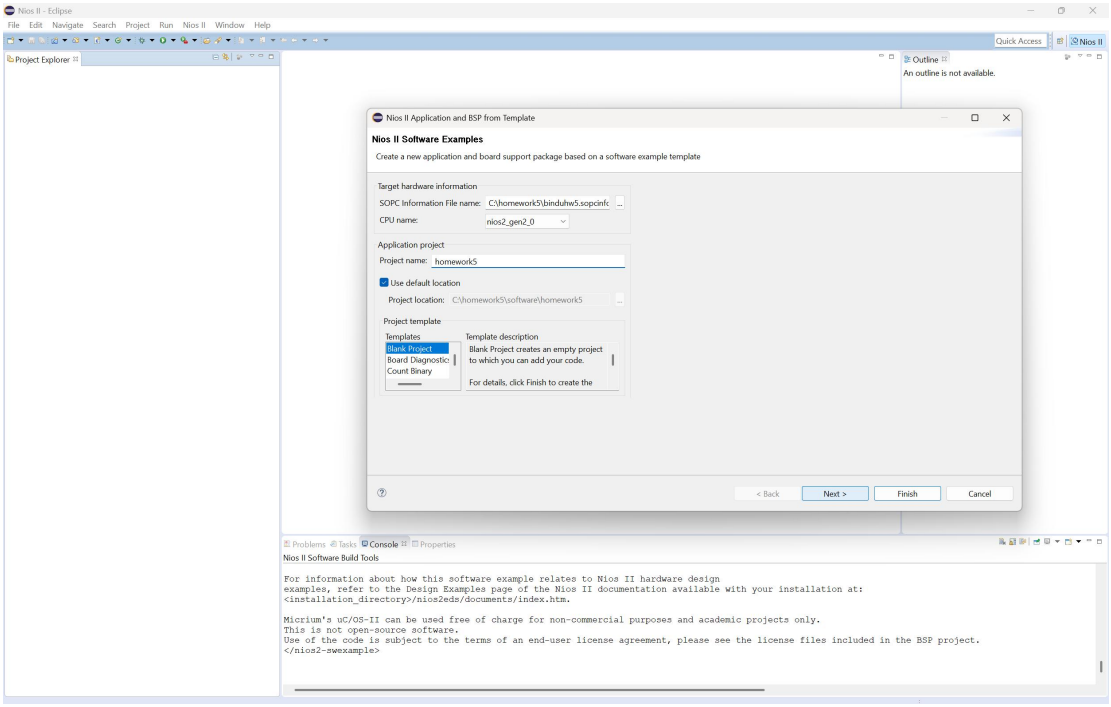


Figure 51: In the Nios II Application and BSP from Template window

Select the .sopcinfo file, enter the project name and chooses a project template such as Blank Project to create the application and board support package (BSP) based on the hardware design.

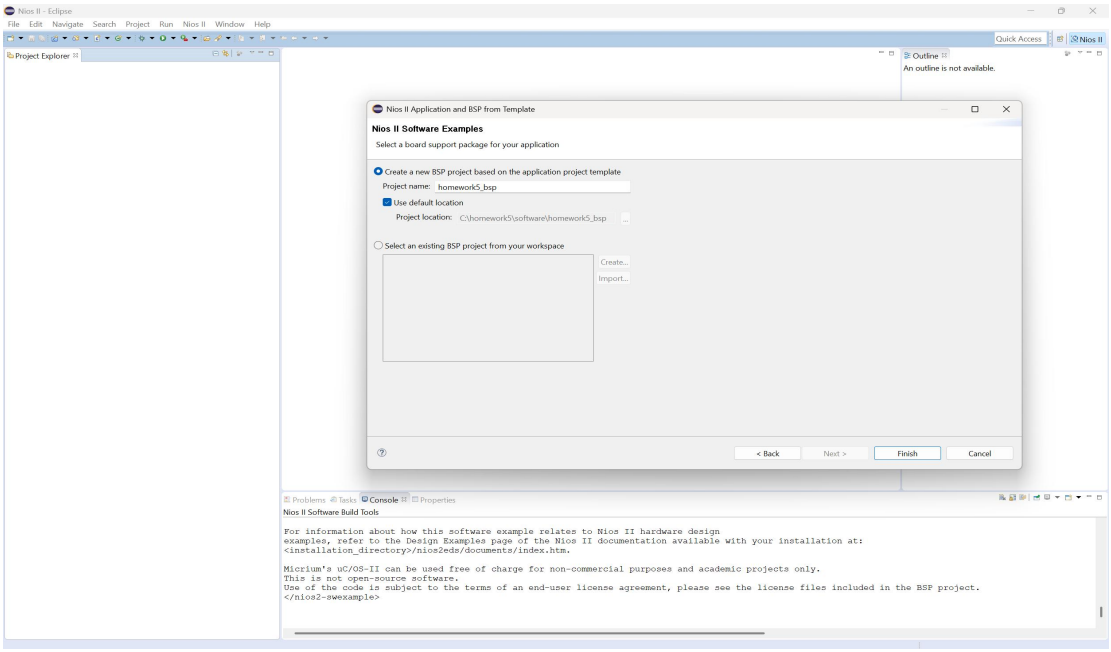
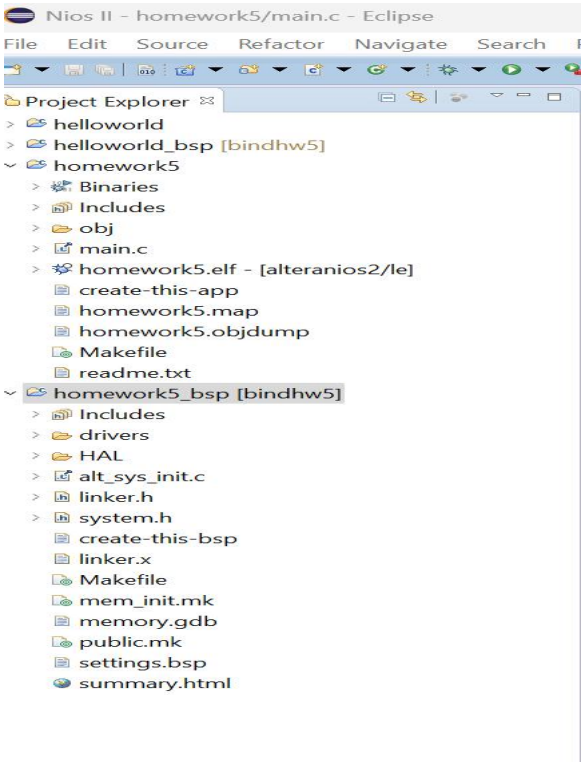


Figure 52: For creating bsp project

In this step, to use an existing BSP project from the workspace instead of creating a new one. This allows reuse of the board support package previously generated for the hardware design. Keep Operating system type as HAL.



After the projects created, add the main.c file which satisfies our requirements.

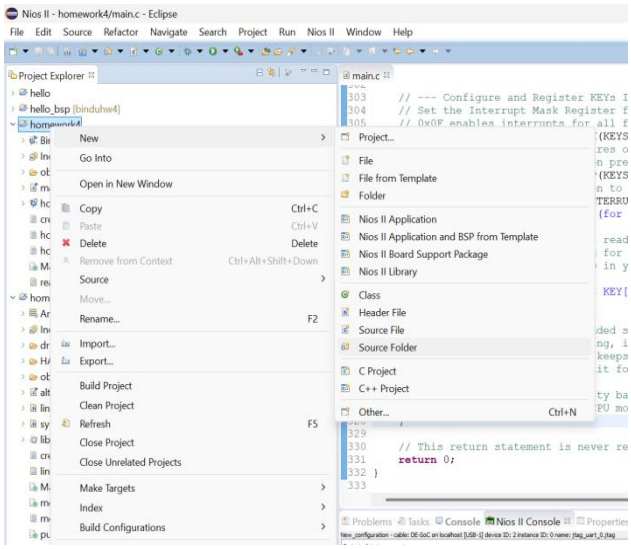


Figure 53: For creating source folder

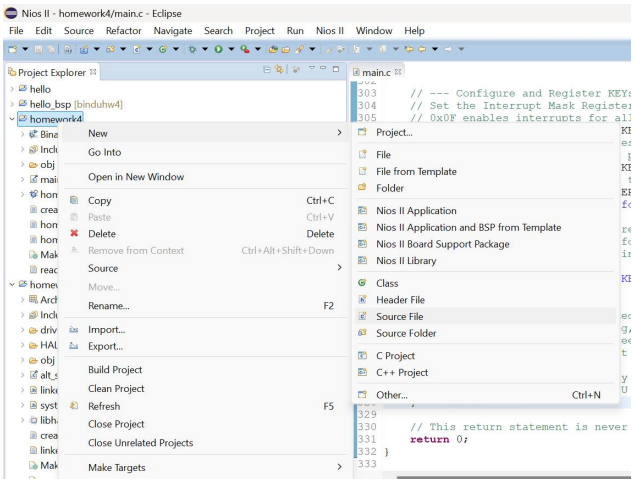


Figure 54: For creating source file

To develop a custom application for the Nios II processor, a new source file must be created within the application project. In this step, right-click the project in the Nios II Eclipse IDE and selects New ->> Source folder to add a new files inside it. This folder contains the custom application logic to be executed on the Nios II processor. Similarly, add main.c codes to all application projects.

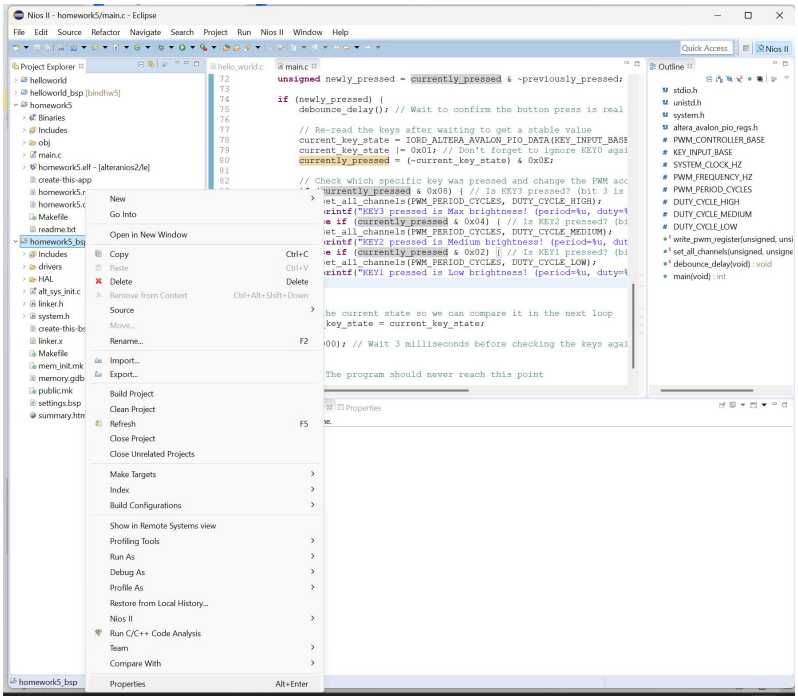


Figure 55: Right-clicking on the bsp project in Eclipse provides access to BSP settings.

This step is used to open the BSP Properties window for configuration. It is essential to ensure the HAL and required drivers are correctly set before building the project.

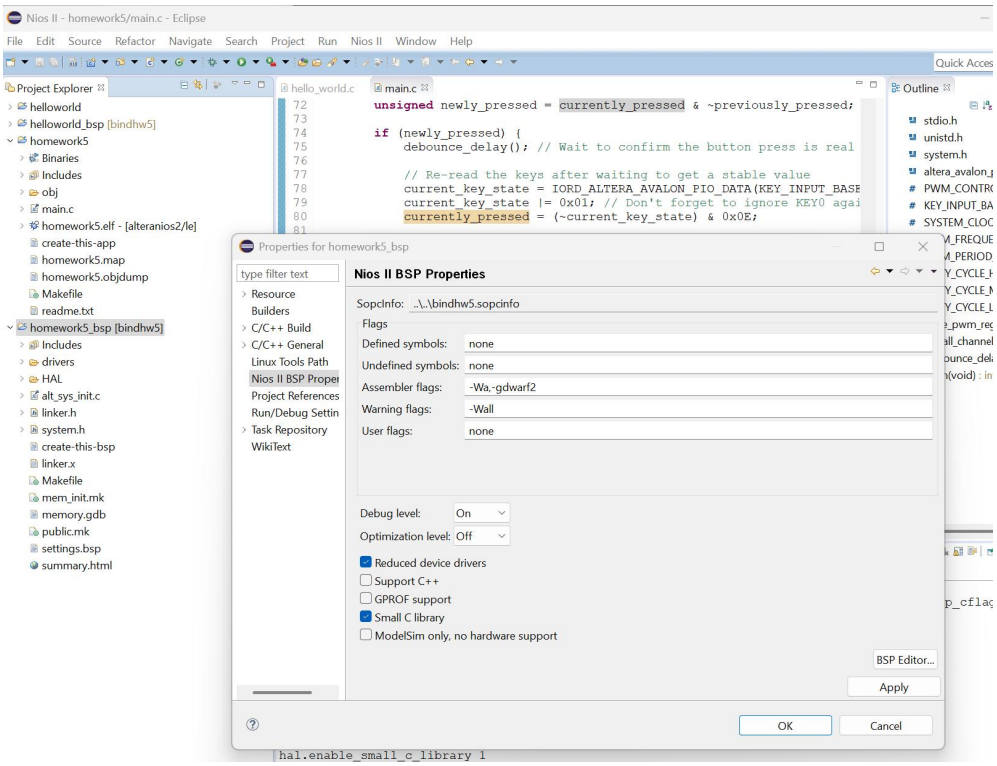


Figure 57: Nios BSP editor

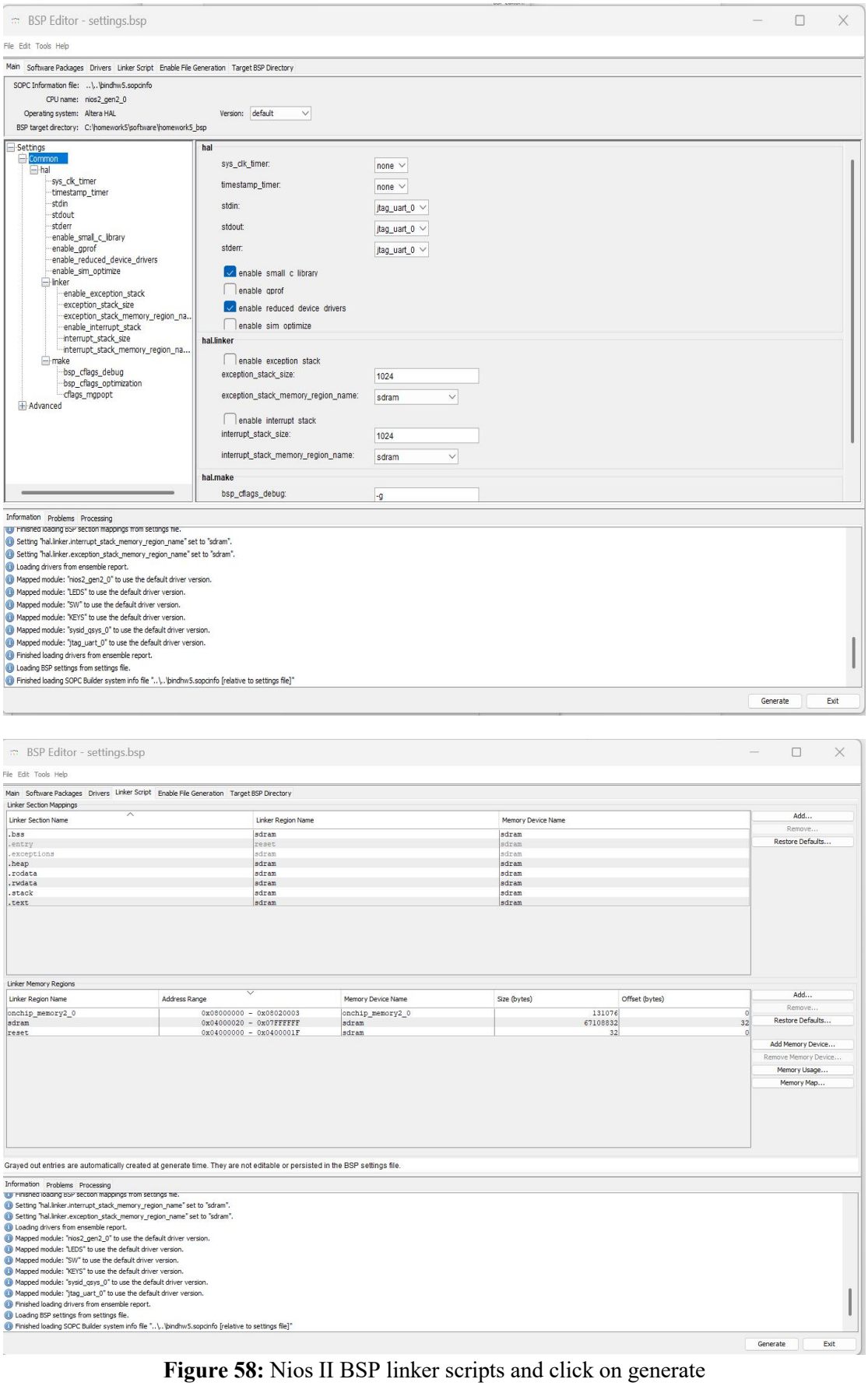


Figure 58: Nios II BSP linker scripts and click on generate

Here, click on BSP editor, then make sure to check the linker scripts and then regenrate the BSP settings and exit. Then apply changes and OK

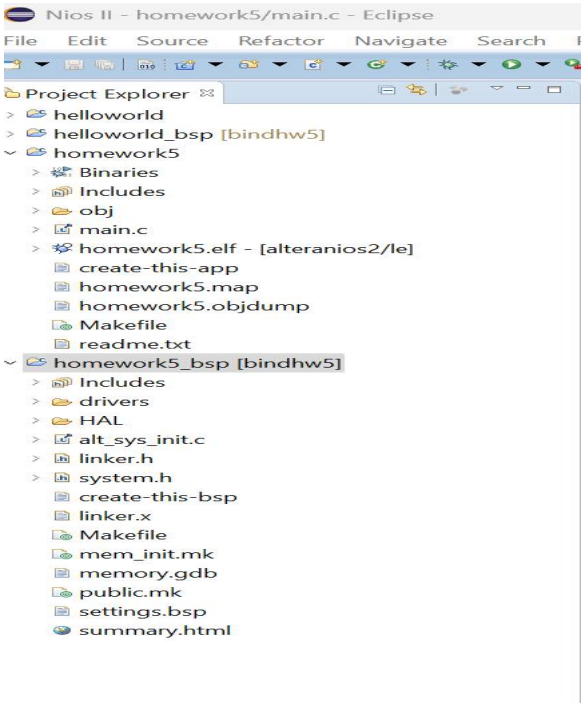
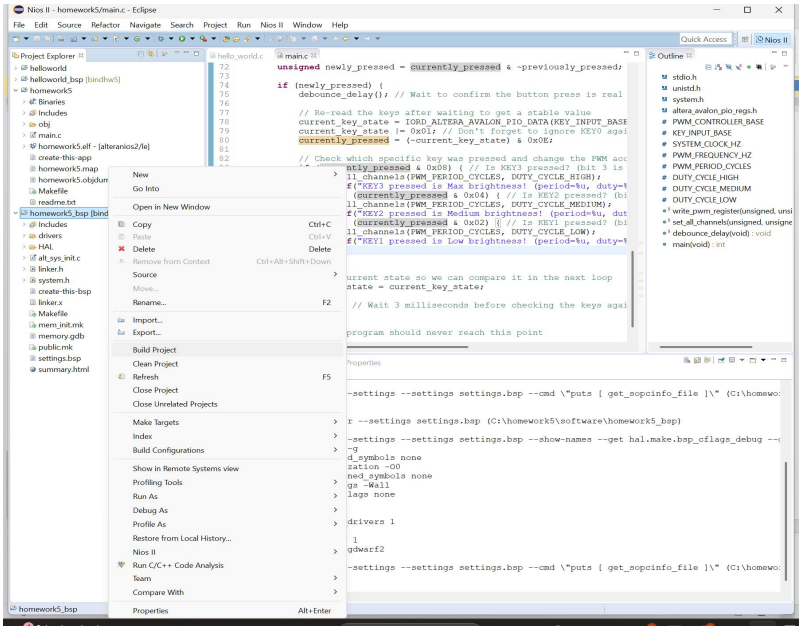


Figure 59: Artifacts of projects in Nios II

Again the next step is to program the FPGA with the sof file using Quartus programmer and it says 100% status.

Now, we need to build and run the projects to see the expected results in the console and on board. For building the projects, either right clicking and selecting build option for each project or project option and then build all works.



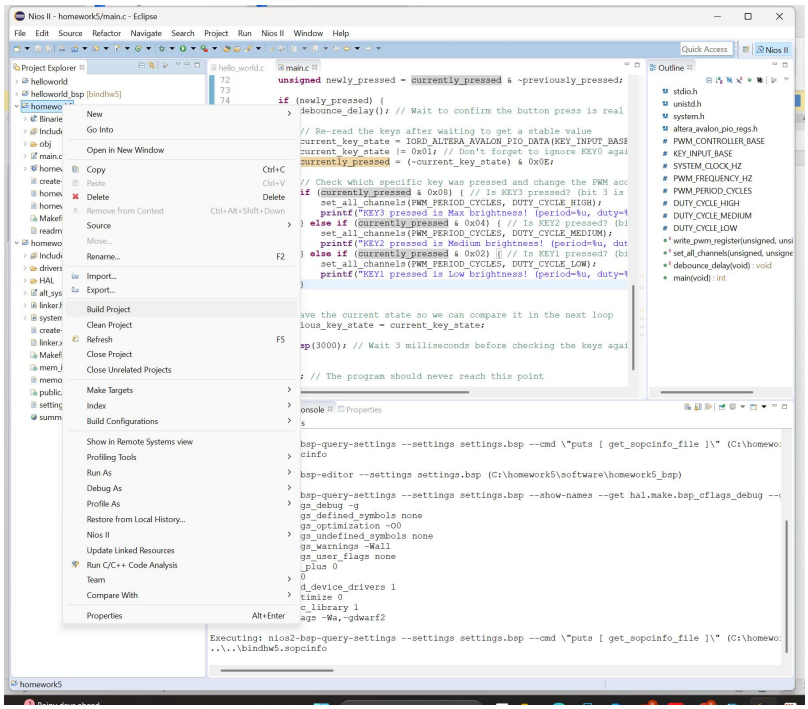


Figure 60: Building both projects

After Building, it says build finished as shown in below figure.

Figure 61: Build complete succesfully for application project

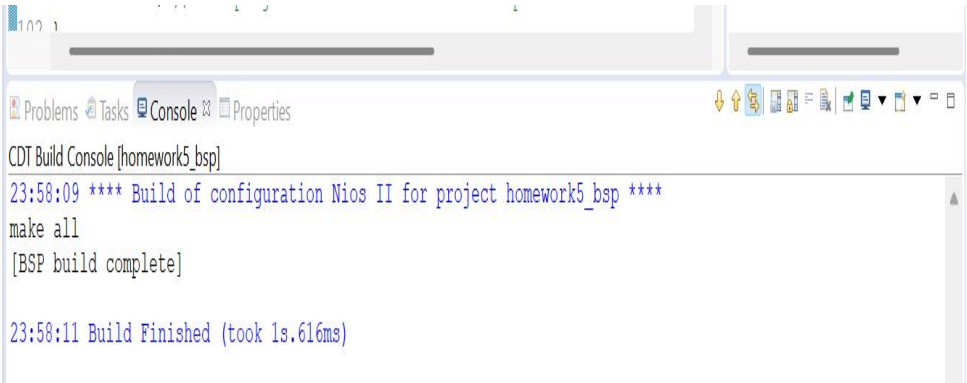


Figure 62: Build complete succesfully for bsp project

For running the projects, make sure to click to on projects, run as --> then click on Configurations. This open up the below configurations window which has all the details.

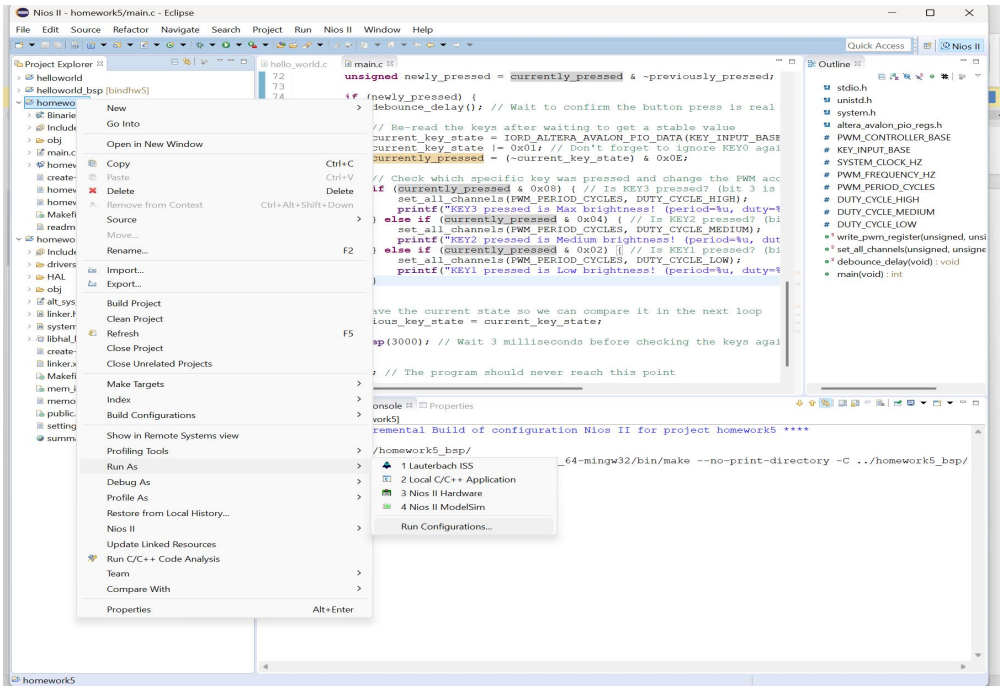


Figure 63: Run options

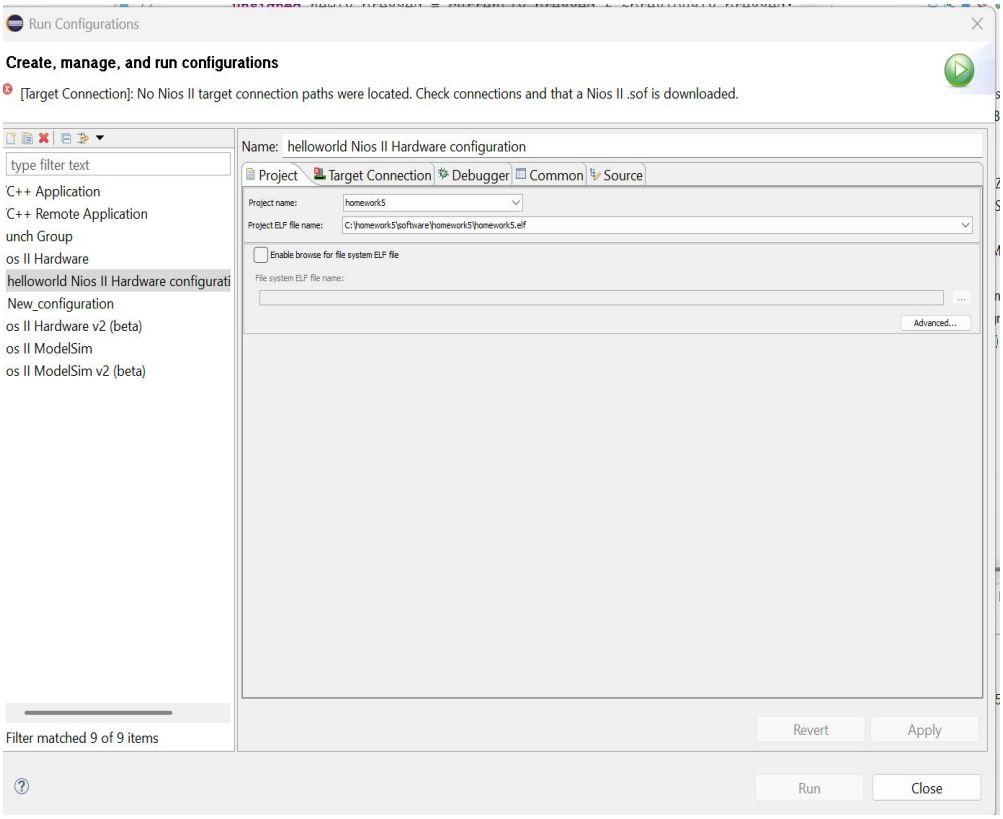


Figure 63: Run Configurations – Project Tab

Refresh the connections in Target connection to see output with Board.

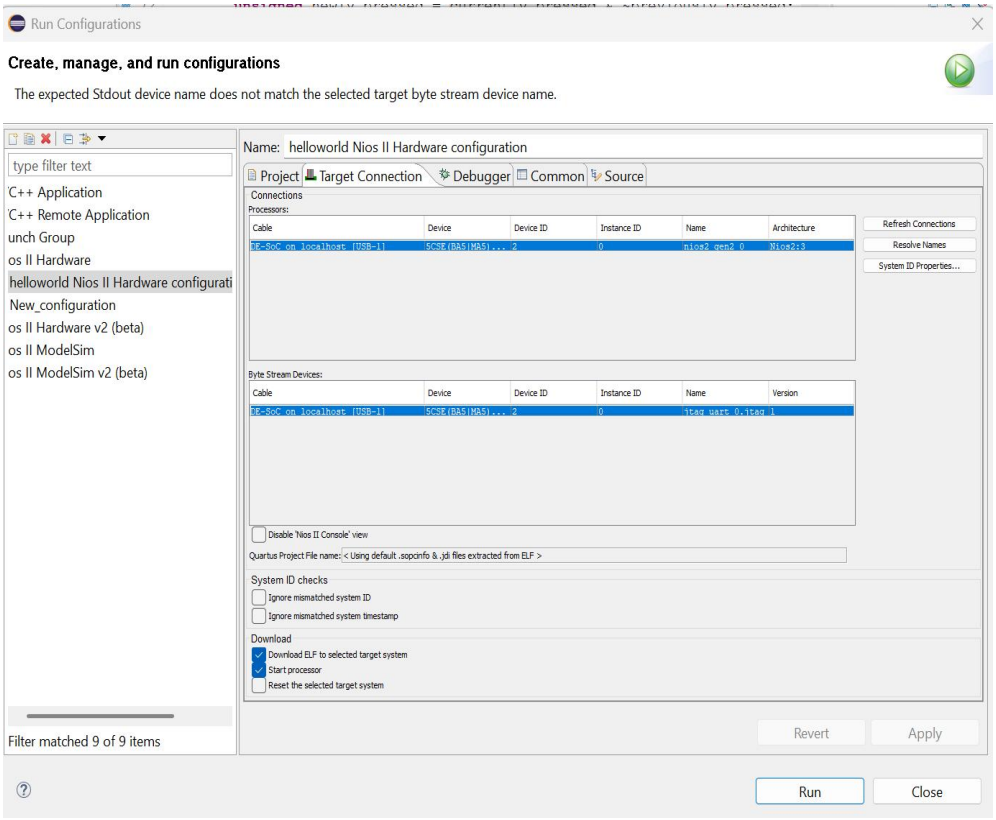


Figure 64: Run Configurations – Target Connection Tab

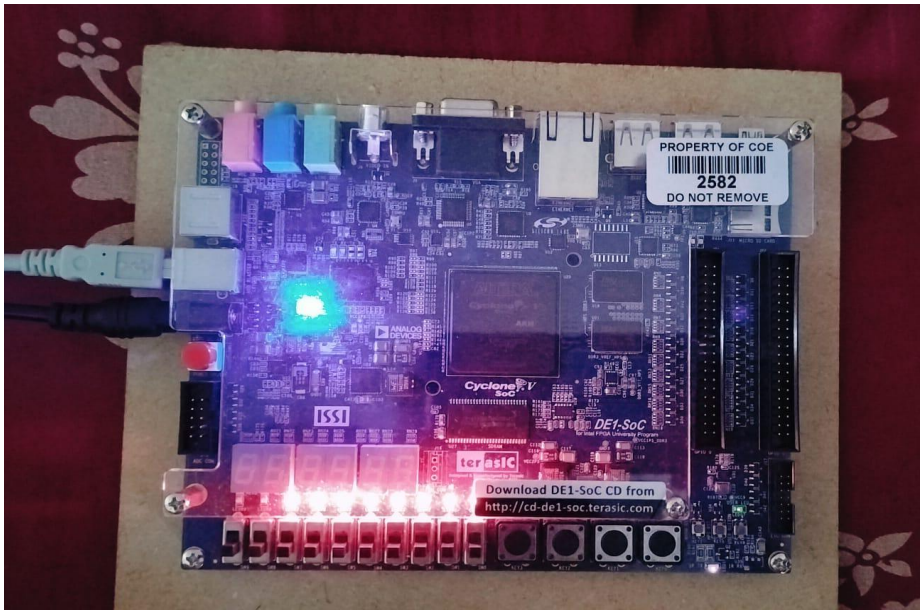


Figure 65: board after programming .sof file and before hitting run in ecllipse

Clicked apply and then run the project. As soon as we run the project, it initializes the game And says user to press sw1 to start game.

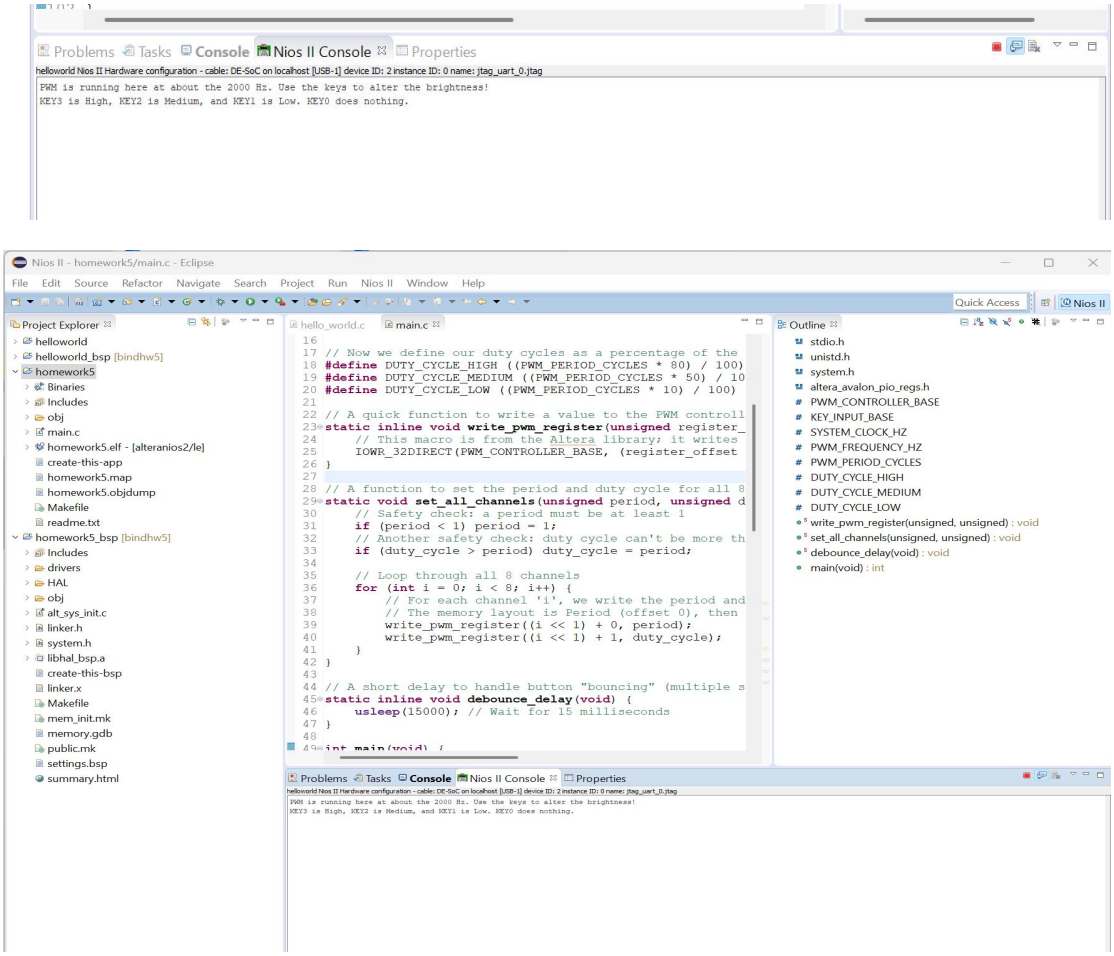


Figure 65: Eclipse IDE – Nios II Console and Source Code

After presing KEY3, low brightness is seen in LEDs on board.

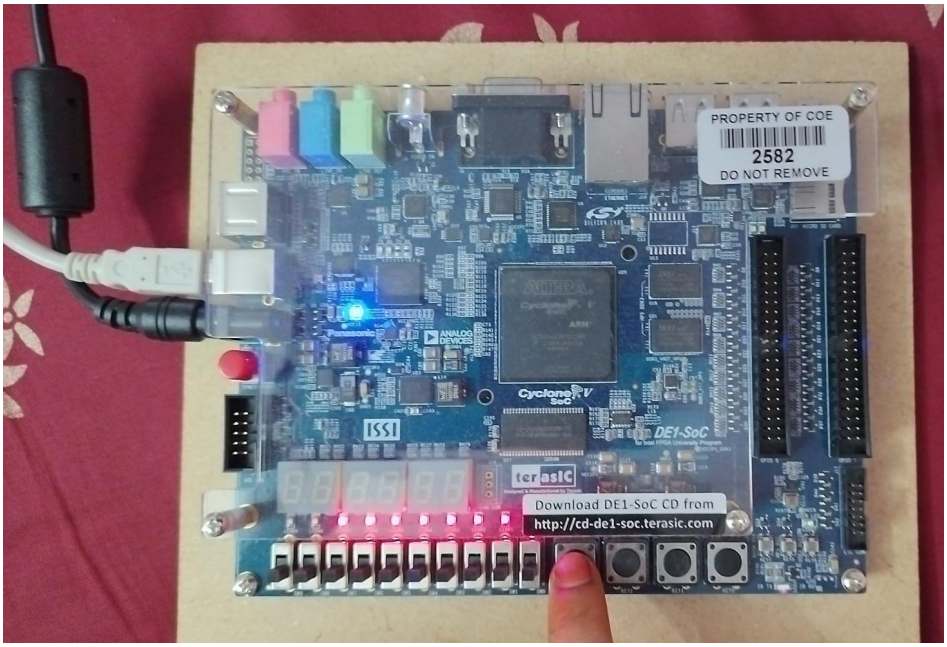


Figure : key 3 is pressed, low brighness

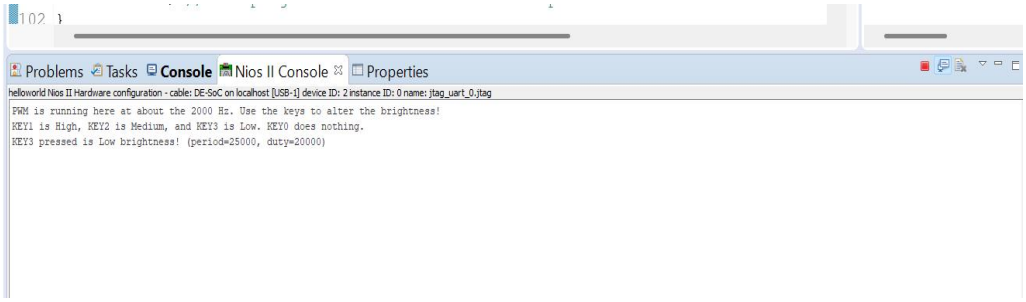


Figure 66: Eclipse IDE – Nios II Console

Pressing key 2, makes LED medium brighter

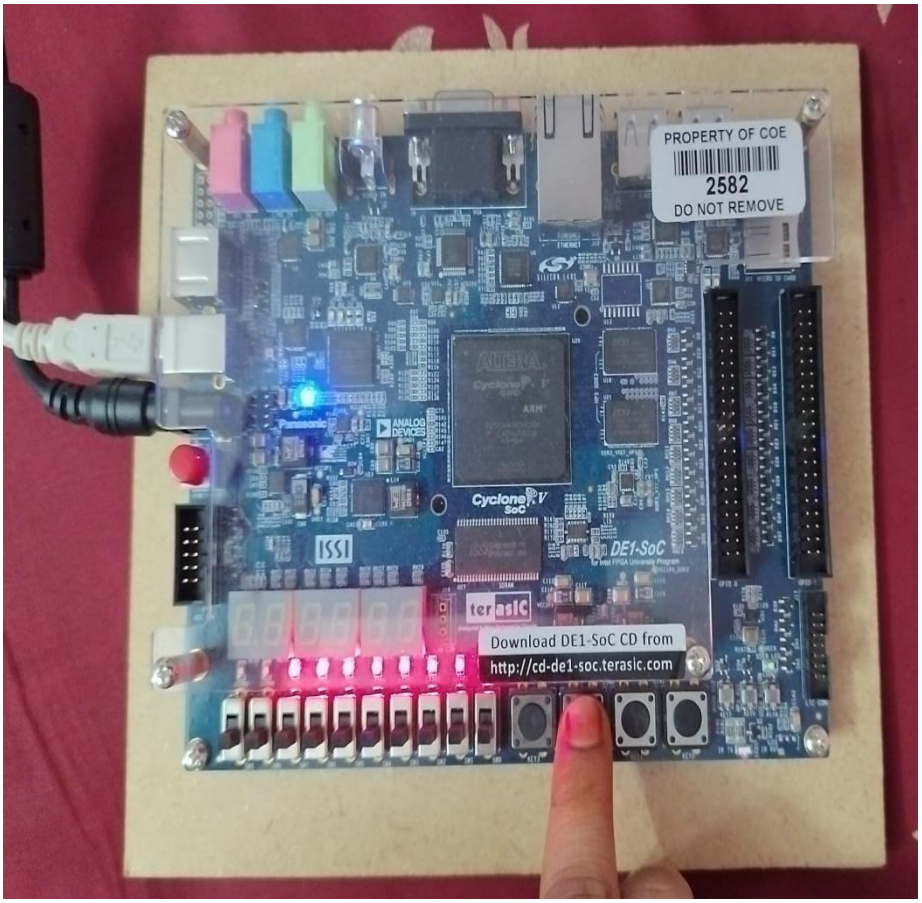


Figure 67: key 2 is pressed

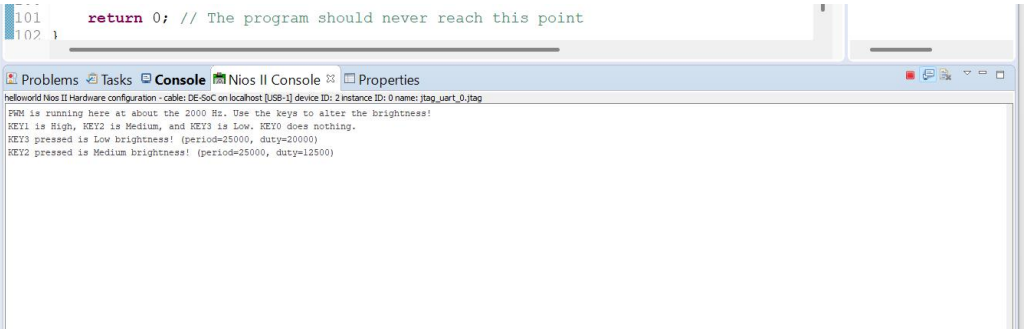


Figure 66: Eclipse IDE – Nios II Console

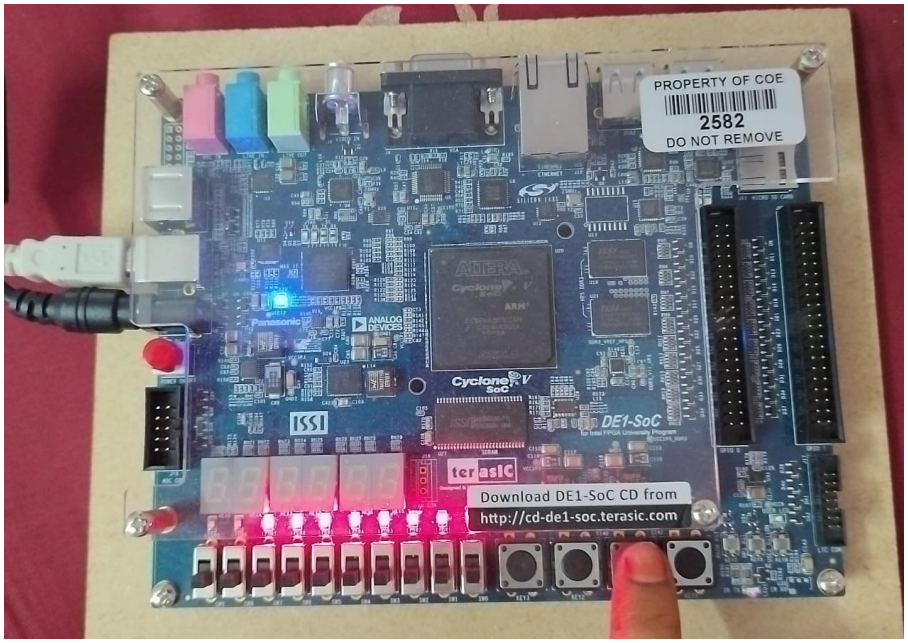


Figure 68 : key 1 is pressed, high brightness

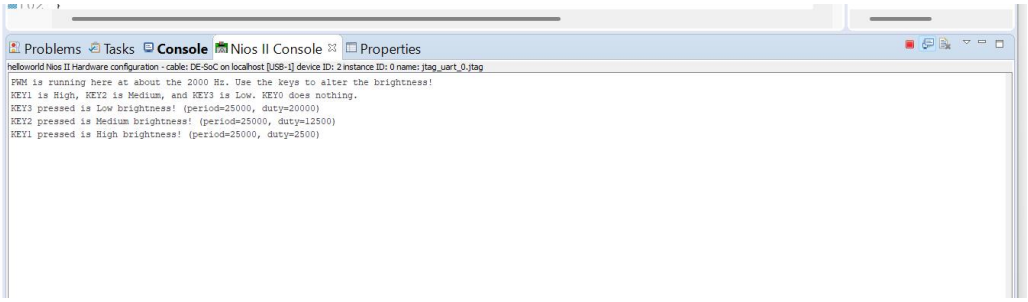


Figure 69: Ecllipse - Nios II IDE

When Key [0] is pressed, it resets and it starts again .

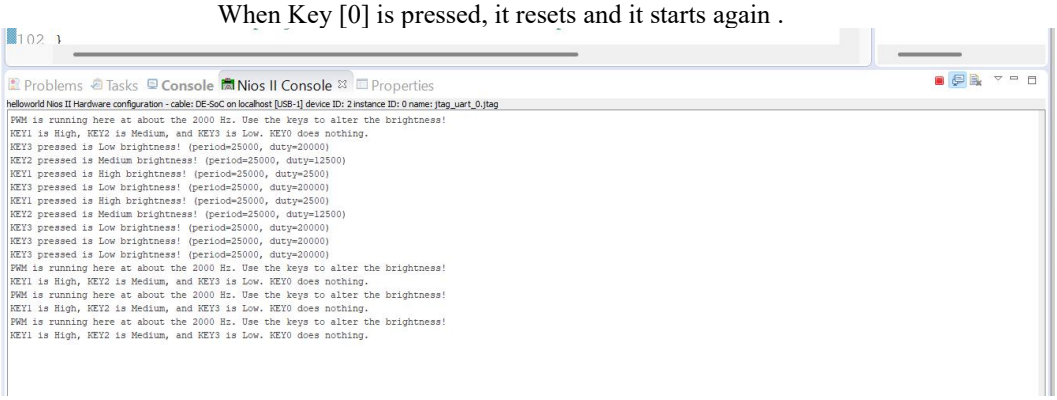


Figure : Key 0 resets the program

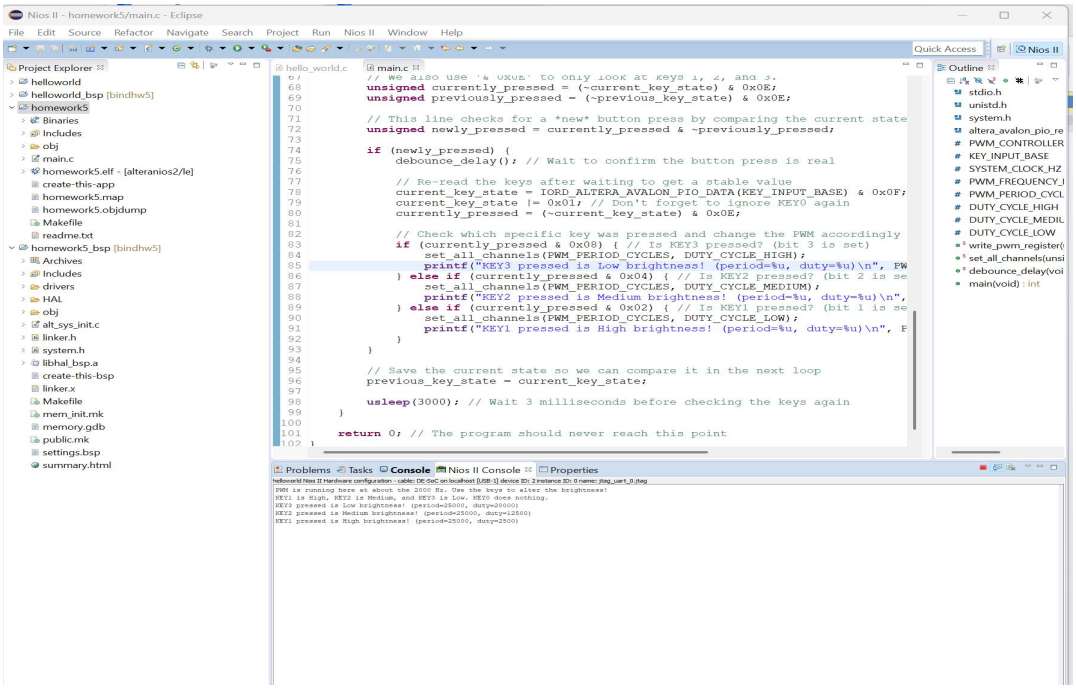


Figure 70: Nios console output

7 Analysis

Purpose:

The main goal for this homework was to design and implement a custom 8-channel Pulse Width Modulation (PWM) system on the Intel DE1-SoC FPGA. This project required me to combine hardware and software design, using Qsys to integrate a custom PWM IP core and then writing the C code for a Nios II processor to control it. The project was about putting theoretical knowledge into practice, using PWM principles to manipulate the brightness of LEDs and using a polling-based approach to handle user input.

What I Learned:

Working on this assignment provided significant hands-on experience with the embedded systems design flow. I gained a deeper understanding of how to create and integrate a custom hardware IP in Verilog, seeing how the Avalon-MM interface enables good communication between the Nios II processor and peripheral's memory-mapped registers. The C code development taught me about polling-based input handling and the necessity of debouncing logic to prevent erroneous inputs from a single button press. Most importantly, I learned how to use the JTAG UART , tools to verify that the software was correctly interacting with the hardware.

Analysis of Output:

The final system functioned exactly as intended. I was happy to see that pressing each of the keys—KEY[1], KEY[2] and KEY[3]—reliably changed the brightness of the red LEDs. The different duty cycle values I implemented successfully produced three

distinct and noticeable levels of light intensity, confirming my understanding of the relationship between duty cycle and brightness. The console output on the JTAG UART was helpful, displaying the current period and duty cycle values, which provided strong confirmation that the correct data was being written to the PWM IP core's registers. This verified that the entire hardware-software data path, from the button press to the final LED output, was working correctly and efficiently.

8 Results

The project was successfully compiled and downloaded onto the DE1-SoC board, with all objectives being met. The custom Verilog PWM IP core correctly generated an 8-channel PWM signal at a frequency of 2 kHz. This was confirmed by the observable variation in LED brightness. The Nios II C code reliably controlled the brightness levels based on user input from the push buttons KEY[3], KEY[2] and KEY[1], with each key producing a distinct level of light intensity. The polling-based logic for button presses, including the debouncing mechanism, worked as intended and the JTAG UART provided valuable feedback that confirmed the system's correct operation.

9 Reflection on the Software Development

What I Learned in homework

I learned that designing a custom hardware system in Qsys is only the first step; the real challenge and satisfaction came from writing the C code. Understanding how to use the Hardware Abstraction Layer (HAL) to control the custom PWM IP and read button inputs was a learning plus. The polling-based approach was a major takeaway, teaching me how a system can continuously check for events and react instantly, especially when contrasted with an interrupt-driven method. Finally, using the JTAG UART console for real-time debugging was valuable, as it allowed me to see inside the program and confirm that every piece of logic, from setting duty cycles to detecting button presses, was functioning exactly as intended.

What Was Challenging to finish work

The most challenging part of this assignment was definitely the debugging and ensuring a seamless connection between the hardware and software. It was one thing to design the custom PWM IP in Verilog and another to correctly link up its base addresses and register offsets in the C code using the IOWR_32DIRECT macro. Implementing a simple software delay within the polling loop was a critical step to debounce the buttons and ensure a single physical press didn't trigger multiple changes to the LED brightness. To say, working through these issues provided a much deeper understanding of how the embedded system interact.

10 Conclusion

The PWM system was successfully implemented and tested on the DE1-SoC board, meeting all the project objectives. The custom Verilog IP core correctly generated the 8-channel PWM signals and the Nios II software provided a reliable control mechanism. The use of a polling-based approach for button handling, along with a debouncing mechanism, ensured the system responded correctly to user input. The project demonstrated a good understanding of hardware software design, from custom IP creation to embedded programming and system integration on an FPGA platform.

11 Appendix

Top level module

```
module homework5top (
    // These are the inputs and outputs from the physical board
    input  wire       CLOCK_50,    // The main 50 MHz clock signal.
    input  wire [3:0]  KEY,        // The four physical push buttons.
    output wire [9:0]  LEDR,       // The ten red LEDs.

    // These are all the connections for the SDRAM memory chip.
    // Our Qsys system needs these to use the memory.
    output wire [12:0] DRAM_ADDR,
    output wire [1:0]  DRAM_BA,
    output wire        DRAM_CAS_N,
    output wire        DRAM_CKE,
    output wire        DRAM_CS_N,
    inout  wire [15:0] DRAM_DQ,
    output wire        DRAM_LDQM,
    output wire        DRAM_RAS_N,
    output wire        DRAM_UDQM,
    output wire        DRAM_WE_N,
    output wire        DRAM_CLK
);
    // This wire will carry the PWM signals from our Qsys system.
        wire [7:0] pwm_custom;

    // We'll use KEY[0] as our reset button. Since the buttons are active-low,
    // we need to make the reset signal high when the button is not pressed.
        wire reset = KEY[0];

    // we add our whole Qsys system (which we called bindhw5) into this top-level design.
    bindhw5 u0 (
        // Connecting the clock and reset to our Qsys system.
        .pll_ref_clk_clk         (CLOCK_50),
        .pll_ref_reset_reset     (~reset),    // Inverting the KEY[0] signal for the reset.

        // Connecting the buttons and PWM output to Qsys.
        .keys_external_connection_export            (KEY),
          .leds_external_connection_export     (LEDR[9:8]), // two leds not needed now
          .pwm_custom_export    (pwm_custom),

        // Passing all the SDRAM connections straight through to our Qsys system's
SDRAM controller.
        .sdram_wire_addr         (DRAM_ADDR),
        .sdram_wire_ba           (DRAM_BA),
        .sdram_wire_cas_n        (DRAM_CAS_N),
        .sdram_wire_cke          (DRAM_CKE),
        .sdram_wire_cs_n         (DRAM_CS_N),
        .sdram_wire_dq           (DRAM_DQ),
        .sdram_wire_dqm          ({DRAM_UDQM, DRAM_LDQM}), // Combining
the upper and lower data masks.
```

```
.sdram_wire_ras_n        (DRAM_RAS_N),
    .sdram_wire_we_n         (DRAM_WE_N),
    .pll_sdram_clk_clk       (DRAM_CLK)
);

        // Finally, we connect the PWM output signals to the first eight LEDs.
        assign LEDR[7:0] = pwm_custom;

endmodule
```

Main.c Code

```
// include the necessary libraries and files.
#include <stdio.h>
#include <unistd.h>
#include "system.h"
#include "altera_avalon_pio_regs.h"

// Here are the addresses for our hardware components.
// We use these names instead of long numbers to make the code easy to read.
#define PWM_CONTROLLER_BASE PWM_CUSTOM_0_BASE
#define KEY_INPUT_BASE KEYS_BASE

// set up important numbers for our PWM signal.
#define SYSTEM_CLOCK_HZ 50000000u // Our board's clock speed is 50 million
cycles per second.
#define PWM_FREQUENCY_HZ 2000u    // We want our LED brightness to change
2,000 times a second.

// This is how many clock cycles make up one PWM period.
#define PWM_PERIOD_CYCLES (SYSTEM_CLOCK_HZ /
PWM_FREQUENCY_HZ)

// These numbers control the brightness. A higher number means the LED is on for
// a longer time in each period, but here,  higher value means dimmer LEDs.
#define DUTY_CYCLE_HIGH ((PWM_PERIOD_CYCLES * 80) / 100) // This is
for low brightness (80% on).
#define DUTY_CYCLE_MEDIUM ((PWM_PERIOD_CYCLES * 50) / 100) // This
is for medium brightness (50% on).
#define DUTY_CYCLE_LOW ((PWM_PERIOD_CYCLES * 10) / 100)  // This is
for high brightness (10% on).

//  function to write a value to our custom PWM hardware.
static inline void write_pwm_register(unsigned register_offset, unsigned value) {
   IOWR_32DIRECT(PWM_CONTROLLER_BASE, (register_offset << 2), value);
}
```

```
// This function sets the brightness for all 8 LEDs at the same time.
static void set_all_channels(unsigned period, unsigned duty_cycle) {
   // We loop through each of the 8 LEDs (channels).
   for (int i = 0; i < 8; i++) {
      // We send the period and duty cycle numbers to our hardware for each LED.
      write_pwm_register((i << 1) + 0, period);
      write_pwm_register((i << 1) + 1, duty_cycle);
   }
}

// A short pause to handle a problem called "button bounce."
// A single button press can sometimes be read as multiple quick presses.
static inline void debounce_delay(void) {
   usleep(15000); // We wait for 15 milliseconds.
}

int main(void) {
   // When the program starts, we set all the LEDs to a medium brightness.
   set_all_channels(PWM_PERIOD_CYCLES, DUTY_CYCLE_MEDIUM);

   printf("Use the keys to change the LED brightness!\n");
   printf("KEY1 is Bright, KEY2 is Medium, and KEY3 is Dim.\n");

   // We store the state of the buttons so we can tell when a new button is pressed.
   unsigned previous_key_state =
IORD_ALTERA_AVALON_PIO_DATA(KEY_INPUT_BASE) & 0x0F;
   previous_key_state |= 0x01;

   // This is our main loop. It runs forever and constantly checks for button presses.
   while (1) {
      // Read the current state of the buttons.
      unsigned current_key_state =
IORD_ALTERA_AVALON_PIO_DATA(KEY_INPUT_BASE) & 0x0F;
      current_key_state |= 0x01;

      // We check to see if a button was just pressed.
      unsigned newly_pressed = ((~current_key_state) & 0x0E) &
~((~previous_key_state) & 0x0E);

      if (newly_pressed) {
         debounce_delay(); // Wait to make sure the press is real.

         // Re-read the buttons to get the final state after the bounce.
         current_key_state =
IORD_ALTERA_AVALON_PIO_DATA(KEY_INPUT_BASE) & 0x0F;
         current_key_state |= 0x01;
         unsigned currently_pressed = (~current_key_state) & 0x0E;
```

```
            // Now, we see which button was pressed and set the brightness.
            if (currently_pressed & 0x08) {  // Is KEY3 pressed?
                set_all_channels(PWM_PERIOD_CYCLES, DUTY_CYCLE_HIGH);
                printf("KEY3 pressed! Low brightness selected.\n");
            } else if (currently_pressed & 0x04) { // Is KEY2 pressed?
                set_all_channels(PWM_PERIOD_CYCLES, DUTY_CYCLE_MEDIUM);
                printf("KEY2 pressed! Medium brightness selected.\n");
            } else if (currently_pressed & 0x02) { // Is KEY1 pressed?
                set_all_channels(PWM_PERIOD_CYCLES, DUTY_CYCLE_LOW);
                printf("KEY1 pressed! High brightness selected.\n");
            }
        }

        // We save the current state for the next time we check.
        previous_key_state = current_key_state;

        usleep(3000); // Wait a tiny time before checking again.
    }

    return 0; // This line is never actually reached.
}
```

Pins assignment file

```
# -------------------------------------------------------------------------- #
#
# Copyright (C) 2018  Intel Corporation. All rights reserved.
# Your use of Intel Corporation's design tools, logic functions
# and other software and tools, and its AMPP partner logic
# functions, and any output files from any of the foregoing
# (including device programming or simulation files), and any
# associated documentation or information are expressly subject
# to the terms and conditions of the Intel Program License
# Subscription Agreement, the Intel Quartus Prime License Agreement,
# the Intel FPGA IP License Agreement, or other applicable license
# agreement, including, without limitation, that your use is for
# the sole purpose of programming logic devices manufactured by
# Intel and sold by Intel or its authorized distributors.  Please
# refer to the applicable agreement for further details.
#
# -------------------------------------------------------------------------- #
#
# Quartus Prime
# Version 18.1.0 Build 625 09/12/2018 SJ Lite Edition
# Date created = 08:43:41  August 08, 2025
#
# -------------------------------------------------------------------------- #
#
# Notes:
#
```

```
# 1) The default values for assignments are stored in the file:
#                homework5_assignment_defaults.qdf
#   If this file doesn't exist, see file:
#                assignment_defaults.qdf
#
# 2) Altera recommends that you do not modify this file. This
#   file is updated automatically by the Quartus Prime software
#   and any changes you make may be lost or overwritten.
#
# ------------------------------------------------------------------------- #


set_global_assignment -name FAMILY "Cyclone V"
set_global_assignment -name DEVICE 5CSEMA5F31C6
set_global_assignment -name TOP_LEVEL_ENTITY homework5top
set_global_assignment -name ORIGINAL_QUARTUS_VERSION 18.1.0
set_global_assignment -name PROJECT_CREATION_TIME_DATE "08:43:41
AUGUST 08, 2025"
set_global_assignment -name LAST_QUARTUS_VERSION "18.1.0 Lite Edition"
set_global_assignment -name PROJECT_OUTPUT_DIRECTORY output_files
set_global_assignment -name MIN_CORE_JUNCTION_TEMP 0
set_global_assignment -name MAX_CORE_JUNCTION_TEMP 85
set_global_assignment -name ERROR_CHECK_FREQUENCY_DIVISOR 256
set_global_assignment -name POWER_PRESET_COOLING_SOLUTION "23 MM
HEAT SINK WITH 200 LFPM AIRFLOW"
set_global_assignment -name POWER_BOARD_THERMAL_MODEL "NONE
(CONSERVATIVE)"
set_global_assignment -name PARTITION_NETLIST_TYPE SOURCE -section_id
Top
set_global_assignment -name PARTITION_FITTER_PRESERVATION_LEVEL
PLACEMENT_AND_ROUTING -section_id Top
set_global_assignment -name PARTITION_COLOR 16764057 -section_id Top


set_location_assignment PIN_AF14 -to CLOCK_50
set_location_assignment PIN_AK14 -to DRAM_ADDR[0]
set_location_assignment PIN_AH14 -to DRAM_ADDR[1]
set_location_assignment PIN_AG15 -to DRAM_ADDR[2]
set_location_assignment PIN_AE14 -to DRAM_ADDR[3]
set_location_assignment PIN_AB15 -to DRAM_ADDR[4]
set_location_assignment PIN_AC14 -to DRAM_ADDR[5]
set_location_assignment PIN_AD14 -to DRAM_ADDR[6]
set_location_assignment PIN_AF15 -to DRAM_ADDR[7]
set_location_assignment PIN_AH15 -to DRAM_ADDR[8]
set_location_assignment PIN_AG13 -to DRAM_ADDR[9]
set_location_assignment PIN_AG12 -to DRAM_ADDR[10]
set_location_assignment PIN_AH13 -to DRAM_ADDR[11]
set_location_assignment PIN_AJ14 -to DRAM_ADDR[12]
set_location_assignment PIN_AF13 -to DRAM_BA[0]
set_location_assignment PIN_AJ12 -to DRAM_BA[1]
```

```
set_location_assignment PIN_AF11 -to DRAM_CAS_N
set_location_assignment PIN_AK13 -to DRAM_CKE
set_location_assignment PIN_AH12 -to DRAM_CLK
set_location_assignment PIN_AG11 -to DRAM_CS_N
set_location_assignment PIN_AK6 -to DRAM_DQ[0]
set_location_assignment PIN_AJ7 -to DRAM_DQ[1]
set_location_assignment PIN_AK7 -to DRAM_DQ[2]
set_location_assignment PIN_AK8 -to DRAM_DQ[3]
set_location_assignment PIN_AK9 -to DRAM_DQ[4]
set_location_assignment PIN_AG10 -to DRAM_DQ[5]
set_location_assignment PIN_AK11 -to DRAM_DQ[6]
set_location_assignment PIN_AJ11 -to DRAM_DQ[7]
set_location_assignment PIN_AH10 -to DRAM_DQ[8]
set_location_assignment PIN_AJ10 -to DRAM_DQ[9]
set_location_assignment PIN_AJ9 -to DRAM_DQ[10]
set_location_assignment PIN_AH9 -to DRAM_DQ[11]
set_location_assignment PIN_AH8 -to DRAM_DQ[12]
set_location_assignment PIN_AH7 -to DRAM_DQ[13]
set_location_assignment PIN_AJ6 -to DRAM_DQ[14]
set_location_assignment PIN_AJ5 -to DRAM_DQ[15]
set_location_assignment PIN_AB13 -to DRAM_LDQM
set_location_assignment PIN_AE13 -to DRAM_RAS_N
set_location_assignment PIN_AK12 -to DRAM_UDQM
set_location_assignment PIN_AA13 -to DRAM_WE_N
set_location_assignment PIN_AE26 -to HEX0[0]
set_location_assignment PIN_AE27 -to HEX0[1]
set_location_assignment PIN_AE28 -to HEX0[2]
set_location_assignment PIN_AG27 -to HEX0[3]
set_location_assignment PIN_AF28 -to HEX0[4]
set_location_assignment PIN_AG28 -to HEX0[5]
set_location_assignment PIN_AH28 -to HEX0[6]
set_location_assignment PIN_AJ29 -to HEX1[0]
set_location_assignment PIN_AH29 -to HEX1[1]
set_location_assignment PIN_AH30 -to HEX1[2]
set_location_assignment PIN_AG30 -to HEX1[3]
set_location_assignment PIN_AF29 -to HEX1[4]
set_location_assignment PIN_AF30 -to HEX1[5]
set_location_assignment PIN_AD27 -to HEX1[6]
set_location_assignment PIN_AB23 -to HEX2[0]
set_location_assignment PIN_AE29 -to HEX2[1]
set_location_assignment PIN_AD29 -to HEX2[2]
set_location_assignment PIN_AC28 -to HEX2[3]
set_location_assignment PIN_AD30 -to HEX2[4]
set_location_assignment PIN_AC29 -to HEX2[5]
set_location_assignment PIN_AC30 -to HEX2[6]
set_location_assignment PIN_AD26 -to HEX3[0]
set_location_assignment PIN_AC27 -to HEX3[1]
set_location_assignment PIN_AD25 -to HEX3[2]
set_location_assignment PIN_AC25 -to HEX3[3]
set_location_assignment PIN_AB28 -to HEX3[4]
```

```
set_location_assignment PIN_AB25 -to HEX3[5]
set_location_assignment PIN_AB22 -to HEX3[6]
set_location_assignment PIN_AA24 -to HEX4[0]
set_location_assignment PIN_Y23 -to HEX4[1]
set_location_assignment PIN_Y24 -to HEX4[2]
set_location_assignment PIN_W22 -to HEX4[3]
set_location_assignment PIN_W24 -to HEX4[4]
set_location_assignment PIN_V23 -to HEX4[5]
set_location_assignment PIN_W25 -to HEX4[6]
set_location_assignment PIN_V25 -to HEX5[0]
set_location_assignment PIN_AA28 -to HEX5[1]
set_location_assignment PIN_Y27 -to HEX5[2]
set_location_assignment PIN_AB27 -to HEX5[3]
set_location_assignment PIN_AB26 -to HEX5[4]
set_location_assignment PIN_AA26 -to HEX5[5]
set_location_assignment PIN_AA25 -to HEX5[6]
set_location_assignment PIN_V16 -to LEDR[0]
set_location_assignment PIN_W16 -to LEDR[1]
set_location_assignment PIN_V17 -to LEDR[2]
set_location_assignment PIN_V18 -to LEDR[3]
set_location_assignment PIN_W17 -to LEDR[4]
set_location_assignment PIN_W19 -to LEDR[5]
set_location_assignment PIN_Y19 -to LEDR[6]
set_location_assignment PIN_W20 -to LEDR[7]
set_location_assignment PIN_W21 -to LEDR[8]
set_location_assignment PIN_Y21 -to LEDR[9]
set_instance_assignment -name IO_STANDARD "3.3-V LVCMOS" -to
DRAM_ADDR[12]
set_instance_assignment -name IO_STANDARD "1.2 V" -to
"DRAM_ADDR[12](n)"
set_instance_assignment -name IO_STANDARD "3.3-V LVCMOS" -to
DRAM_ADDR[11]
set_instance_assignment -name IO_STANDARD "3.3-V LVCMOS" -to
DRAM_ADDR[10]
set_instance_assignment -name IO_STANDARD "3.3-V LVCMOS" -to
DRAM_ADDR[9]
set_instance_assignment -name IO_STANDARD "3.3-V LVCMOS" -to
DRAM_ADDR[8]
set_instance_assignment -name IO_STANDARD "3.3-V LVCMOS" -to
DRAM_ADDR[7]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HEX0[6]
set_instance_assignment -name IO_STANDARD "3.3-V LVCMOS" -to
DRAM_ADDR[6]
set_instance_assignment -name IO_STANDARD "3.3-V LVCMOS" -to
DRAM_ADDR[5]
set_instance_assignment -name IO_STANDARD "3.3-V LVCMOS" -to
DRAM_ADDR[4]
set_instance_assignment -name IO_STANDARD "3.3-V LVCMOS" -to
DRAM_ADDR[3]
```

```
set_instance_assignment -name IO_STANDARD "3.3-V LVCMOS" -to
DRAM_ADDR[2]
set_instance_assignment -name IO_STANDARD "3.3-V LVCMOS" -to
DRAM_ADDR[1]
set_instance_assignment -name IO_STANDARD "3.3-V LVCMOS" -to
DRAM_BA[1]
set_instance_assignment -name IO_STANDARD "3.3-V LVCMOS" -to
DRAM_BA[0]
set_instance_assignment -name IO_STANDARD "3.3-V LVCMOS" -to DRAM_BA
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HEX0[5]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HEX0[4]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HEX0[3]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HEX0[2]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HEX0[1]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HEX0[0]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HEX1[6]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HEX1[5]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HEX1[4]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HEX1[3]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HEX1[2]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HEX1[1]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HEX1[0]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HEX2[6]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HEX2[5]
set_instance_assignment -name IO_STANDARD "3.3-V LVCMOS" -to
DRAM_CAS_N
set_instance_assignment -name IO_STANDARD "3.3-V LVCMOS" -to
DRAM_CKE
set_instance_assignment -name IO_STANDARD "3.3-V LVCMOS" -to
DRAM_CLK
set_instance_assignment -name IO_STANDARD "3.3-V LVCMOS" -to
DRAM_CS_N
set_instance_assignment -name IO_STANDARD "3.3-V LVCMOS" -to
DRAM_WE_N
set_instance_assignment -name IO_STANDARD "3.3-V LVCMOS" -to
DRAM_UDQM
set_instance_assignment -name IO_STANDARD "3.3-V LVCMOS" -to
DRAM_RAS_N
set_instance_assignment -name IO_STANDARD "3.3-V LVCMOS" -to
DRAM_LDQM
set_instance_assignment -name IO_STANDARD "3.3-V LVCMOS" -to
DRAM_DQ[0]
set_instance_assignment -name IO_STANDARD "3.3-V LVCMOS" -to
DRAM_DQ[1]
set_instance_assignment -name IO_STANDARD "3.3-V LVCMOS" -to
DRAM_DQ[2]
set_instance_assignment -name IO_STANDARD "3.3-V LVCMOS" -to
DRAM_DQ[3]
set_instance_assignment -name IO_STANDARD "3.3-V LVCMOS" -to
DRAM_DQ[4]
```

```
set_instance_assignment -name IO_STANDARD "3.3-V LVCMOS" -to
DRAM_DQ[5]
set_instance_assignment -name IO_STANDARD "3.3-V LVCMOS" -to
DRAM_DQ[6]
set_instance_assignment -name IO_STANDARD "3.3-V LVCMOS" -to
DRAM_DQ[7]
set_instance_assignment -name IO_STANDARD "3.3-V LVCMOS" -to
DRAM_DQ[8]
set_instance_assignment -name IO_STANDARD "3.3-V LVCMOS" -to
DRAM_DQ[9]
set_instance_assignment -name IO_STANDARD "3.3-V LVCMOS" -to
DRAM_DQ[10]
set_instance_assignment -name IO_STANDARD "3.3-V LVCMOS" -to
DRAM_DQ[11]
set_instance_assignment -name IO_STANDARD "3.3-V LVCMOS" -to
DRAM_DQ[12]
set_instance_assignment -name IO_STANDARD "3.3-V LVCMOS" -to
DRAM_DQ[13]
set_instance_assignment -name IO_STANDARD "3.3-V LVCMOS" -to
DRAM_DQ[14]
set_instance_assignment -name IO_STANDARD "3.3-V LVCMOS" -to
DRAM_DQ[15]
set_instance_assignment -name IO_STANDARD "3.3-V LVCMOS" -to LEDR[0]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to KEY[3]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to KEY[2]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to KEY[1]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to KEY[0]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to KEY
set_instance_assignment -name IO_STANDARD "3.3-V LVCMOS" -to DRAM_DQ
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HEX2[4]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HEX2[3]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HEX2[2]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HEX2[1]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HEX2[0]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HEX3[6]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HEX3[5]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HEX3[4]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HEX3[3]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HEX3[2]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HEX3[1]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HEX3[0]
set_instance_assignment -name IO_STANDARD "3.3-V LVCMOS" -to LEDR[9]
set_instance_assignment -name IO_STANDARD "3.3-V LVCMOS" -to LEDR[8]
set_instance_assignment -name IO_STANDARD "3.3-V LVCMOS" -to LEDR[7]
set_instance_assignment -name IO_STANDARD "3.3-V LVCMOS" -to LEDR[6]
set_instance_assignment -name IO_STANDARD "3.3-V LVCMOS" -to LEDR[5]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HEX4[0]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HEX5[6]
set_instance_assignment -name IO_STANDARD "3.3-V LVCMOS" -to LEDR[2]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HEX5[4]
```

```
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HEX4[5]
set_instance_assignment -name IO_STANDARD "3.3-V LVCMOS" -to LEDR[3]
set_instance_assignment -name IO_STANDARD "3.3-V LVCMOS" -to LEDR[1]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HEX4[3]
set_instance_assignment -name IO_STANDARD "3.3-V LVCMOS" -to LEDR[4]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HEX4[4]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HEX4[1]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HEX4[2]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HEX5[5]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HEX5[1]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HEX5[3]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to CLOCK_50
set_instance_assignment -name IO_STANDARD "3.3-V LVCMOS" -to
DRAM_ADDR[0]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HEX4[6]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HEX5[0]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HEX5[2]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HEX4
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to HEX5
set_location_assignment PIN_AB12 -to SW[0]
set_location_assignment PIN_AC12 -to SW[1]
set_location_assignment PIN_AF9 -to SW[2]
set_location_assignment PIN_AF10 -to SW[3]
set_location_assignment PIN_AD11 -to SW[4]
set_location_assignment PIN_AD12 -to SW[5]
set_location_assignment PIN_AE11 -to SW[6]
set_location_assignment PIN_AC9 -to SW[7]
set_location_assignment PIN_AD10 -to SW[8]
set_location_assignment PIN_AE12 -to SW[9]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to SW[9]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to SW[8]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to SW[7]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to SW[6]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to SW[5]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to SW[4]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to SW[3]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to SW[2]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to SW[1]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to SW[0]

# Assign IRQ pins for KEY[0] to KEY[3] (connected to interrupt controller in Qsys)
set_location_assignment PIN_AA14 -to KEY[0]
set_location_assignment PIN_AA15 -to KEY[1]
set_location_assignment PIN_W15 -to KEY[2]
set_location_assignment PIN_Y16 -to KEY[3]

set_global_assignment -name VERILOG_FILE bindhw5/synthesis/bindhw5.v
set_global_assignment -name QIP_FILE bindhw5/synthesis/bindhw5.qip
set_global_assignment -name VERILOG_FILE homework5top.v
set_location_assignment PIN_AJ4 -to ADC_CS_N
```

```
set_location_assignment PIN_AK4 -to ADC_DIN
set_location_assignment PIN_AK3 -to ADC_DOUT
set_location_assignment PIN_AK2 -to ADC_SCLK
set_location_assignment PIN_K7 -to AUD_ADCDAT
set_location_assignment PIN_K8 -to AUD_ADCLRCK
set_location_assignment PIN_H7 -to AUD_BCLK
set_location_assignment PIN_J7 -to AUD_DACDAT
set_location_assignment PIN_H8 -to AUD_DACLRCK
set_location_assignment PIN_G7 -to AUD_XCK
set_location_assignment PIN_AA16 -to CLOCK2_50
set_location_assignment PIN_Y26 -to CLOCK3_50
set_location_assignment PIN_K14 -to CLOCK4_50
set_location_assignment PIN_AA12 -to FAN_CTRL
set_location_assignment PIN_J12 -to FPGA_I2C_SCLK
set_location_assignment PIN_K12 -to FPGA_I2C_SDAT
set_location_assignment PIN_AC18 -to GPIO_0[0]
set_location_assignment PIN_AH18 -to GPIO_0[10]
set_location_assignment PIN_AH17 -to GPIO_0[11]
set_location_assignment PIN_AG16 -to GPIO_0[12]
set_location_assignment PIN_AE16 -to GPIO_0[13]
set_location_assignment PIN_AF16 -to GPIO_0[14]
set_location_assignment PIN_AG17 -to GPIO_0[15]
set_location_assignment PIN_AA18 -to GPIO_0[16]
set_location_assignment PIN_AA19 -to GPIO_0[17]
set_location_assignment PIN_AE17 -to GPIO_0[18]
set_location_assignment PIN_AC20 -to GPIO_0[19]
set_location_assignment PIN_Y17 -to GPIO_0[1]
set_location_assignment PIN_AH19 -to GPIO_0[20]
set_location_assignment PIN_AJ20 -to GPIO_0[21]
set_location_assignment PIN_AH20 -to GPIO_0[22]
set_location_assignment PIN_AK21 -to GPIO_0[23]
set_location_assignment PIN_AD19 -to GPIO_0[24]
set_location_assignment PIN_AD20 -to GPIO_0[25]
set_location_assignment PIN_AE18 -to GPIO_0[26]
set_location_assignment PIN_AE19 -to GPIO_0[27]
set_location_assignment PIN_AF20 -to GPIO_0[28]
set_location_assignment PIN_AF21 -to GPIO_0[29]
set_location_assignment PIN_AD17 -to GPIO_0[2]
set_location_assignment PIN_AF19 -to GPIO_0[30]
set_location_assignment PIN_AG21 -to GPIO_0[31]
set_location_assignment PIN_AF18 -to GPIO_0[32]
set_location_assignment PIN_AG20 -to GPIO_0[33]
set_location_assignment PIN_AG18 -to GPIO_0[34]
set_location_assignment PIN_AJ21 -to GPIO_0[35]
set_location_assignment PIN_Y18 -to GPIO_0[3]
set_location_assignment PIN_AK16 -to GPIO_0[4]
set_location_assignment PIN_AK18 -to GPIO_0[5]
set_location_assignment PIN_AK19 -to GPIO_0[6]
set_location_assignment PIN_AJ19 -to GPIO_0[7]
set_location_assignment PIN_AJ17 -to GPIO_0[8]
```

```
set_location_assignment PIN_AJ16 -to GPIO_0[9]
set_location_assignment PIN_AB17 -to GPIO_1[0]
set_location_assignment PIN_AG26 -to GPIO_1[10]
set_location_assignment PIN_AH24 -to GPIO_1[11]
set_location_assignment PIN_AH27 -to GPIO_1[12]
set_location_assignment PIN_AJ27 -to GPIO_1[13]
set_location_assignment PIN_AK29 -to GPIO_1[14]
set_location_assignment PIN_AK28 -to GPIO_1[15]
set_location_assignment PIN_AK27 -to GPIO_1[16]
set_location_assignment PIN_AJ26 -to GPIO_1[17]
set_location_assignment PIN_AK26 -to GPIO_1[18]
set_location_assignment PIN_AH25 -to GPIO_1[19]
set_location_assignment PIN_AA21 -to GPIO_1[1]
set_location_assignment PIN_AJ25 -to GPIO_1[20]
set_location_assignment PIN_AJ24 -to GPIO_1[21]
set_location_assignment PIN_AK24 -to GPIO_1[22]
set_location_assignment PIN_AG23 -to GPIO_1[23]
set_location_assignment PIN_AK23 -to GPIO_1[24]
set_location_assignment PIN_AH23 -to GPIO_1[25]
set_location_assignment PIN_AK22 -to GPIO_1[26]
set_location_assignment PIN_AJ22 -to GPIO_1[27]
set_location_assignment PIN_AH22 -to GPIO_1[28]
set_location_assignment PIN_AG22 -to GPIO_1[29]
set_location_assignment PIN_AB21 -to GPIO_1[2]
set_location_assignment PIN_AF24 -to GPIO_1[30]
set_location_assignment PIN_AF23 -to GPIO_1[31]
set_location_assignment PIN_AE22 -to GPIO_1[32]
set_location_assignment PIN_AD21 -to GPIO_1[33]
set_location_assignment PIN_AA20 -to GPIO_1[34]
set_location_assignment PIN_AC22 -to GPIO_1[35]
set_location_assignment PIN_AC23 -to GPIO_1[3]
set_location_assignment PIN_AD24 -to GPIO_1[4]
set_location_assignment PIN_AE23 -to GPIO_1[5]
set_location_assignment PIN_AE24 -to GPIO_1[6]
set_location_assignment PIN_AF25 -to GPIO_1[7]
set_location_assignment PIN_AF26 -to GPIO_1[8]
set_location_assignment PIN_AG25 -to GPIO_1[9]
set_location_assignment PIN_AA30 -to IRDA_RXD
set_location_assignment PIN_AB30 -to IRDA_TXD
set_location_assignment PIN_AD7 -to PS2_CLK
set_location_assignment PIN_AE7 -to PS2_DAT
set_location_assignment PIN_AD9 -to PS2_CLK2
set_location_assignment PIN_AE9 -to PS2_DAT2
set_location_assignment PIN_H15 -to TD_CLK27
set_location_assignment PIN_D2 -to TD_DATA[0]
set_location_assignment PIN_B1 -to TD_DATA[1]
set_location_assignment PIN_E2 -to TD_DATA[2]
set_location_assignment PIN_B2 -to TD_DATA[3]
set_location_assignment PIN_D1 -to TD_DATA[4]
set_location_assignment PIN_E1 -to TD_DATA[5]
```

```
set_location_assignment PIN_C2 -to TD_DATA[6]
set_location_assignment PIN_B3 -to TD_DATA[7]
set_location_assignment PIN_A5 -to TD_HS
set_location_assignment PIN_F6 -to TD_RESET_N
set_location_assignment PIN_A3 -to TD_VS
set_location_assignment PIN_AF4 -to USB_B2_CLK
set_location_assignment PIN_AH4 -to USB_B2_DATA[0]
set_location_assignment PIN_AH3 -to USB_B2_DATA[1]
set_location_assignment PIN_AJ2 -to USB_B2_DATA[2]
set_location_assignment PIN_AJ1 -to USB_B2_DATA[3]
set_location_assignment PIN_AH2 -to USB_B2_DATA[4]
set_location_assignment PIN_AG3 -to USB_B2_DATA[5]
set_location_assignment PIN_AG2 -to USB_B2_DATA[6]
set_location_assignment PIN_AG1 -to USB_B2_DATA[7]
set_location_assignment PIN_AF5 -to USB_EMPTY
set_location_assignment PIN_AG5 -to USB_FULL
set_location_assignment PIN_AF6 -to USB_OE_N
set_location_assignment PIN_AG6 -to USB_RD_N
set_location_assignment PIN_AG7 -to USB_RESET_N
set_location_assignment PIN_AG8 -to USB_SCL
set_location_assignment PIN_AF8 -to USB_SDA
set_location_assignment PIN_AH5 -to USB_WR_N
set_location_assignment PIN_B13 -to VGA_B[0]
set_location_assignment PIN_G13 -to VGA_B[1]
set_location_assignment PIN_H13 -to VGA_B[2]
set_location_assignment PIN_F14 -to VGA_B[3]
set_location_assignment PIN_H14 -to VGA_B[4]
set_location_assignment PIN_F15 -to VGA_B[5]
set_location_assignment PIN_G15 -to VGA_B[6]
set_location_assignment PIN_J14 -to VGA_B[7]
set_location_assignment PIN_F10 -to VGA_BLANK_N
set_location_assignment PIN_A11 -to VGA_CLK
set_location_assignment PIN_J9 -to VGA_G[0]
set_location_assignment PIN_J10 -to VGA_G[1]
set_location_assignment PIN_H12 -to VGA_G[2]
set_location_assignment PIN_G10 -to VGA_G[3]
set_location_assignment PIN_G11 -to VGA_G[4]
set_location_assignment PIN_G12 -to VGA_G[5]
set_location_assignment PIN_F11 -to VGA_G[6]
set_location_assignment PIN_E11 -to VGA_G[7]
set_location_assignment PIN_B11 -to VGA_HS
set_location_assignment PIN_A13 -to VGA_R[0]
set_location_assignment PIN_C13 -to VGA_R[1]
set_location_assignment PIN_E13 -to VGA_R[2]
set_location_assignment PIN_B12 -to VGA_R[3]
set_location_assignment PIN_C12 -to VGA_R[4]
set_location_assignment PIN_D12 -to VGA_R[5]
set_location_assignment PIN_E12 -to VGA_R[6]
set_location_assignment PIN_F13 -to VGA_R[7]
set_location_assignment PIN_C10 -to VGA_SYNC_N
```

```
set_location_assignment PIN_D11 -to VGA_VS
set_global_assignment -name PARTITION_NETLIST_TYPE SOURCE -entity
DE1_SoC_Computer -section_id Top
set_global_assignment -name PARTITION_FITTER_PRESERVATION_LEVEL
PLACEMENT_AND_ROUTING -entity DE1_SoC_Computer -section_id Top
set_global_assignment -name PARTITION_COLOR 16764057 -entity
DE1_SoC_Computer -section_id Top
set_instance_assignment -name PARTITION_HIERARCHY root_partition -to | -
entity DE1_SoC_Computer -section_id Top
set_instance_assignment -name PARTITION_HIERARCHY root_partition -to | -
section_id Top
```