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

TECHNICAL REPORT

Assignment: Assignment 4

Course Title: ECE 278 - Embedded Systems

Instructor: Dr. Reza Raeisi

Student Name: Bindu Sree Chandu

INSTRUCTOR SECTION

Comments: _____

_____

_____

_____

Final Grade: _____

Contents page

1 Objective

.

To design, implement and verify a complete interrupt-driven reaction game on the Intel DE1-SoC FPGA. This involved developing a custom Nios II embedded system in Qsys for hardware management and writing C code to control game logic, including random LED patterns, button input for score, timed gameplay via an interval timer and difficulty selection.

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, timers, 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.

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
    - Seven-segment displays
    - Switches
- High-resolution timer for microsecond-level accuracy
- System timer for millisecond interval timing

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 game's 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:

Game Initialization & Difficulty Selection

- **Action:** Initiates/restarts the game session. Sets game difficulty.
- **Mechanism:** Activated by KEY0 press (ISR-driven). Reads SW0 for difficulty setting; seeds random number generator.
- **Output:** HEX displays 00, LEDs show initial pattern.

Interrupt-Driven Game Timing

- **Action:** Manages overall game duration and pattern display intervals.
- **Mechanism:** A msTimer ISR tracks game time and pattern display periods. Applies score penalty if a pattern times out.
- **Output:** Triggers new LED patterns. Deducts 5 points from score on timeout.

Player Input & Scoring Logic

- **Action:** Processes user button presses for pattern matching and game control.
- **Mechanism:** KEY PIO ISR detects presses. Compares player input (KEY1-KEY3) with current LED pattern. Updates score based on match (+10) or mismatch (-5). Handles game termination via KEY3.
- **Output:** Real-time score updates on HEX displays. Game ends with final score displayed.

Upon compilation, the .elf file was successfully loaded onto the DE1-SoC board and all game functions were 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.

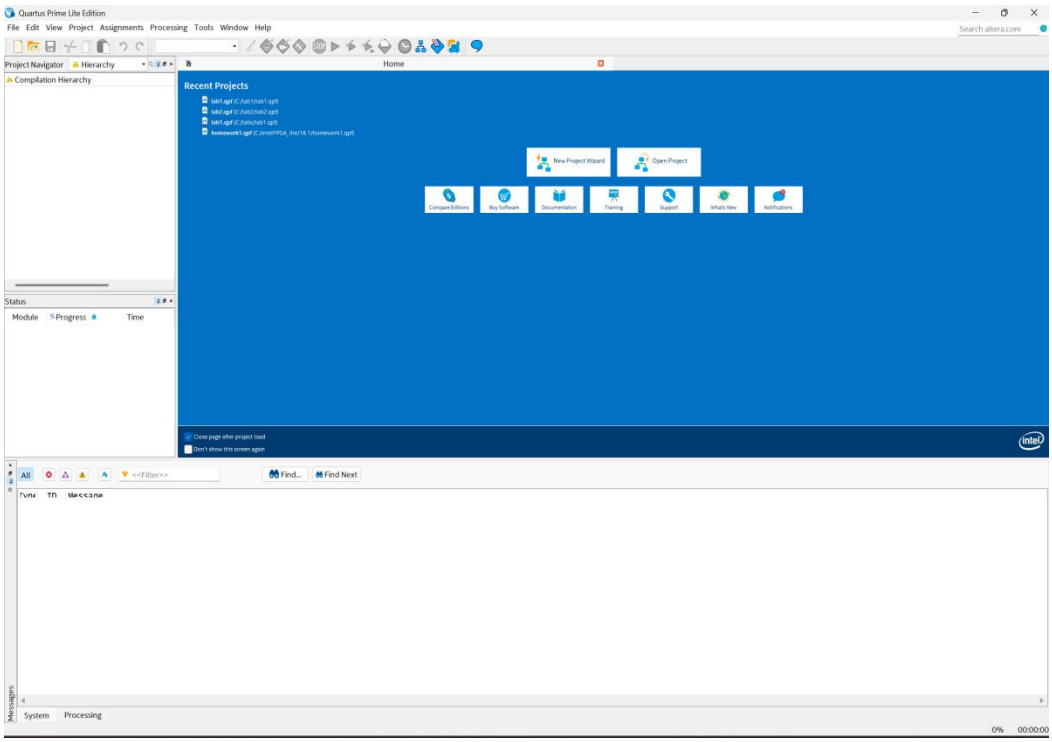


Figure 2: Accessing the New Project Wizard from the Quartus File menu.

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

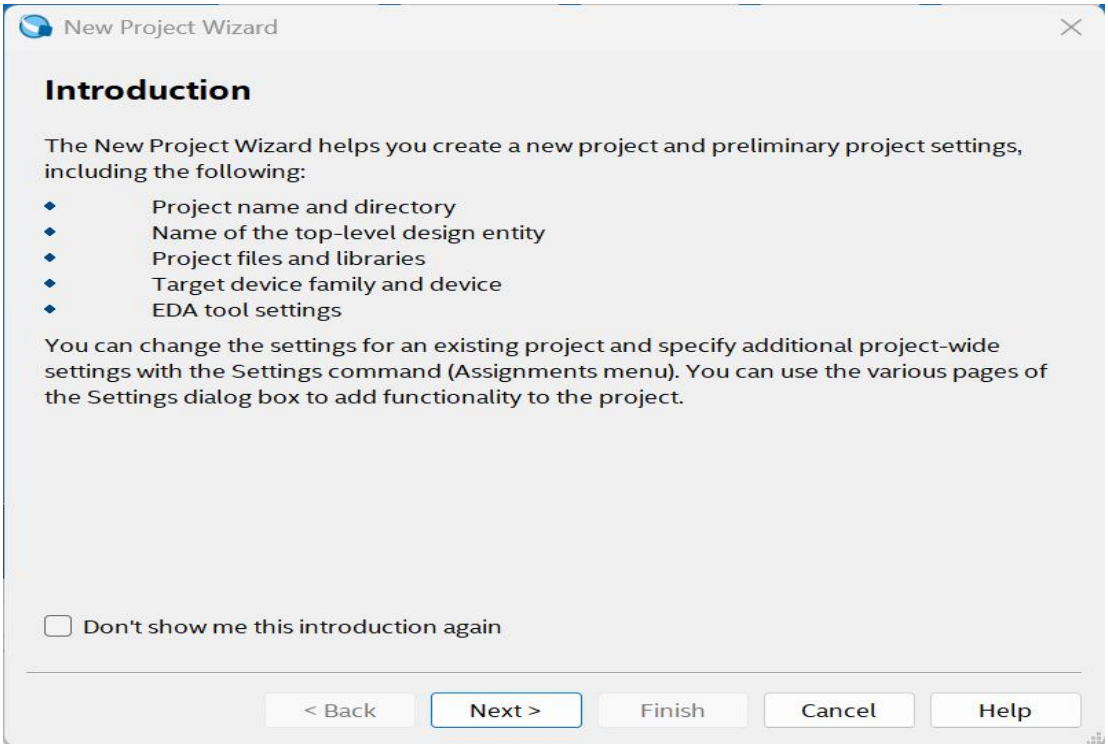


Figure 3: Introduction screen of the New Project Wizard, outlining key setup parameters.

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.

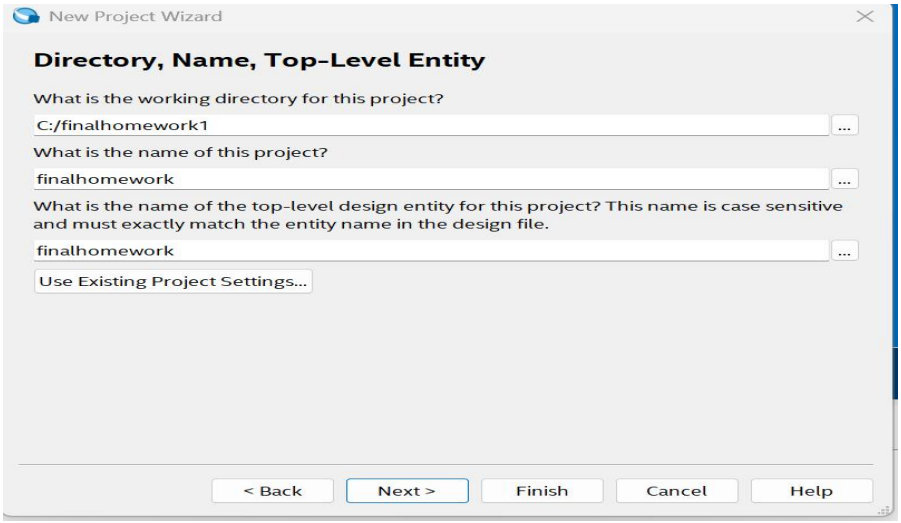


Figure 4: Defining the project directory, name and top-level design entity.

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

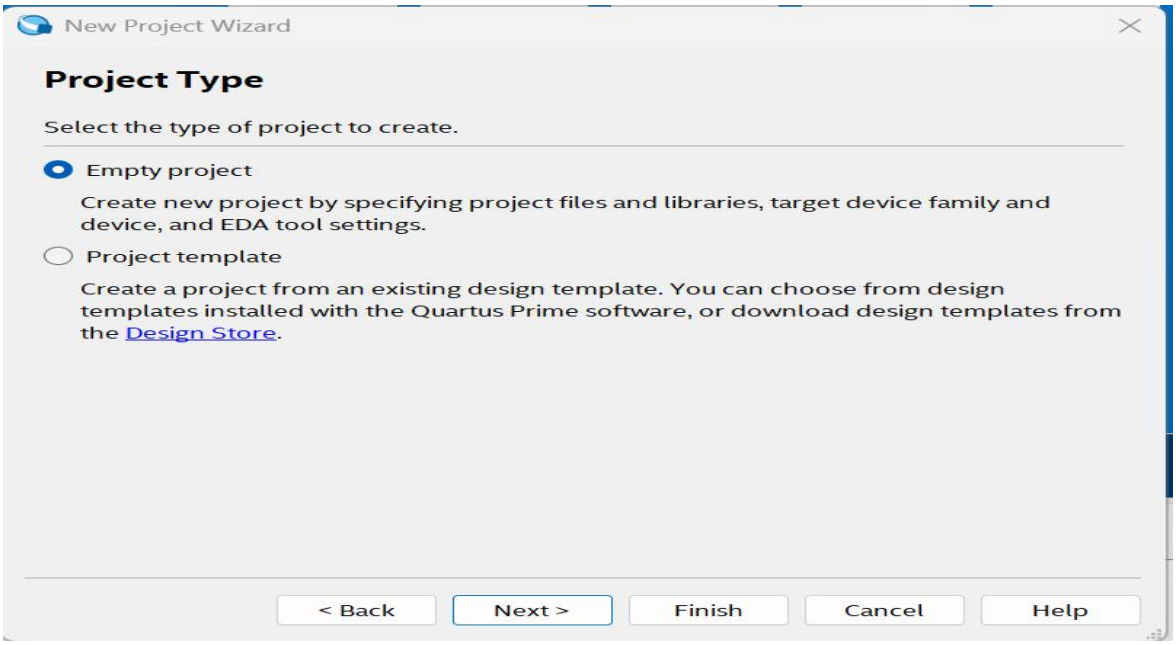


Figure 5: Choosing the "Empty Project" option for a custom setup.

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.

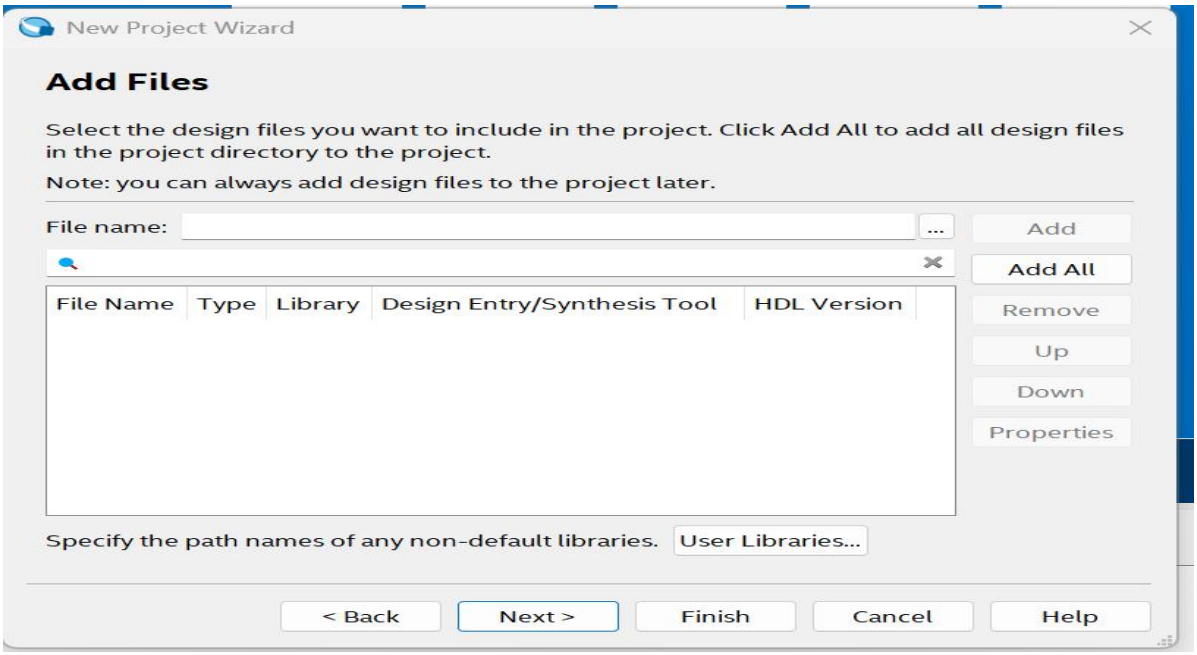


Figure 6: Skipping the file addition step for now, to be handled manually after Qsys generation.

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.

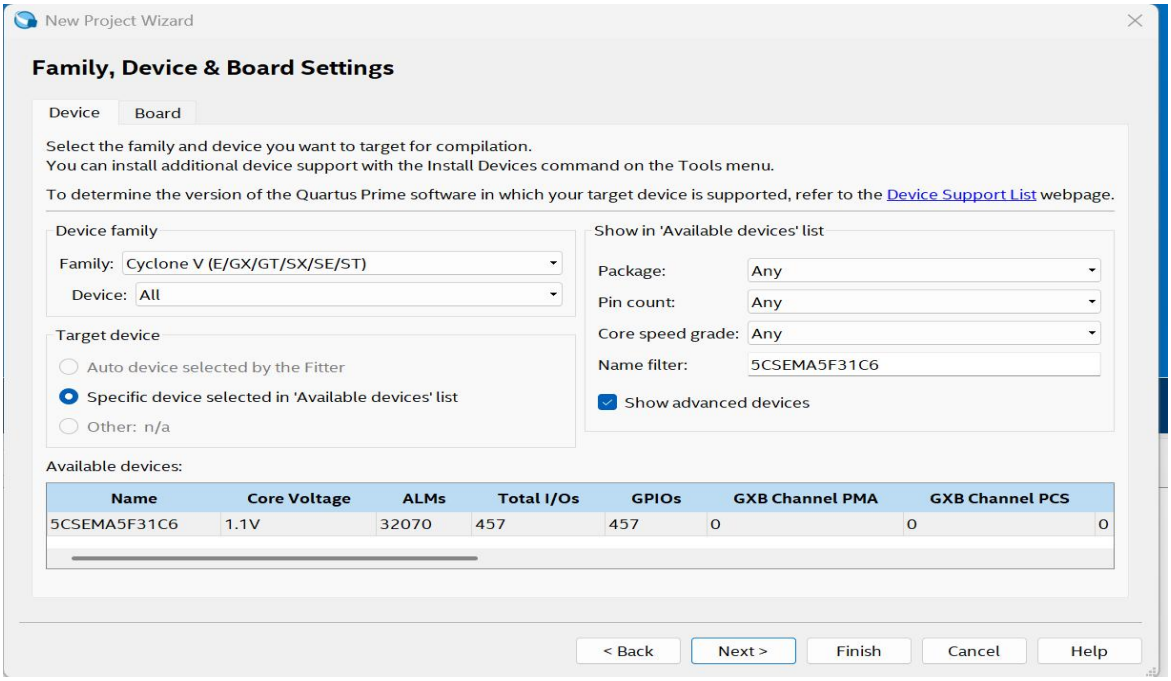


Figure 7: Selecting the Cyclone V device used on the DE1-SoC FPGA board.

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

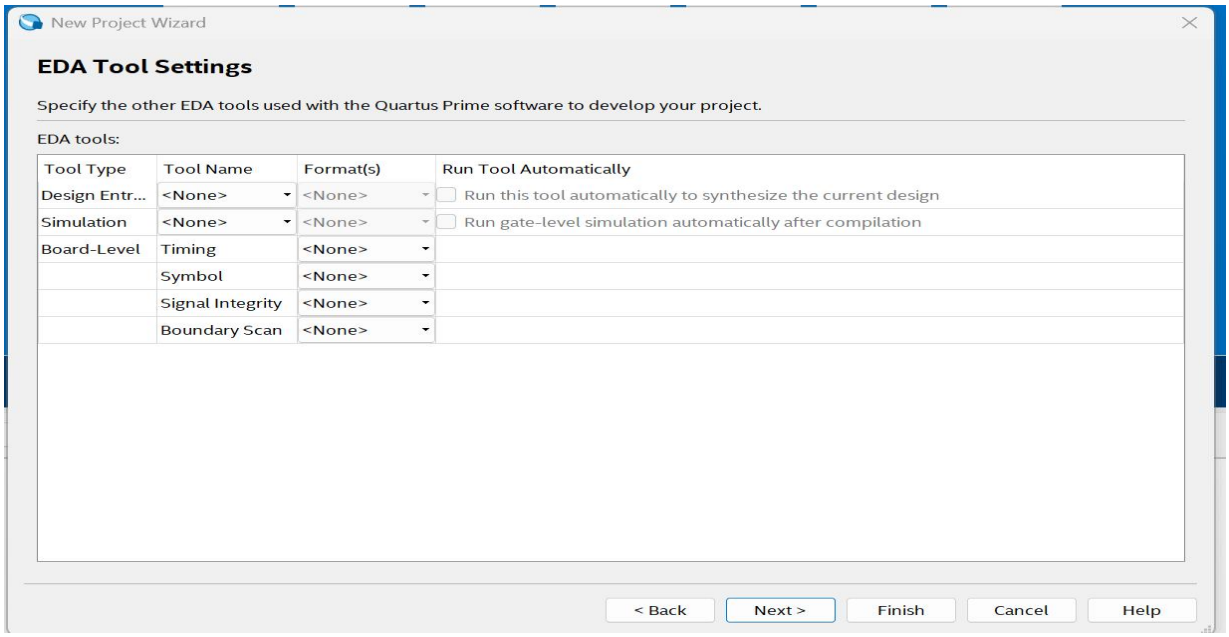


Figure 8: Keeping default settings for EDA tool configuration.

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.

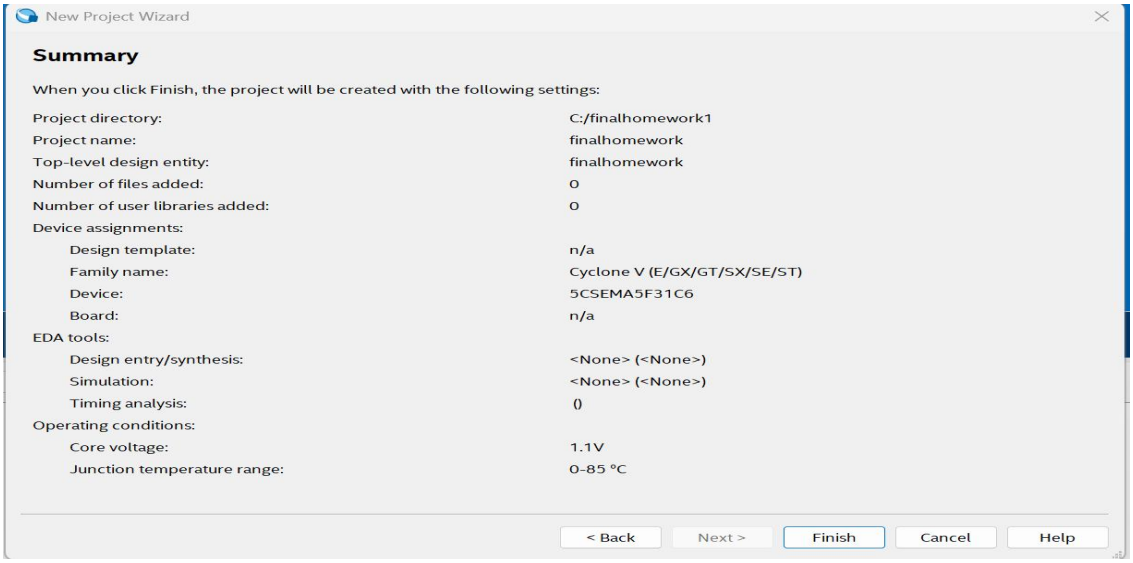


Figure 9: Summary screen confirming all project configuration settings before final creation.

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.

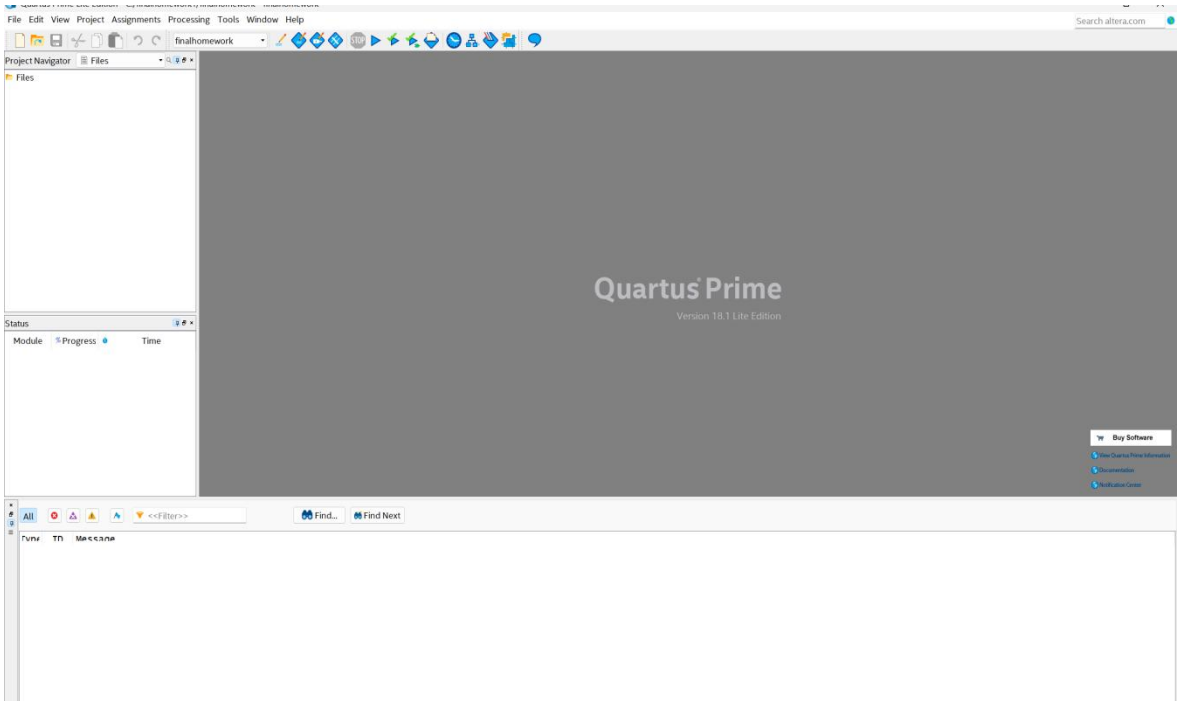


Figure 10: Quartus Prime main interface after project creation, ready for system integration and further development.

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.

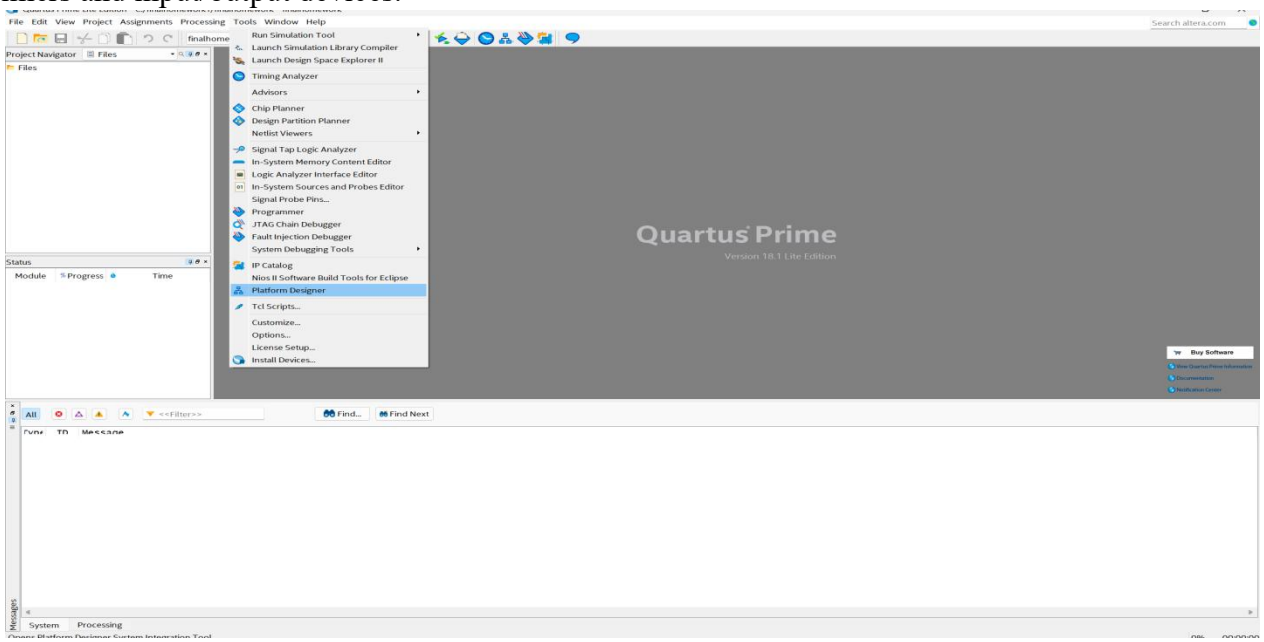


Figure 11: Opening Platform Designer from the Quartus Tools menu.

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

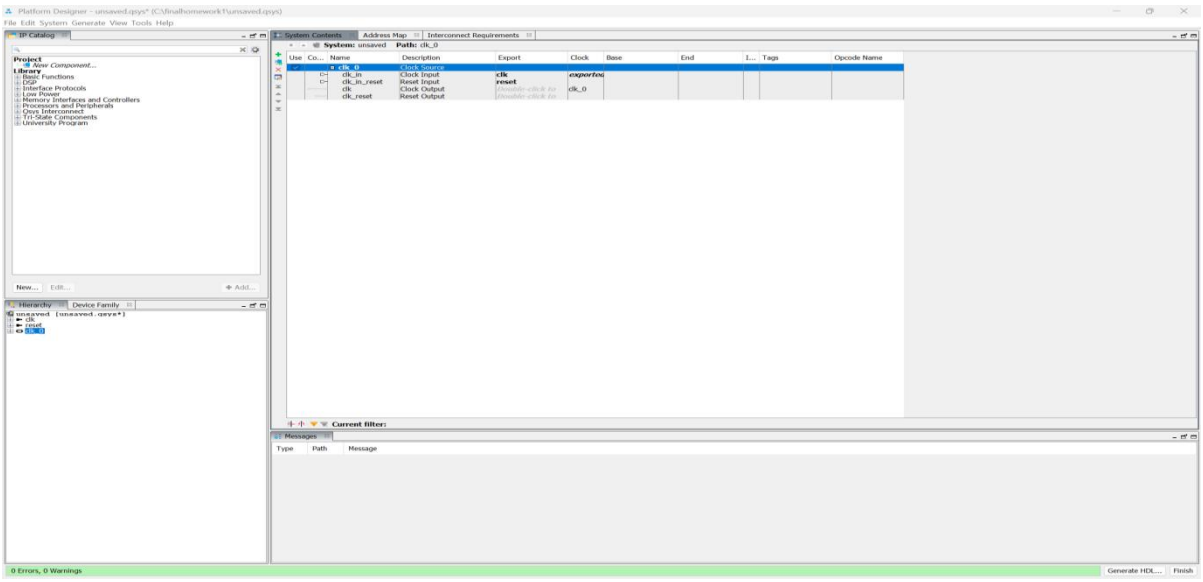


Figure 12: canvas space in qsys, Clock Source component added to the system. Removed this later

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.

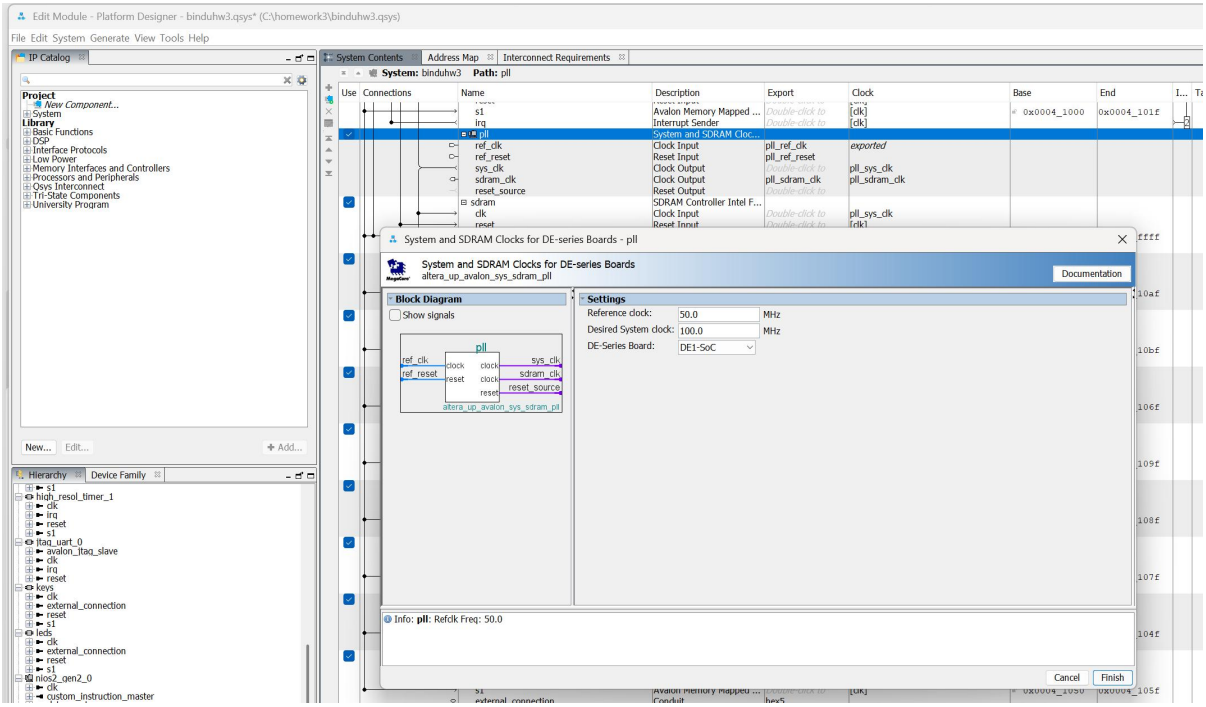


Figure 13: Setting the pll Clock frequency.

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

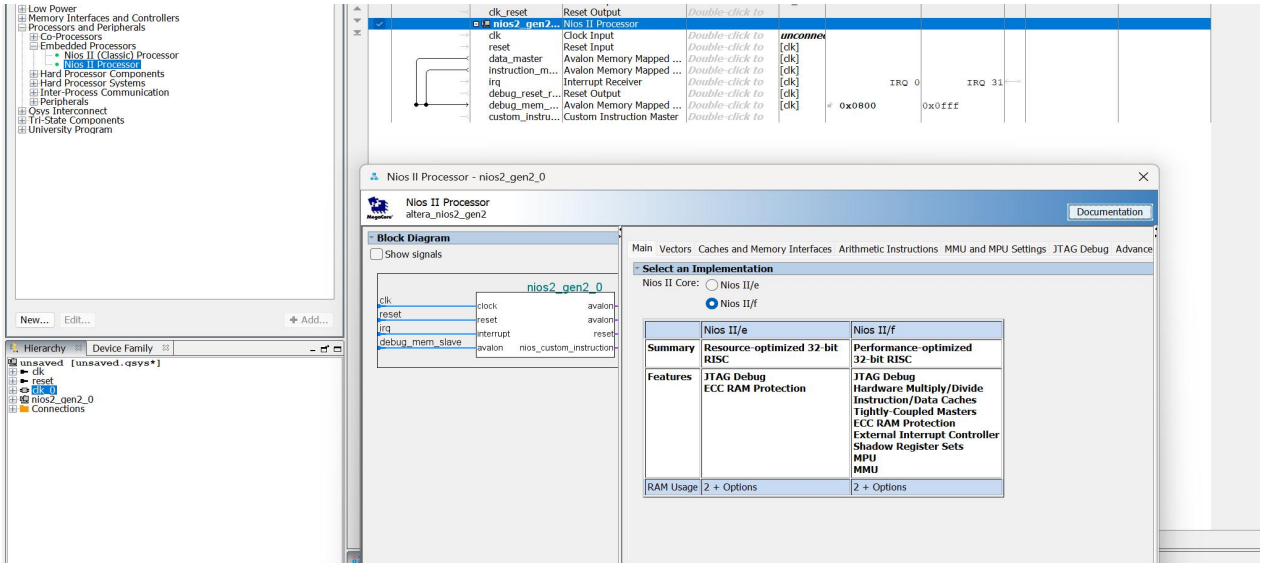


Figure 14: Selecting the Nios II Processor from the IP Catalog and adding into design

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.

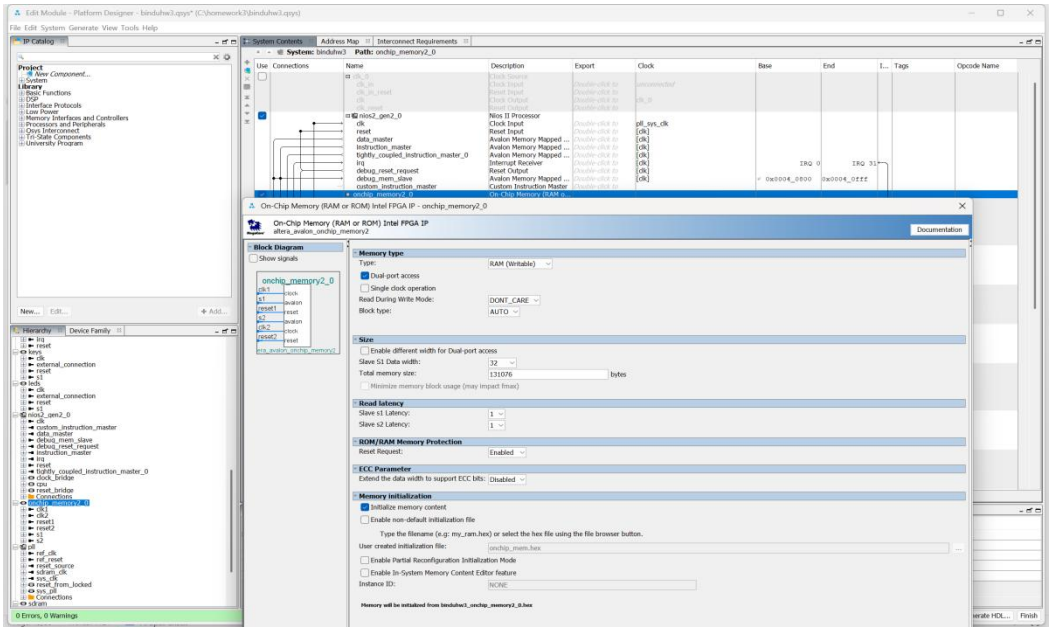


Figure 15: On-Chip Memory Configuration

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.

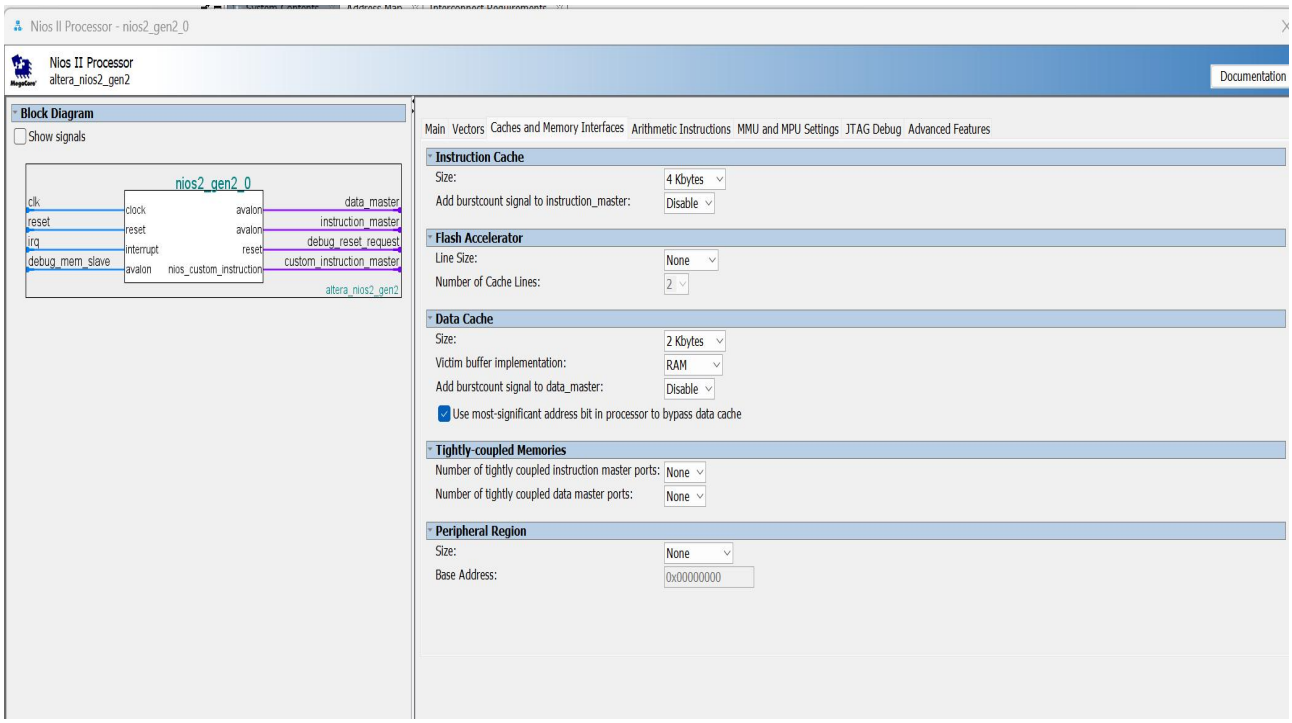


Figure 16: Configuring Cache for Nios II Processor

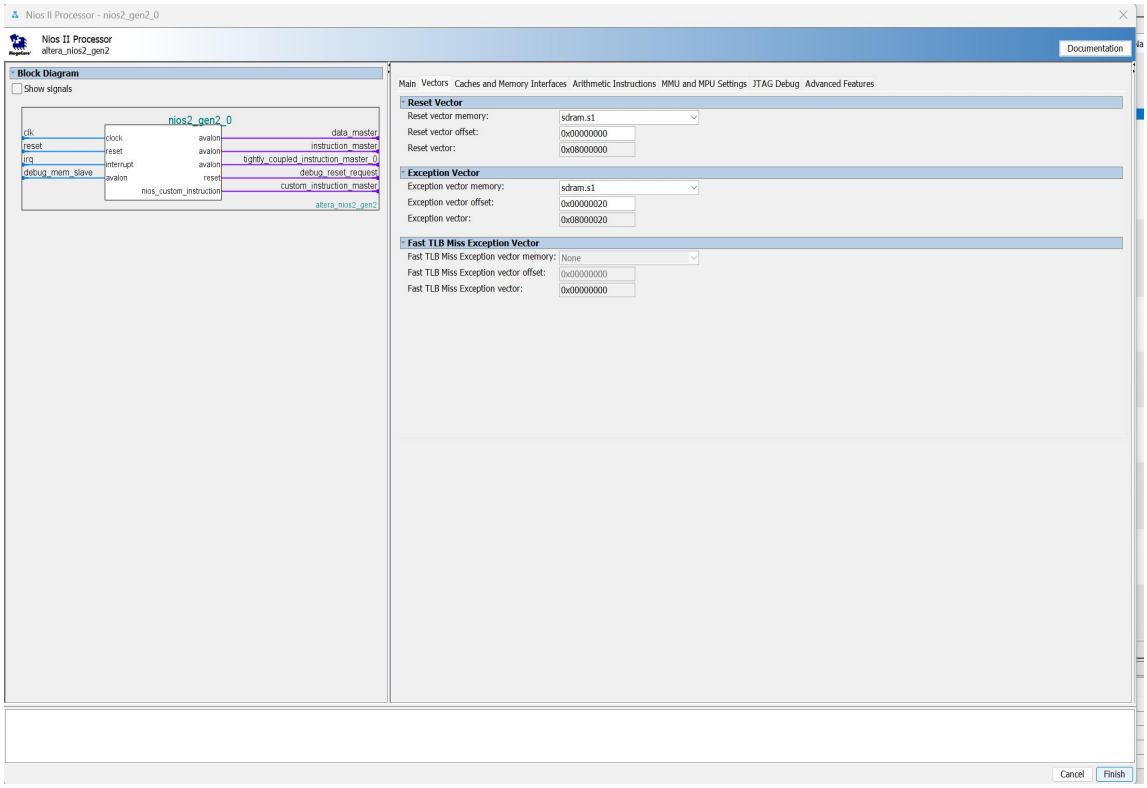


Figure 17: Setting Reset and Exception Vectors

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.

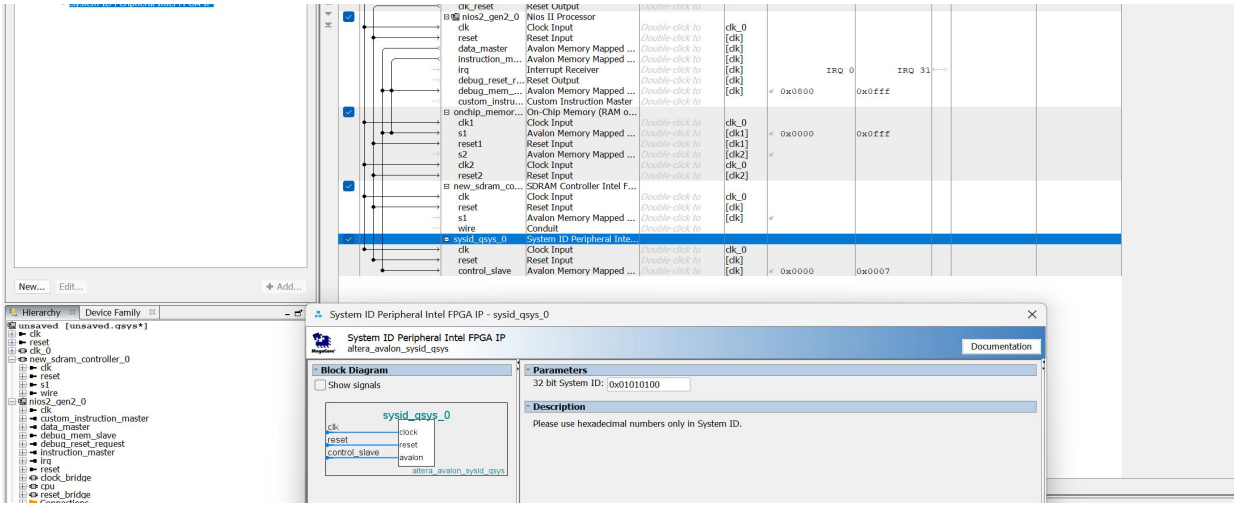


Figure 18: Configuring system ID peripheral

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

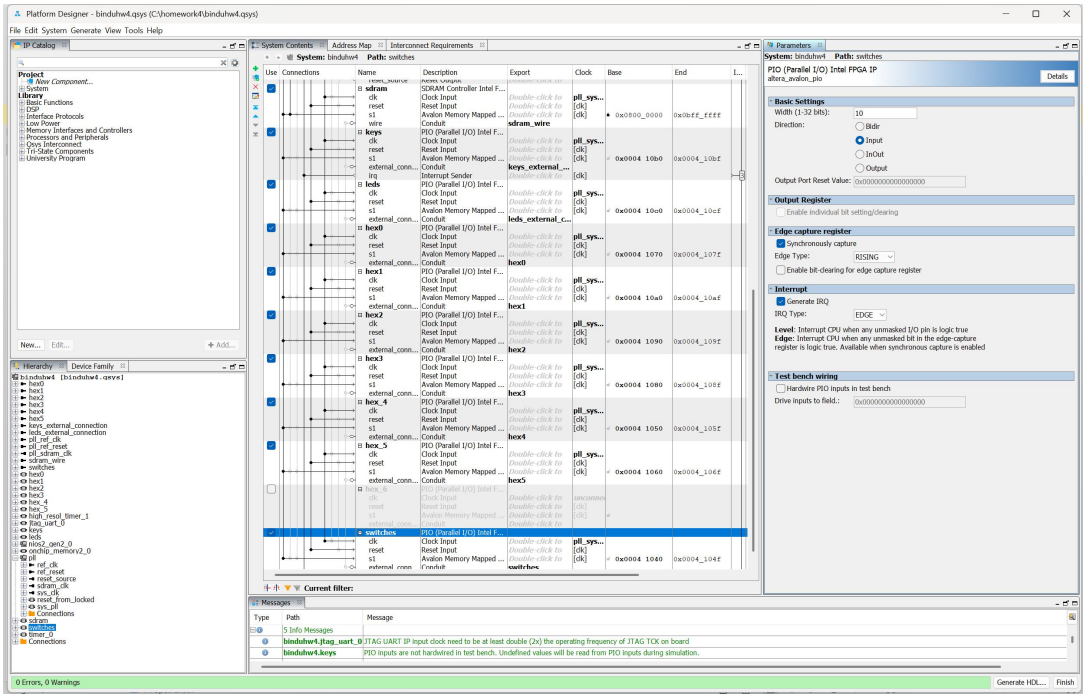


Figure 19: Configuring switches

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.

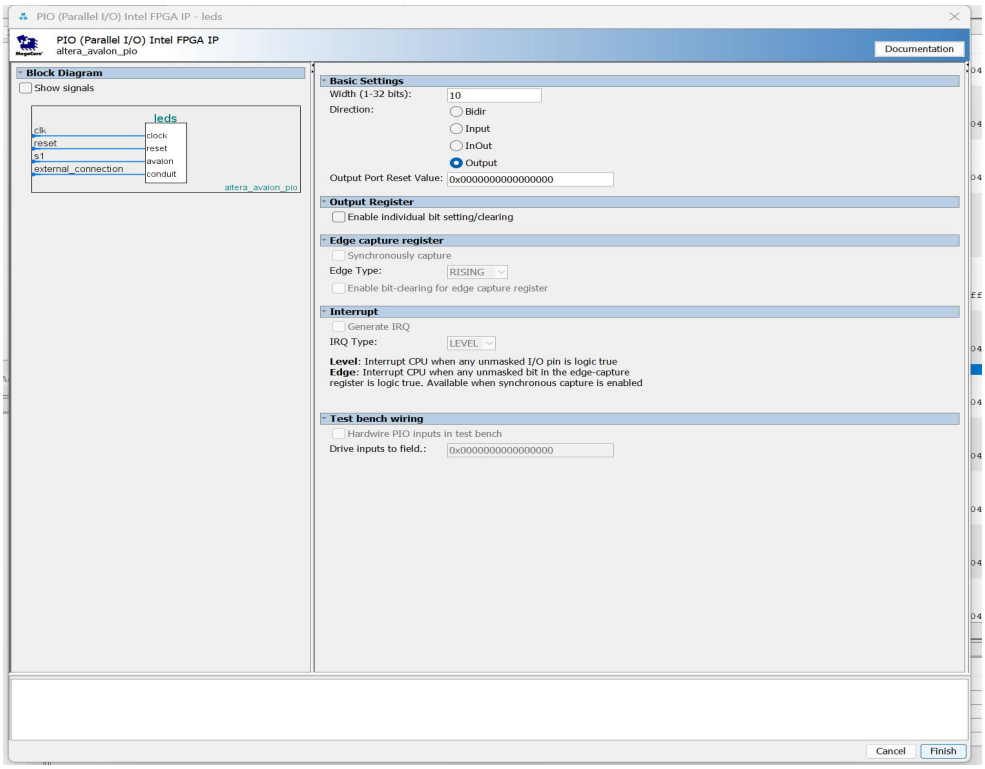


Figure 20: Configuring LEDR PIO

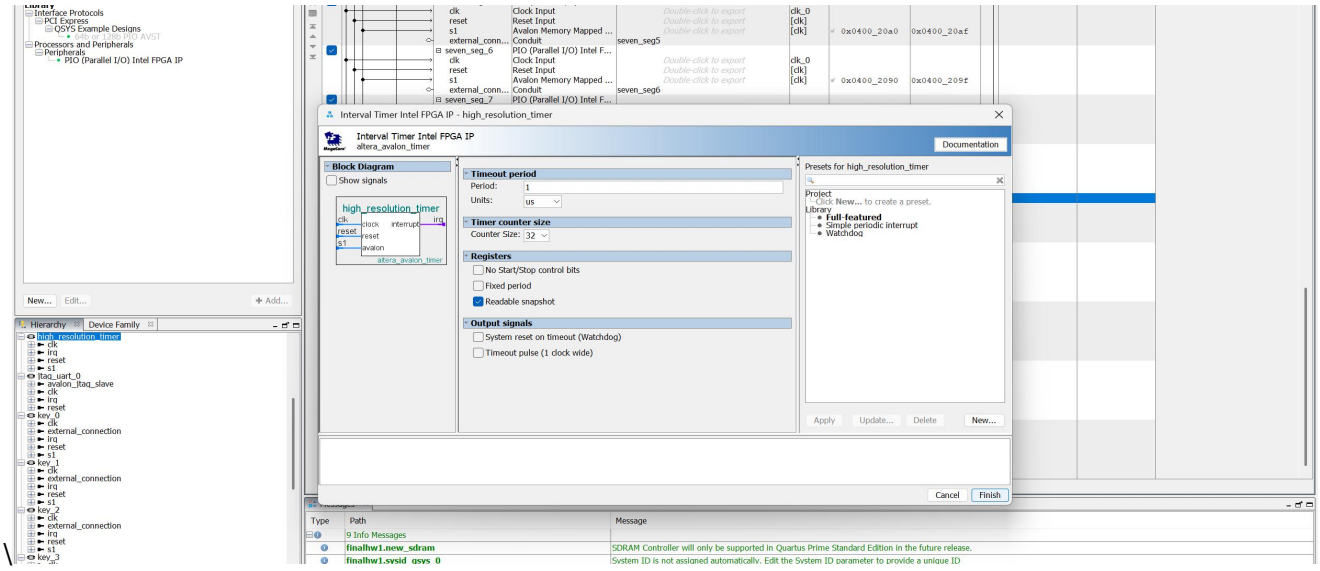


Figure 21: Configuring High Resolution Timer in Platform Designer

In this step, the "Interval Timer Intel FPGA IP" is added and as high_resolution_timer. This timer is configured with a period in microseconds (us) and a 32-bit counter size. This timer will be useful for measuring short time intervals or generating periodic interrupts.

This image shows the setup of the seven-segment display module in Platform Designer. The width is set to 7 bits, direction is set to Output and all other optional features like interrupts and edge capture are left disabled. I duplicated the same components for all seven segments.

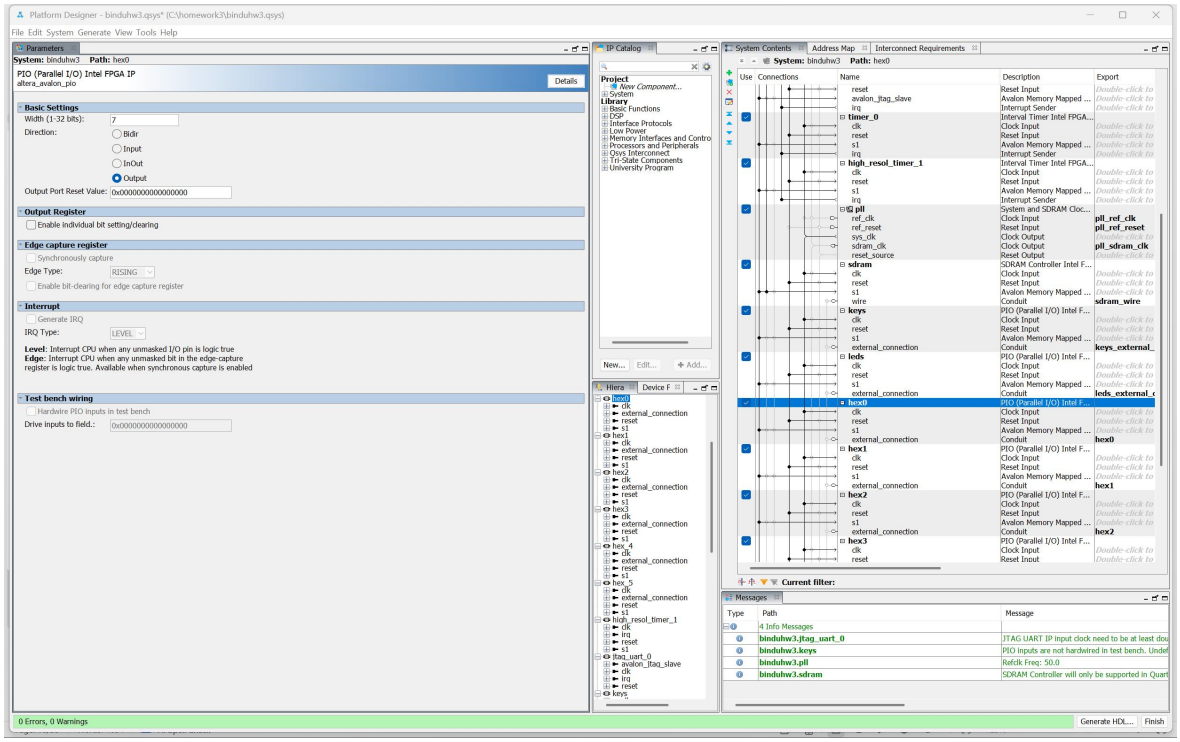


Figure 22: Configuration of Seven Segment Display

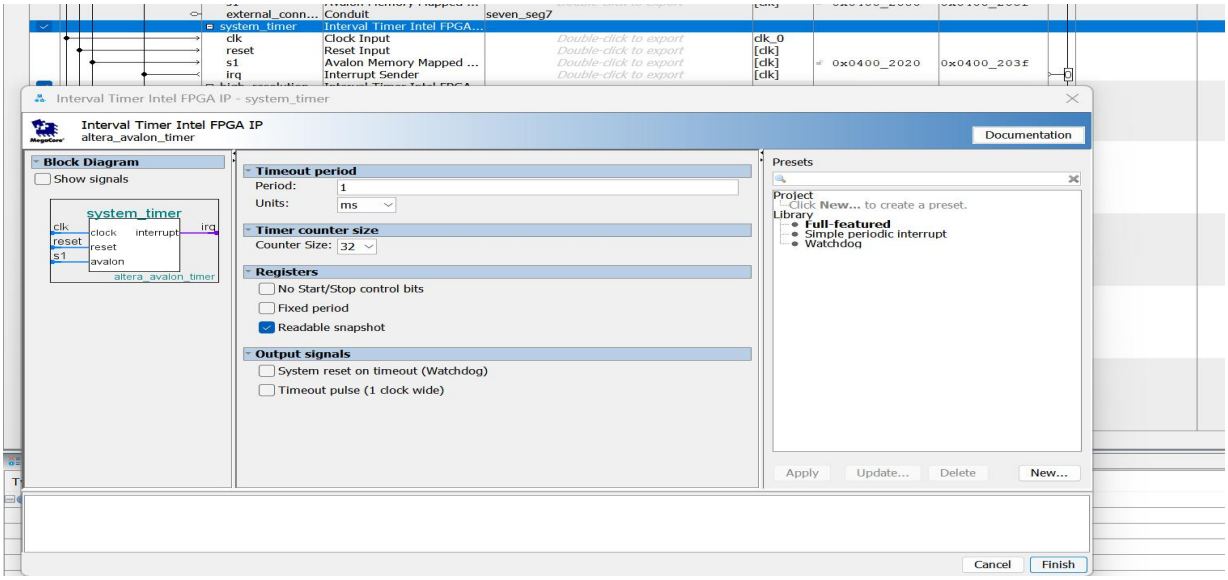


Figure 23: Adding and Configuring System Timer

Another timer, system_timer, was added with a timeout period of 1 millisecond. Like the previous one, it was configured with a 32-bit counter and snapshot enabled.

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.

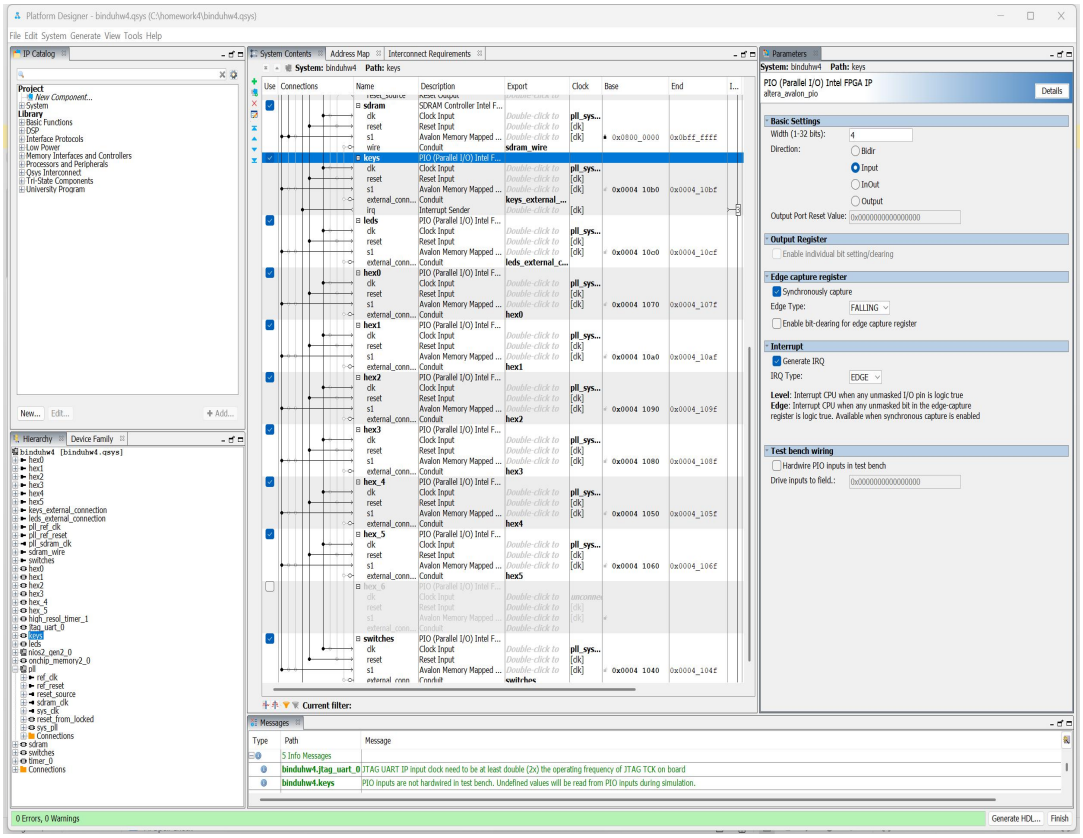


Figure 24: Configuring keys as input (PIO)

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.

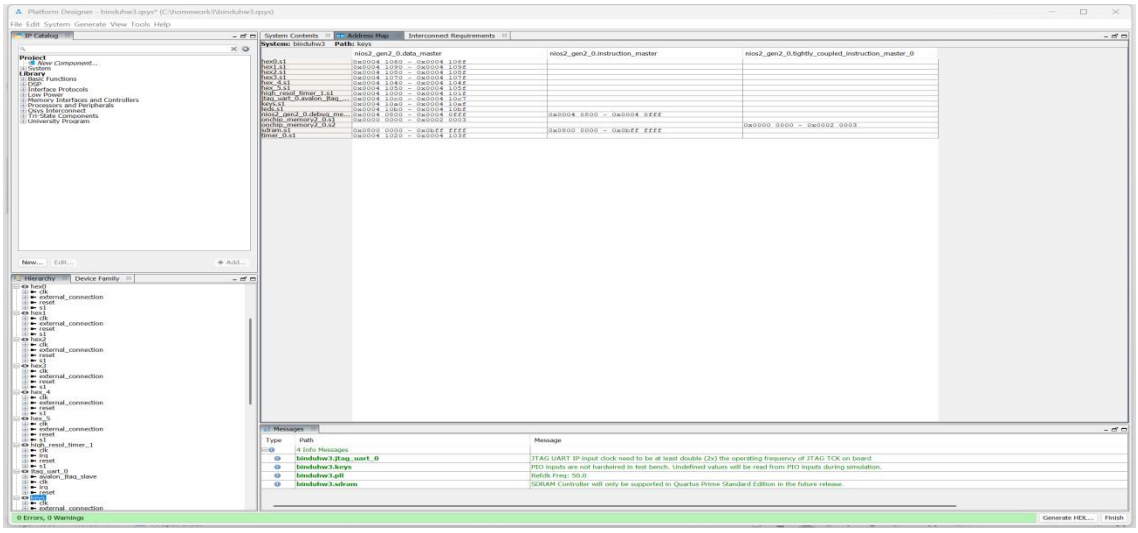


Figure 25: Memory allocation doesn't overlap. Auto assign base address

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

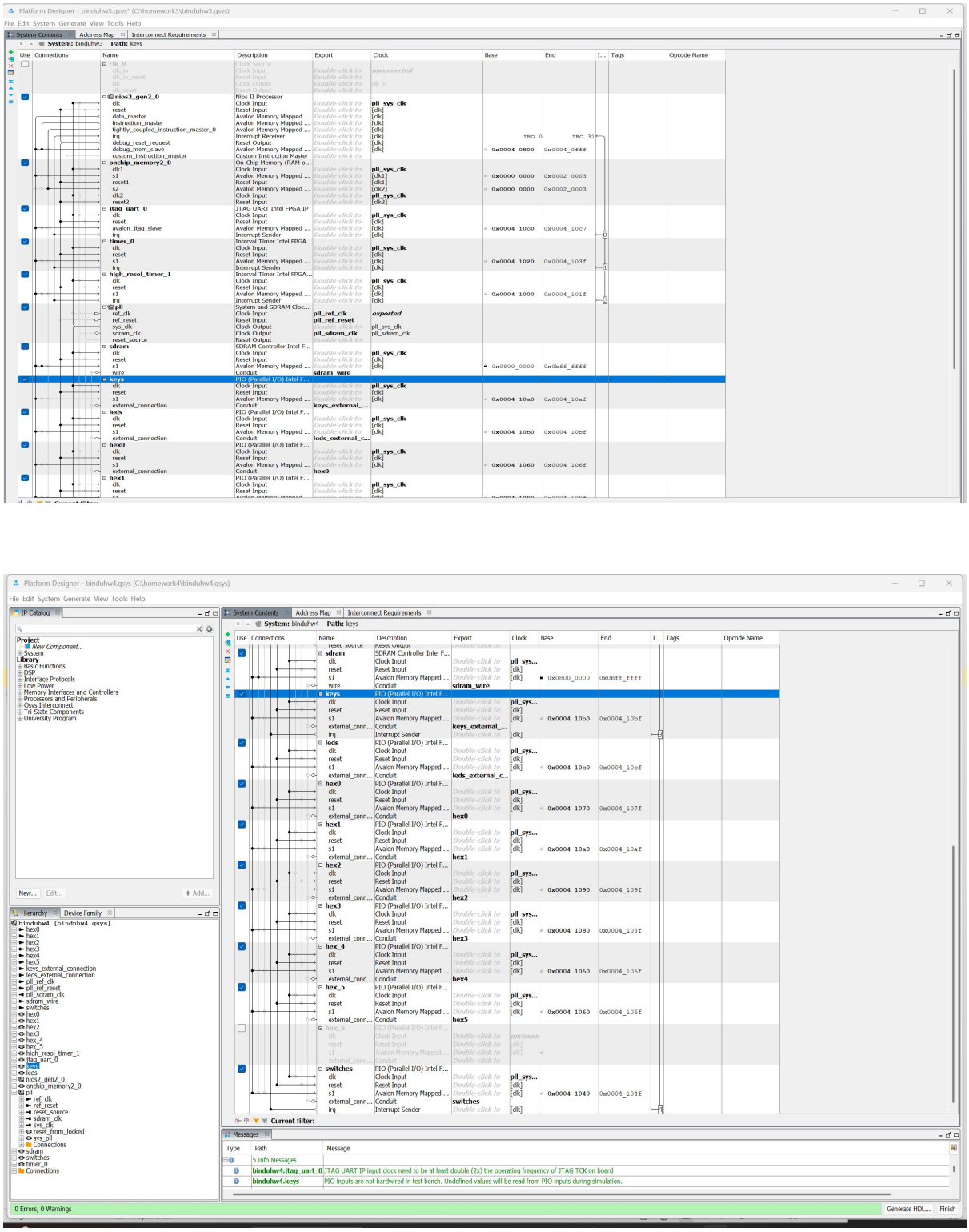


Figure 26: qsys design layout

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.

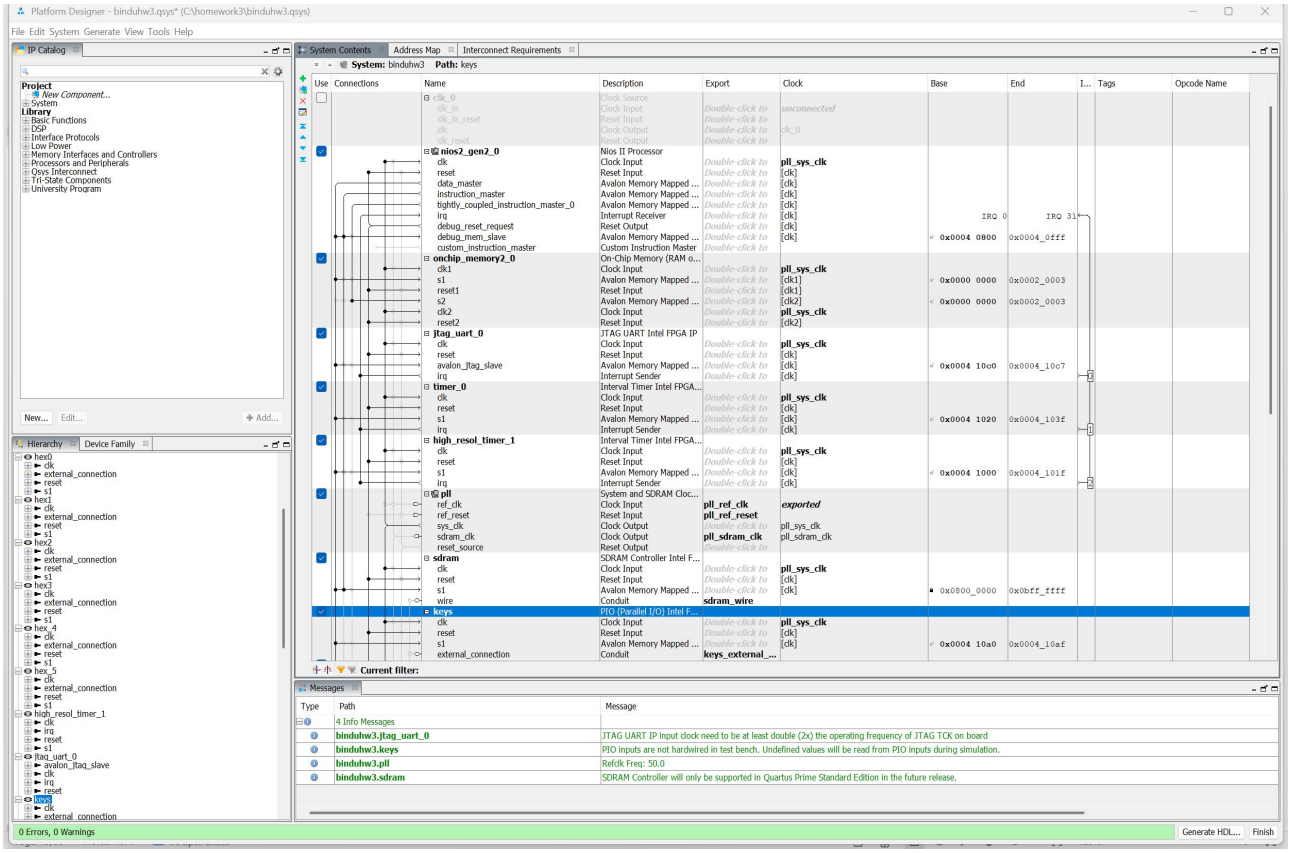


Figure 27: Final Connections and Summary in Platform Designer

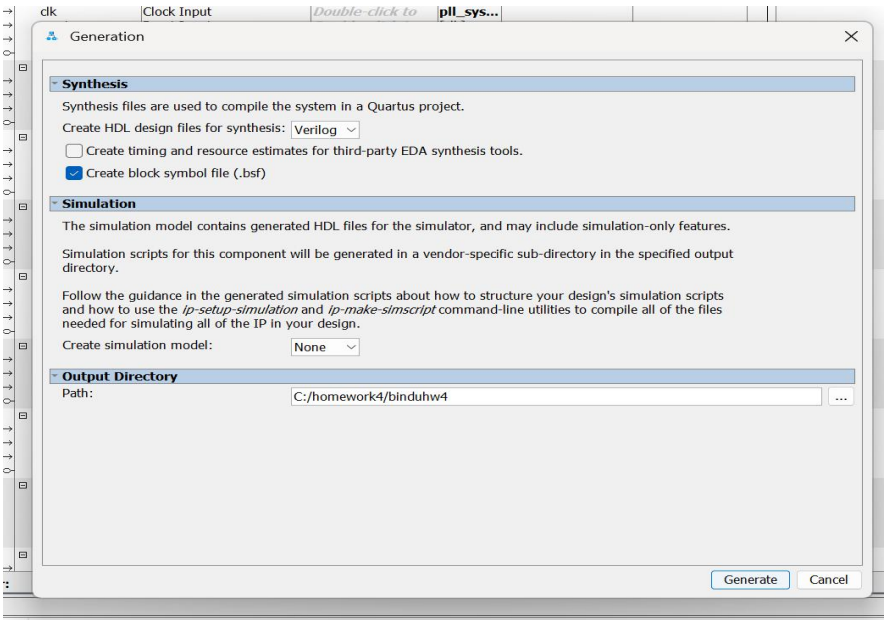


Figure 28: Generate HDL

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.

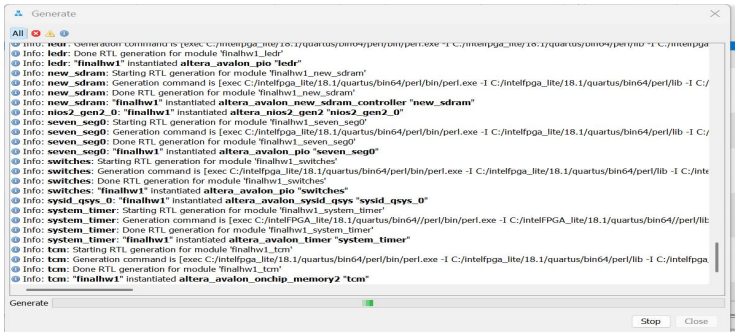


Figure 29: Generation process in progress

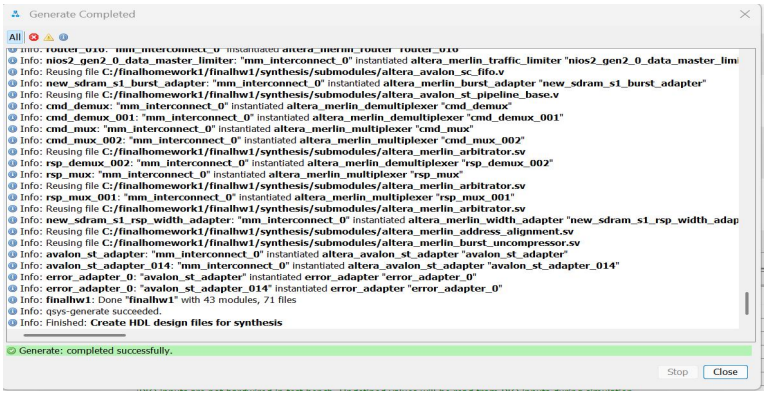


Figure 30: Generation completed successfully

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.

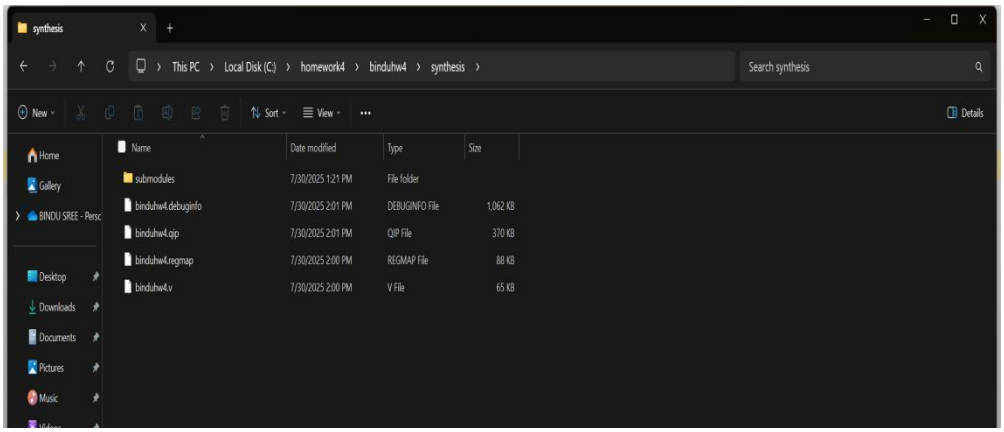


Figure 31: Files created in project file directory

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.

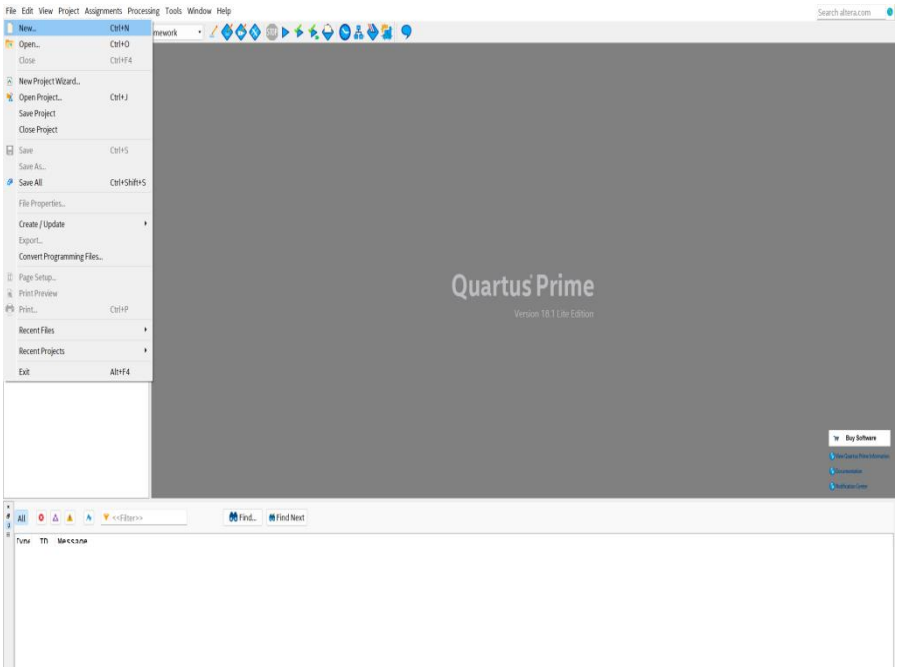


Figure 32: Creating new file from Quartus File Menu

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.

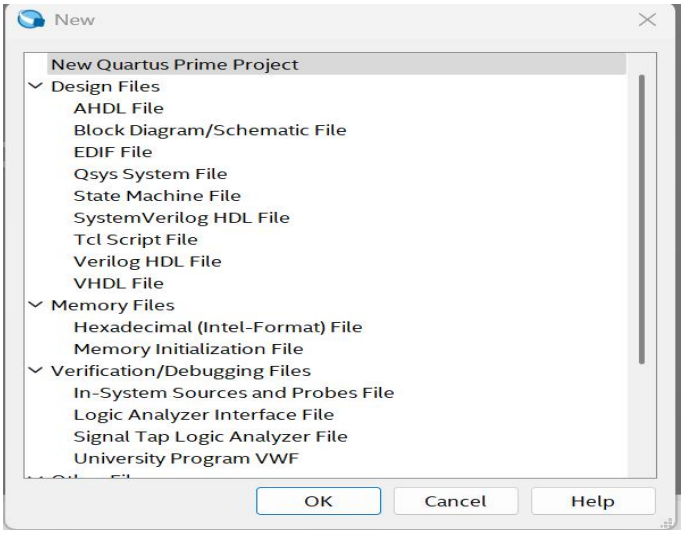


Figure 33: New File Types Selection

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.

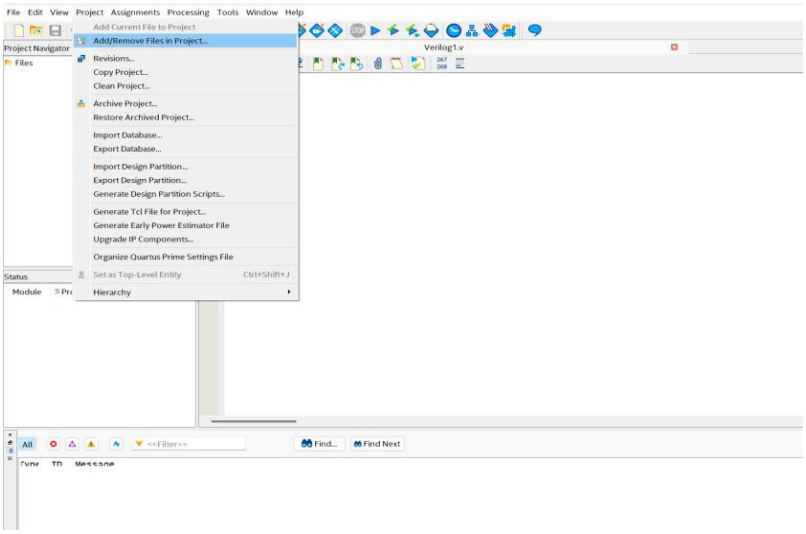


Figure 34: Adding Files to the Quartus Project

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

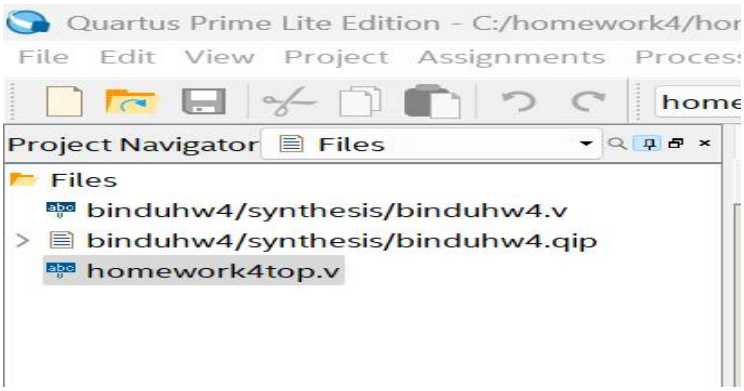
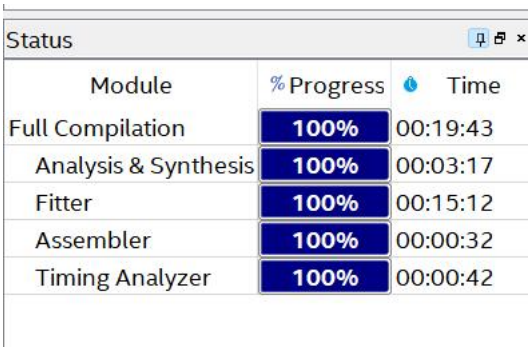


Figure 35: In project navigator, these loaded files can be seen



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.

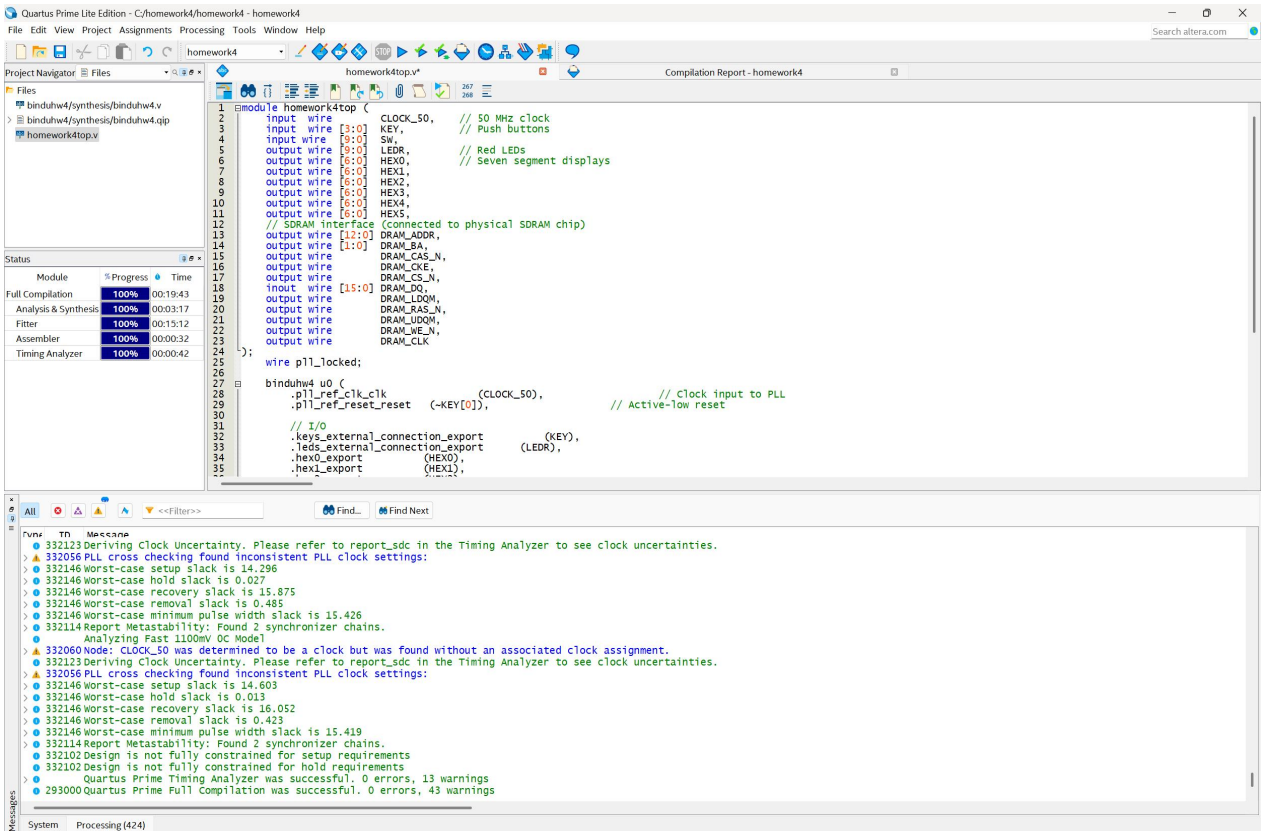


Figure 37: Success compilation for analysis block, no syntax errors

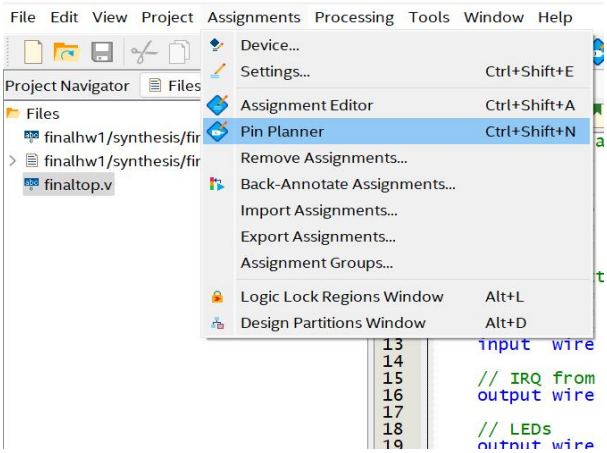


Figure 38: Accessing Pin Planner via Assignments Menu

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.

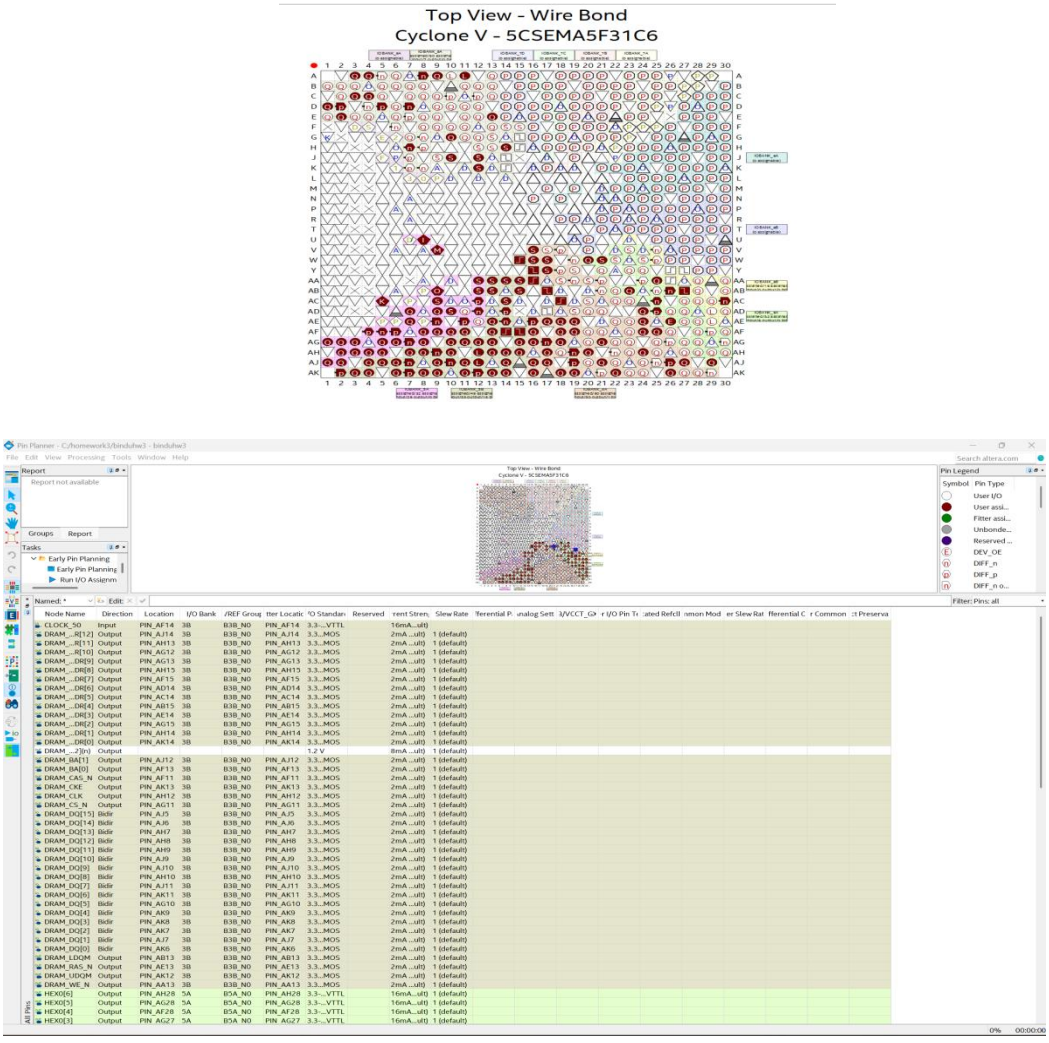


Figure 39: Pin Planner with Assigned I/O Pins

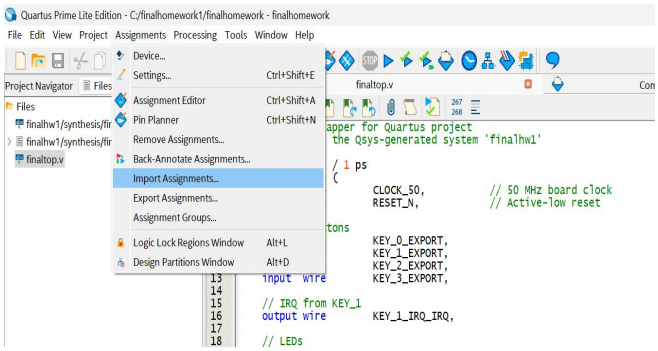


Figure 40: Import Assignments for pins

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

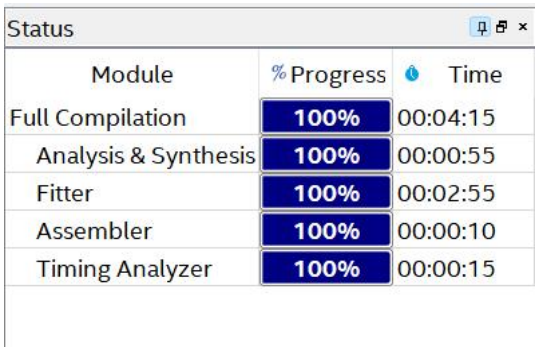


Figure 41: Status panel shows 100%

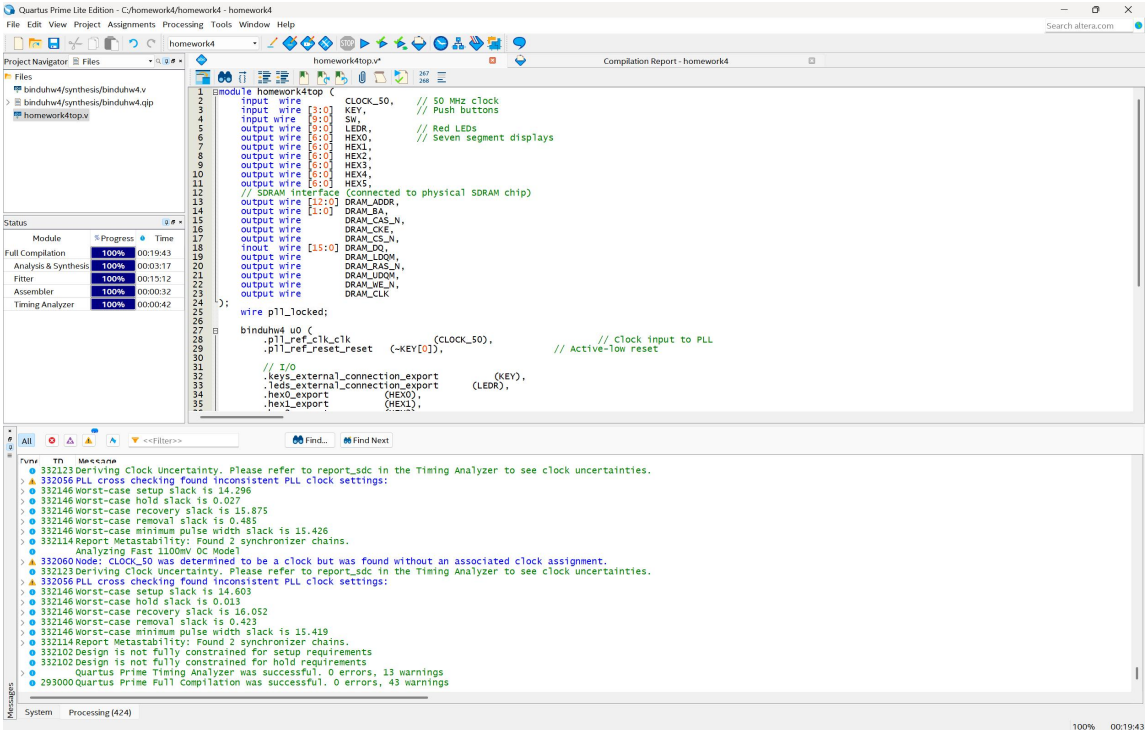


Figure 42: Successful Compilation and Analysis in Quartus Prime

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.

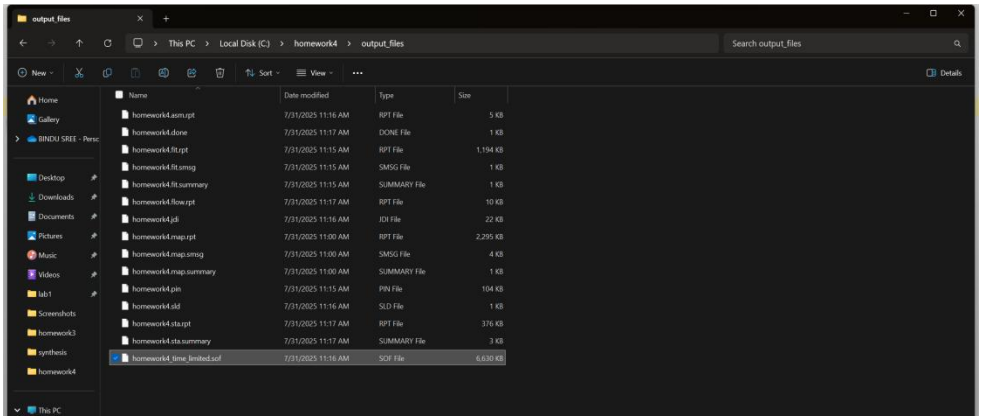


Figure 43: .sof files generated in file manager

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.

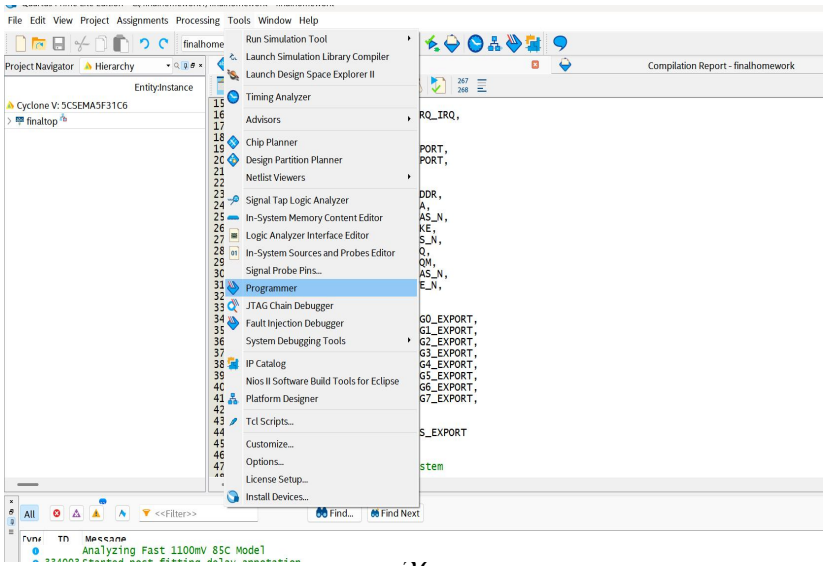


Figure 44: Launching the Programmer Tool in Quartus Prime

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.

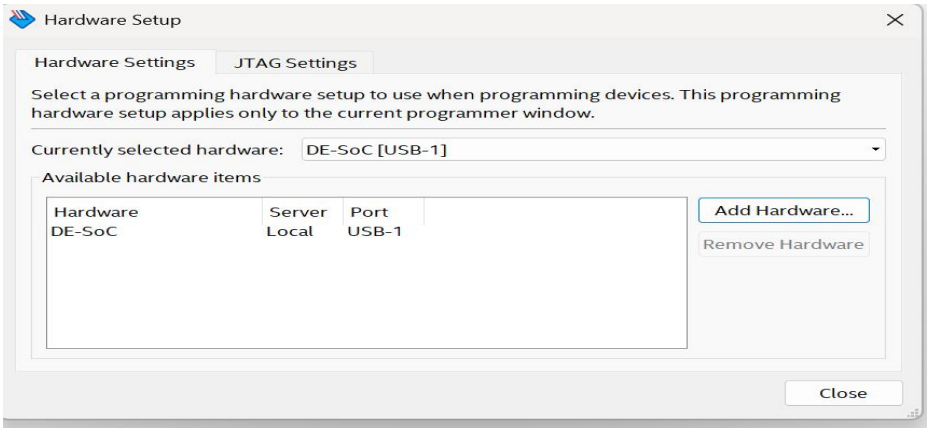


Figure 45: Hardware Setup Selection in Quartus Prime Programmer

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

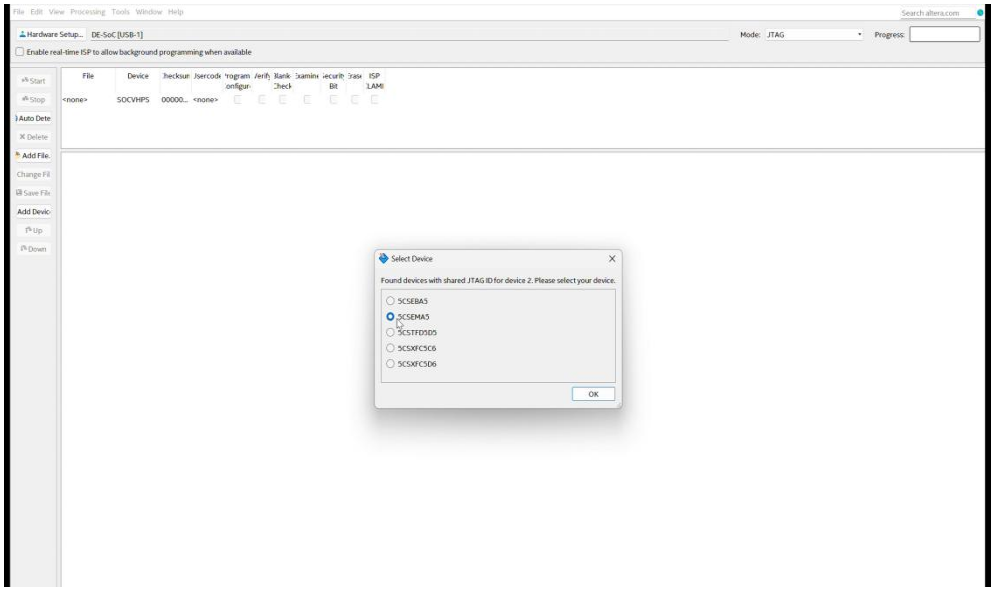


Figure 46: Device selection in Quartus Prime Programmer

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.

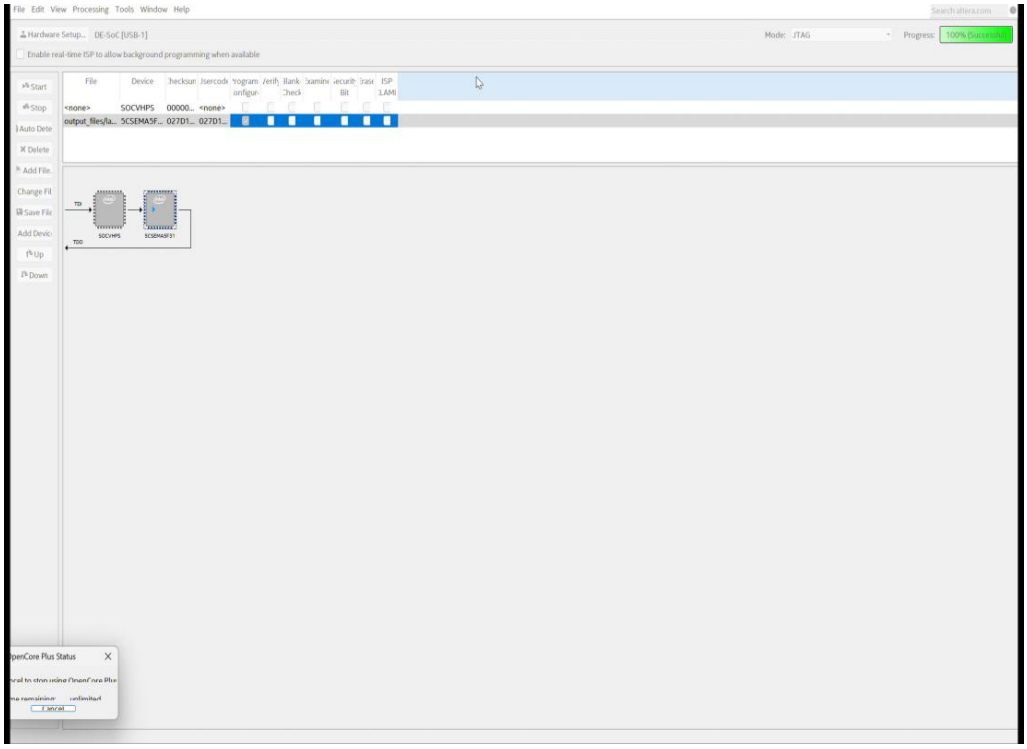


Figure 47: Successful FPGA Programming in Quartus Programmer

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

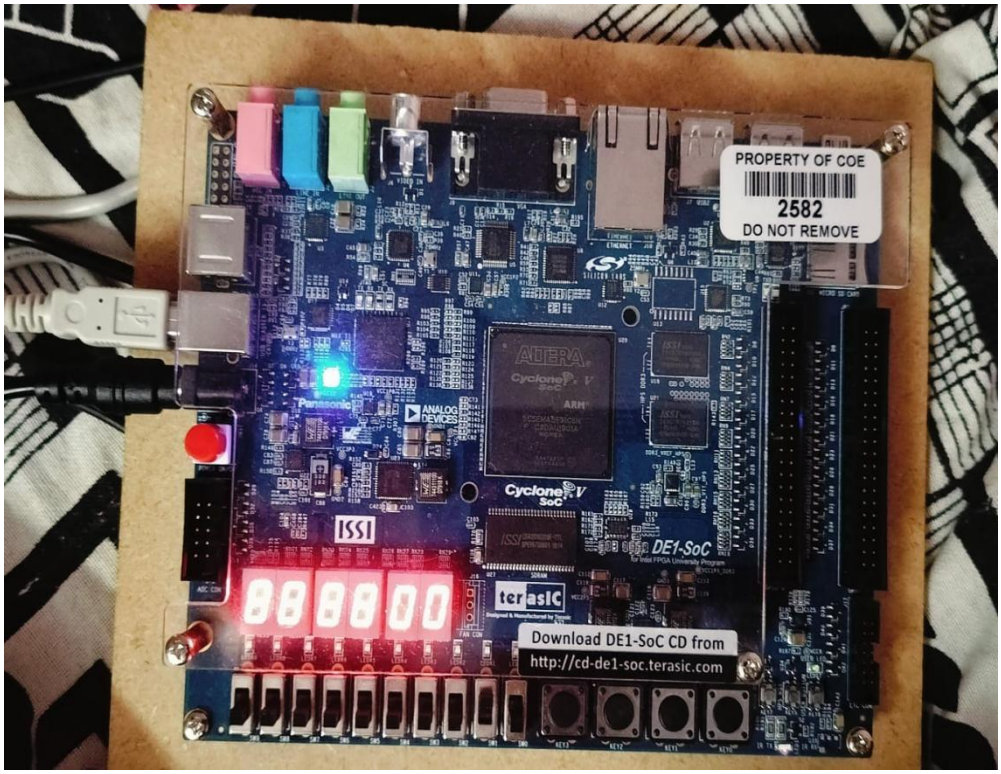


Figure 48: FPGA board after programming .sof file

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.

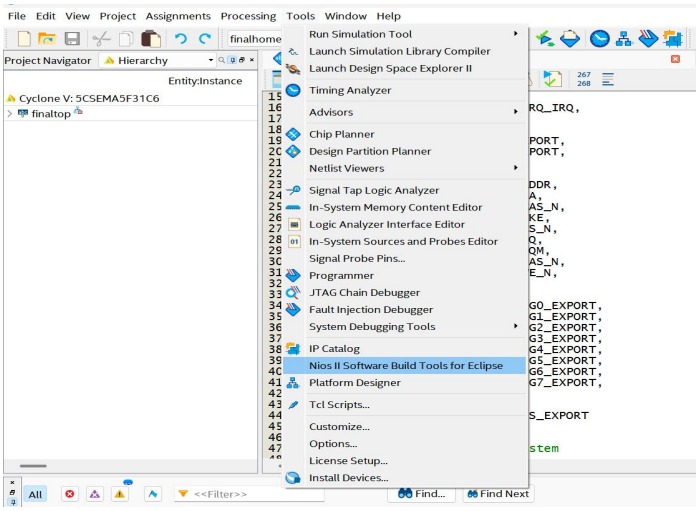


Figure 49: Opening Nios II Software Build Tools for Eclipse

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.

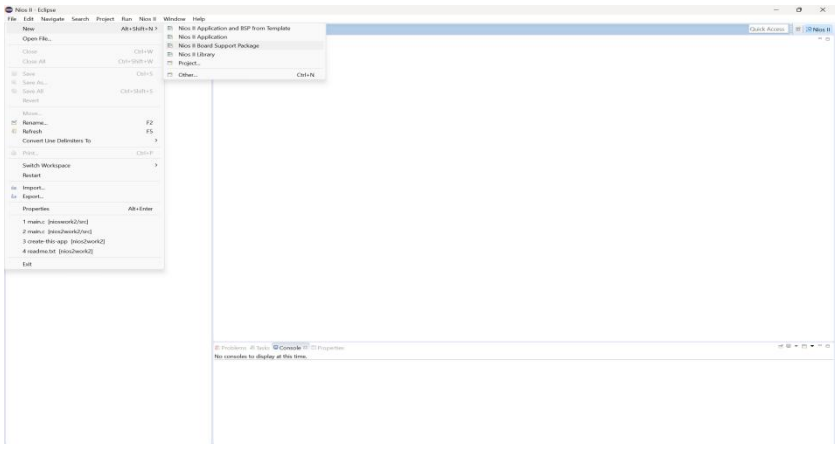


Figure 50: Creating new application project and the BSP

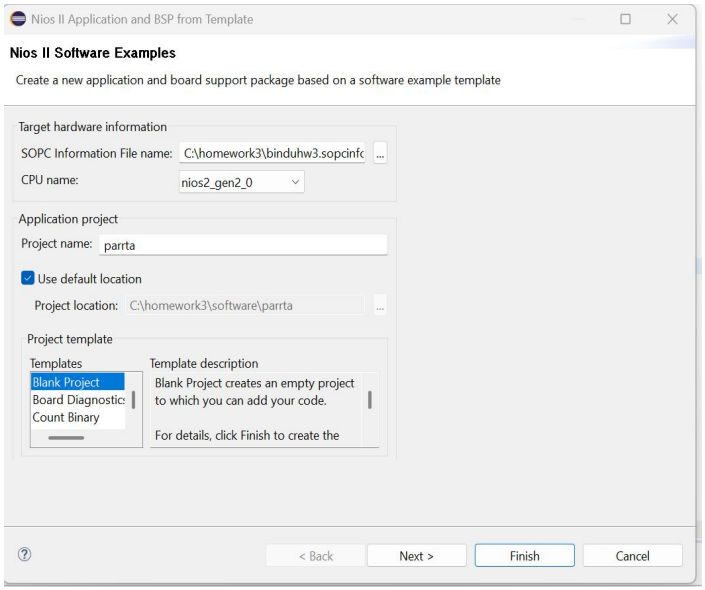


Figure 51: In the Nios II Application and BSP from Template window

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.

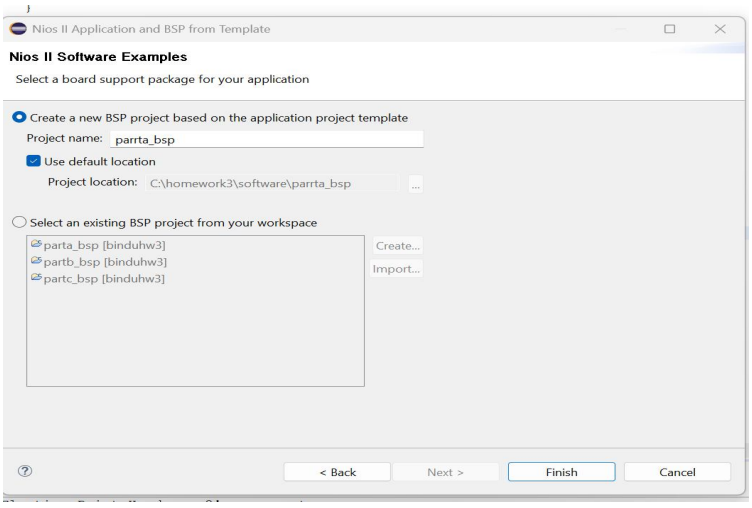
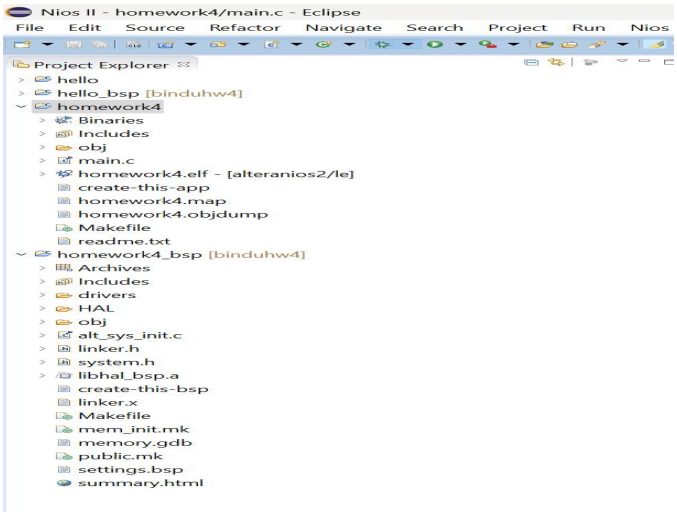


Figure 52: For creating bsp project

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.

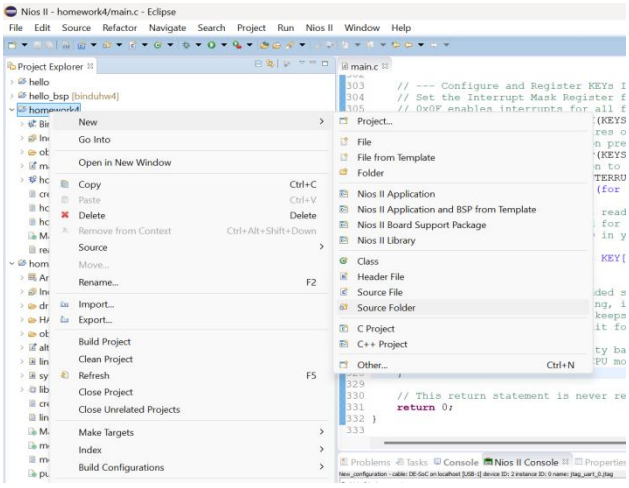


Figure 53: For creating source folder

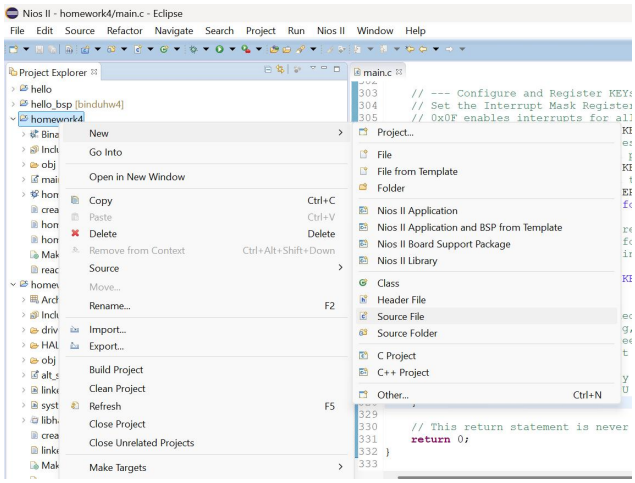


Figure 54: For creating source file

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 (name src file). This folder contains the custom application logic to be executed on the Nios II processor. Similarly, add main.c codes to all application projects.

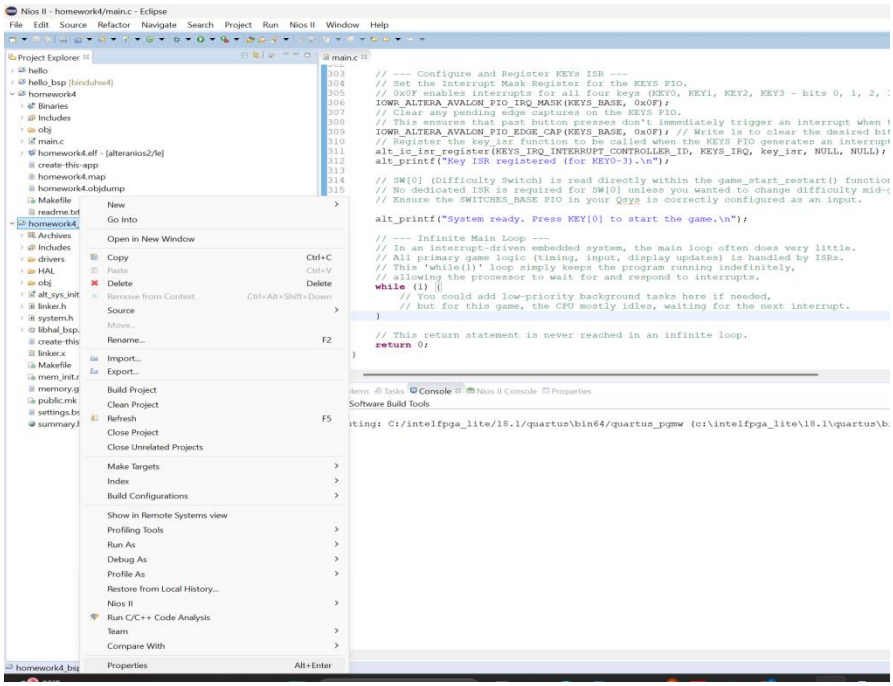


Figure 55: Right-clicking on the bsp project in Eclipse provides access to BSP settings.

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 56: Nios BSP properties for homework4

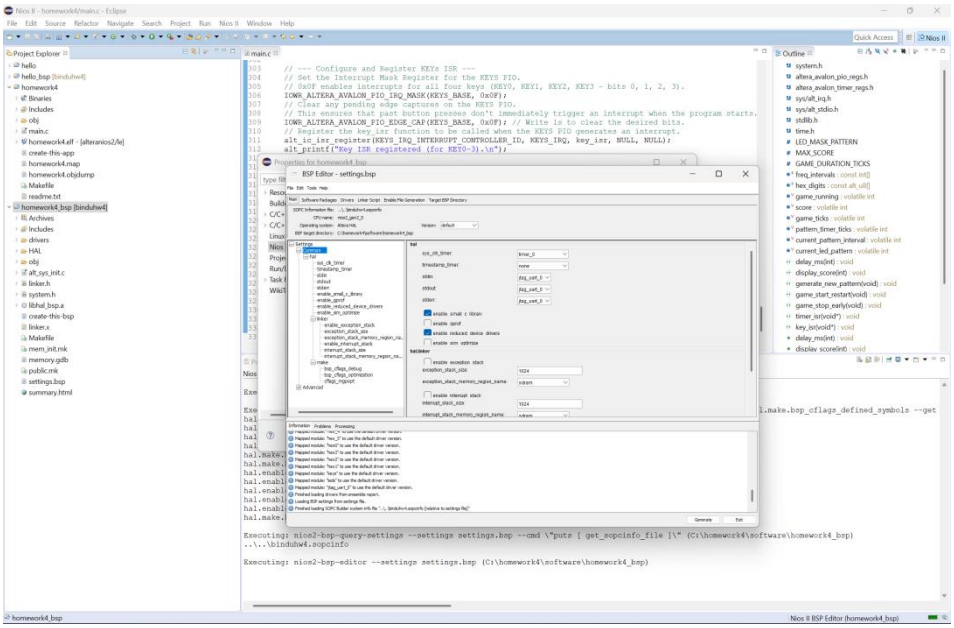


Figure 57: Nios BSP editor

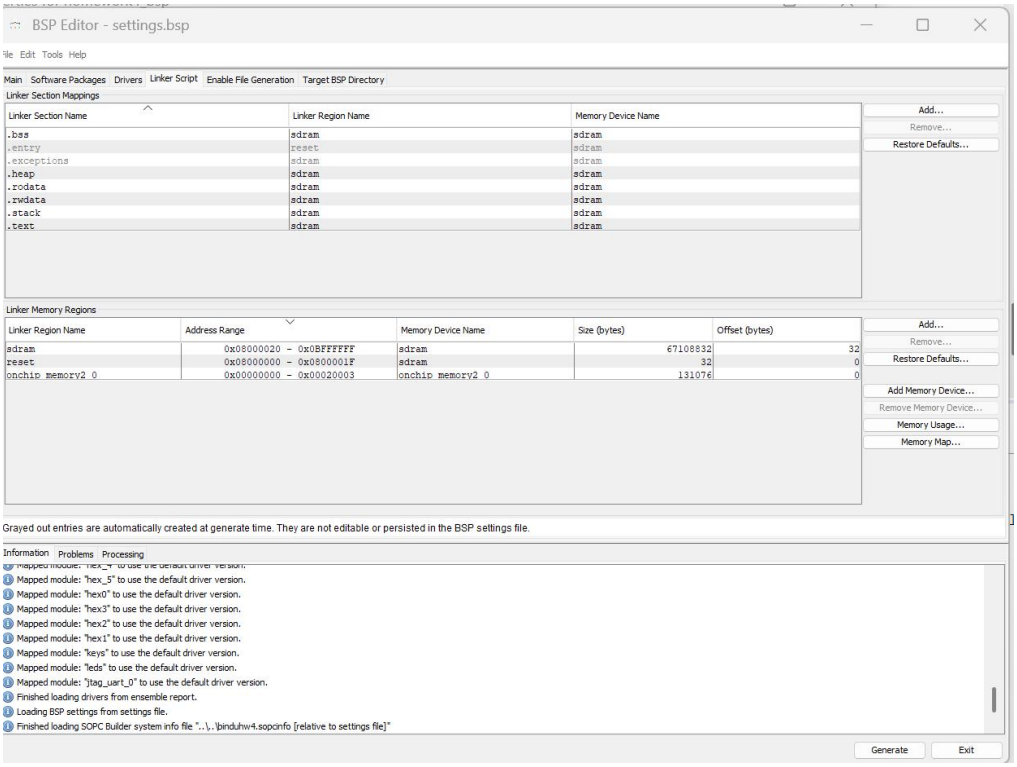


Figure 58: Nios II BSP linker scripts and click on generate

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

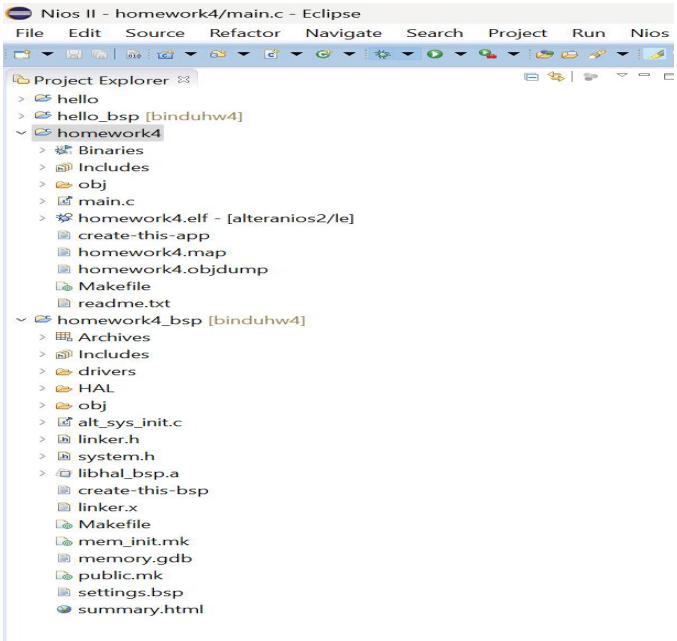
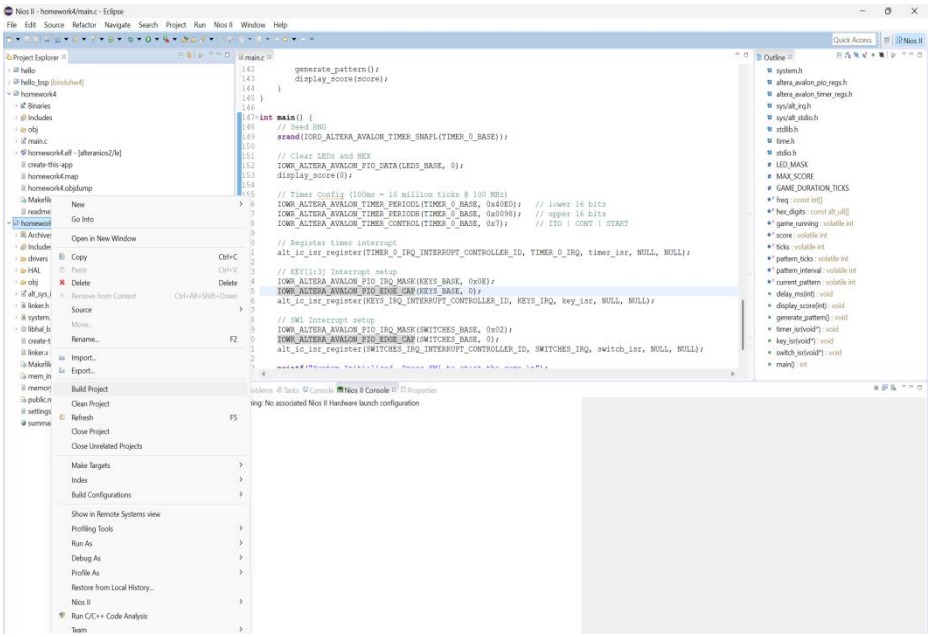


Figure 59: Artifacts of projects in Nios II

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.



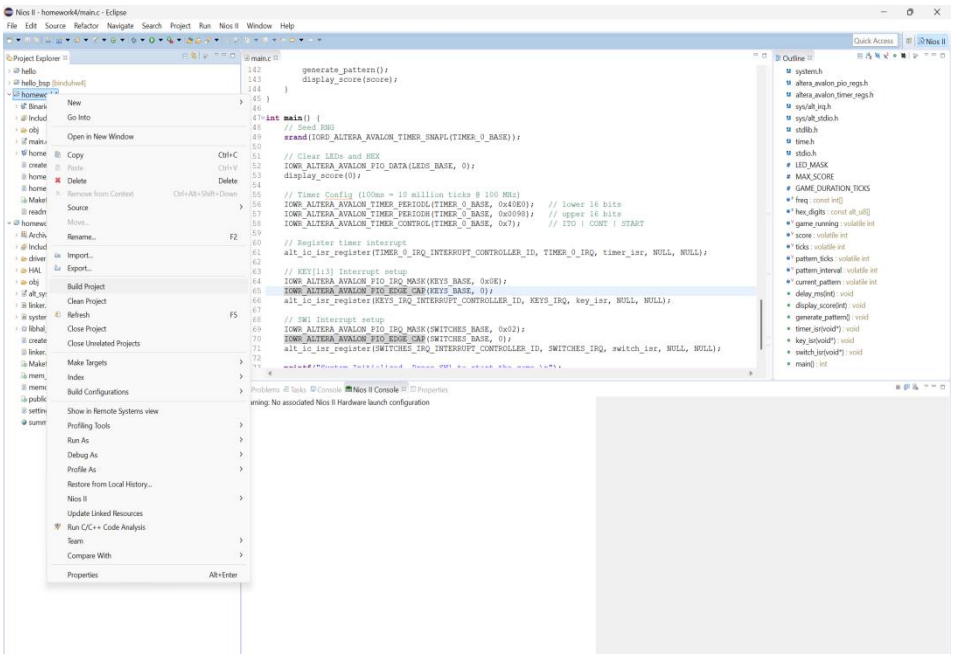


Figure 60: Building both projects

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

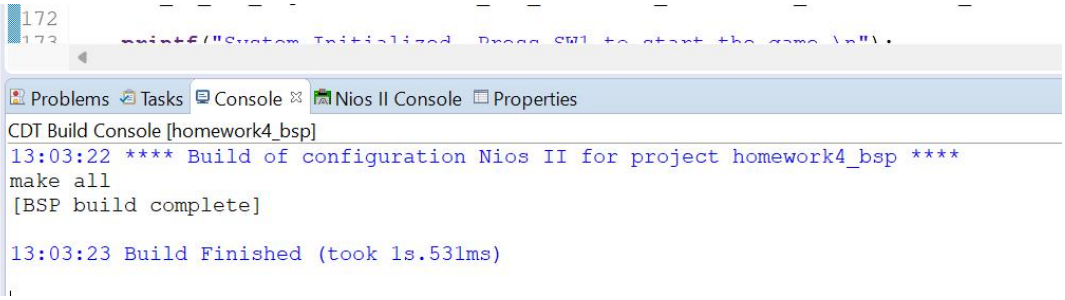


Figure 61: Build complete succesfully for bsp homework4 project

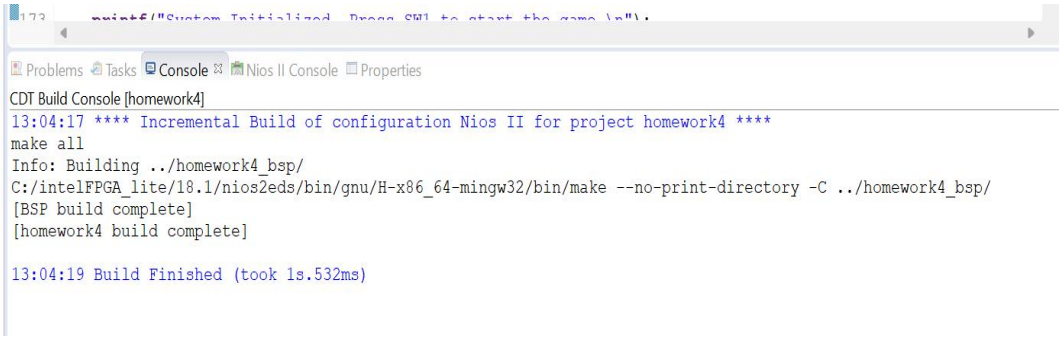


Figure 62: Build complete succesfully for application part a project

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.

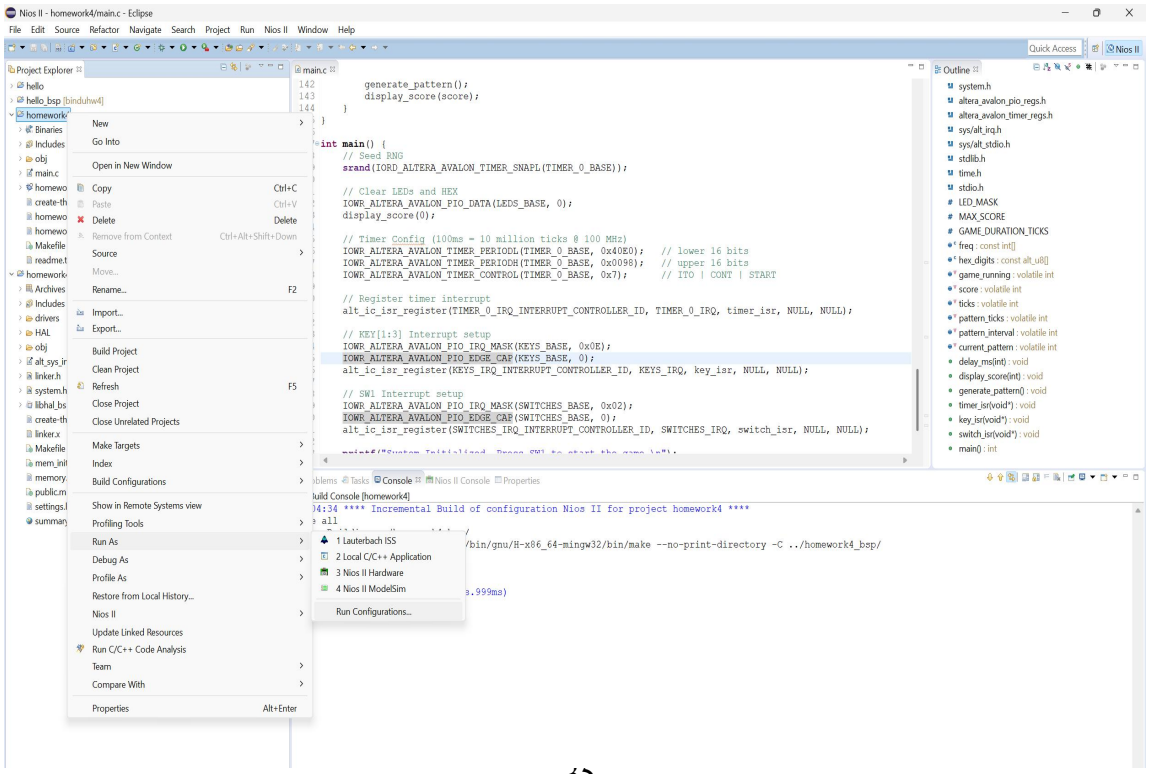


Figure 63: Run options

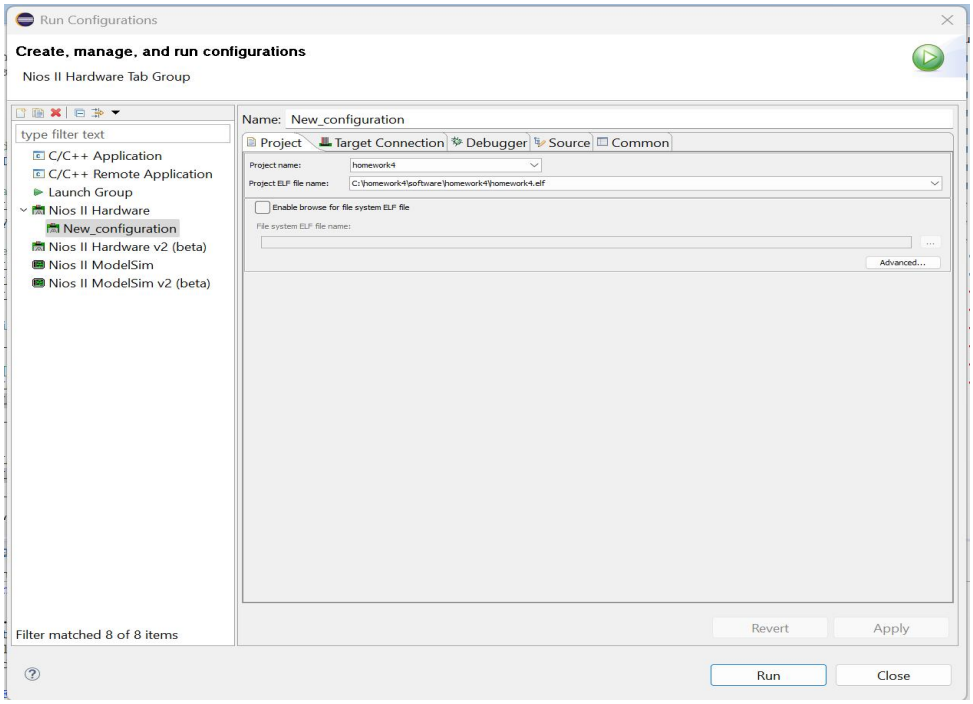


Figure 63: Run Configurations – Project Tab

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

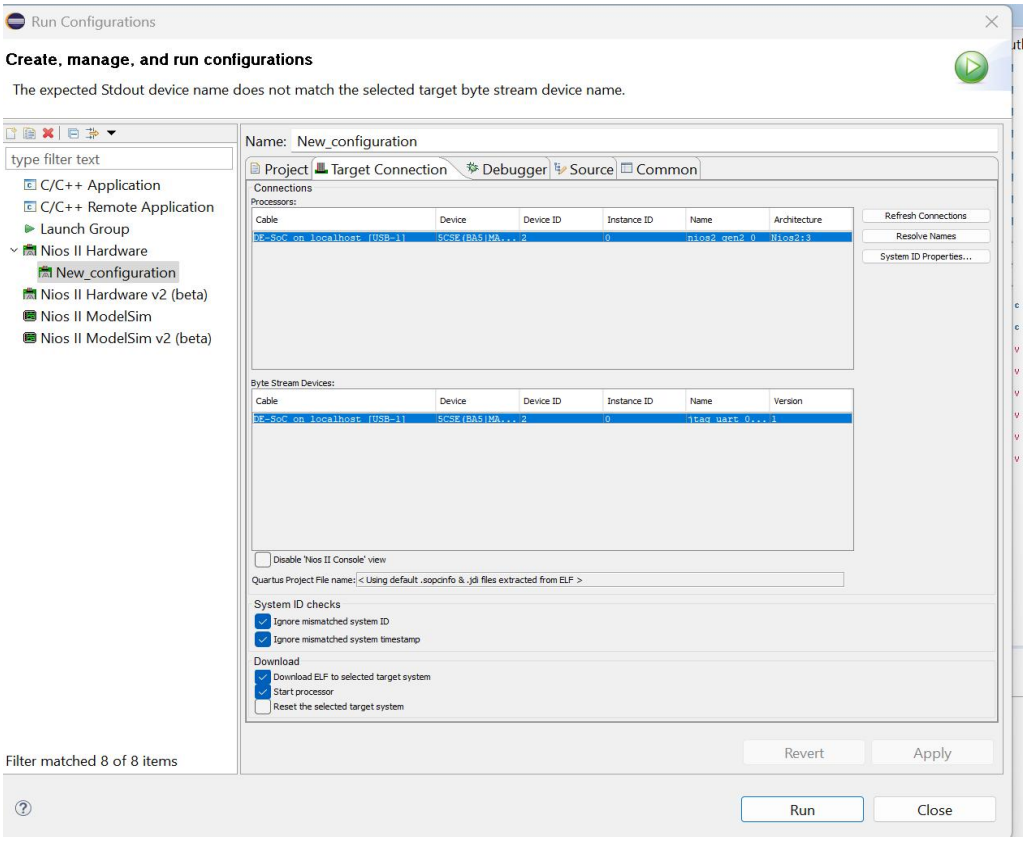


Figure 64: Run Configurations – Target Connection Tab

This is prior board state before hitting run

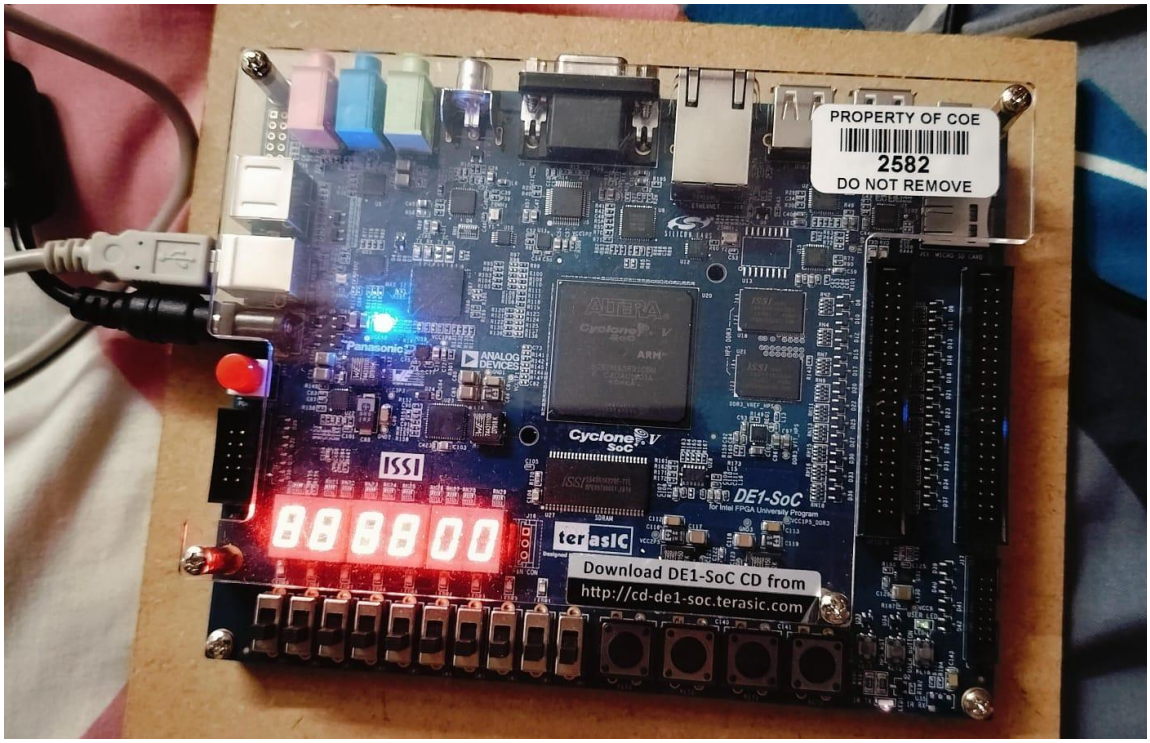


Figure 65: board after programming .sof file and before hitting run in ecllipse

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.

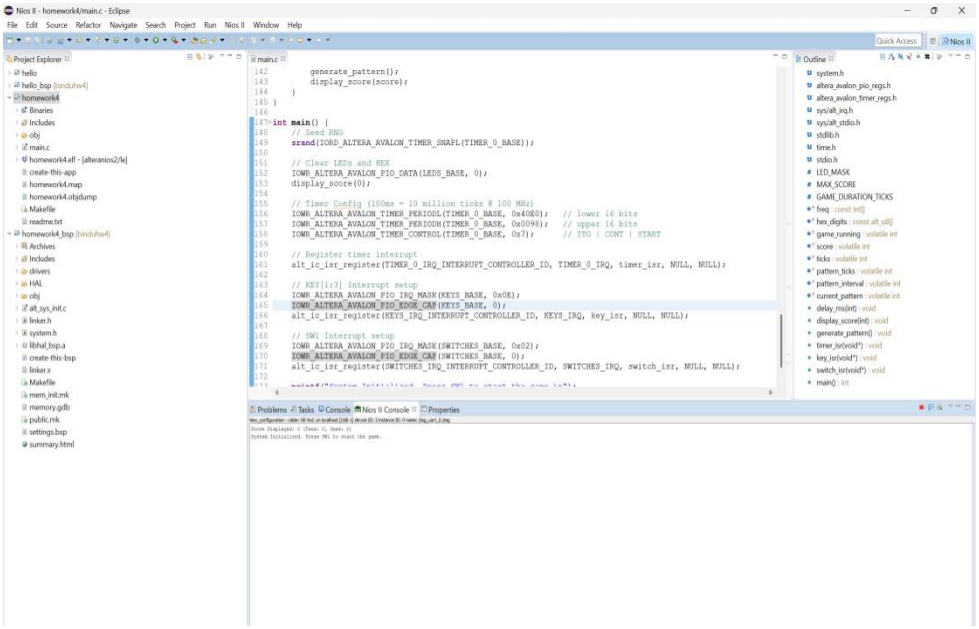


Figure 65: Eclipse IDE – Nios II Console and Source Code

After turning on SW1, I could see the ticks noted in nios console.

Problems Tasks Console Nios II Console Properties

New_configuration - cable: DE-SoC on localhost [USB-1] device ID: 2 instance ID: 0 name: jtag_uart_0.jtag

```
Score displayed: 0
System Initialized. Press SW1 to start the game.

=== GAME STARTED ===
Selected Difficulty (SW0): 0 (0=Easy, 1=Hard)
Pattern Interval set to 10 ticks
New pattern generated: 0x9
Score displayed: 0

=== GAME STARTED ===
Selected Difficulty (SW0): 0 (0=Easy, 1=Hard)
Pattern Interval set to 10 ticks
New pattern generated: 0x3
Score displayed: 0
Timeout. Penalty applied. Score: 0
Score displayed: 0
```

Figure 66: Eclipse IDE – Nios II Console

Also, on board I could see this response that leds were turned on. Leds are toggling

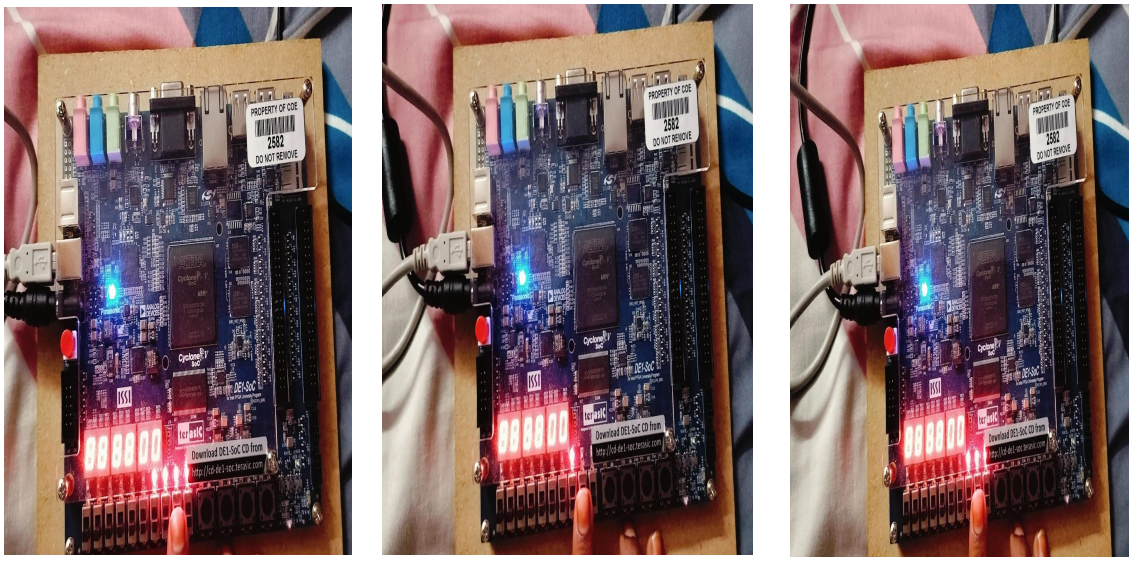


Figure 67: turn on SW1 on board and LEDs get toggle and a pattern is seen on board

Pressing key [0] on board, it turned off the LEDs

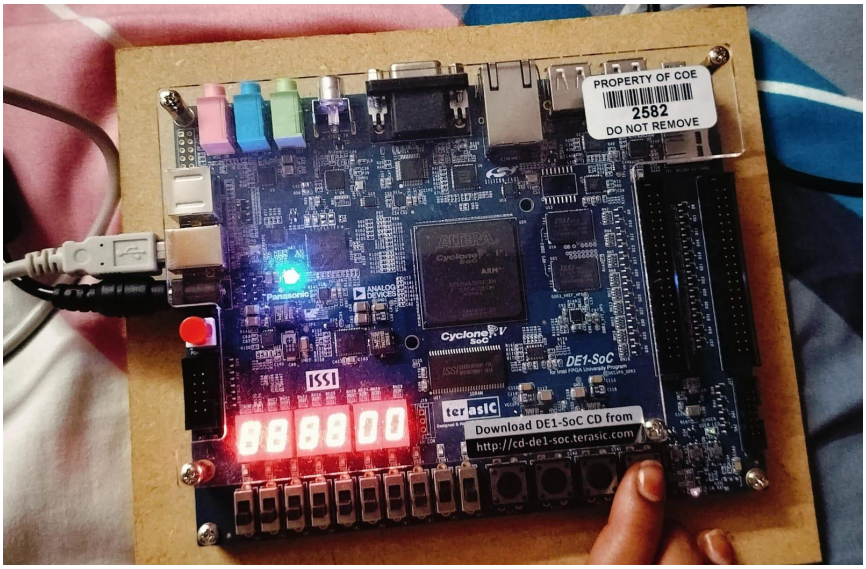


Figure 68 : press key 0 and no led response and game restarts

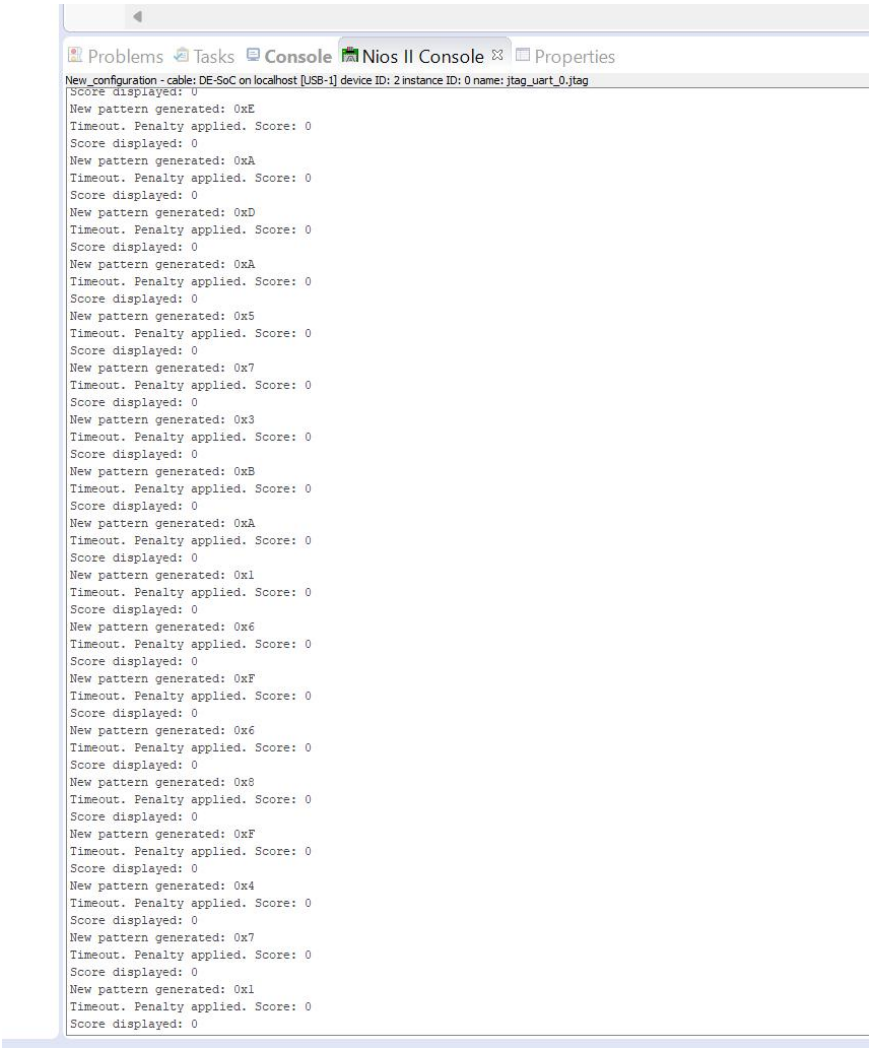


Figure 69: Eclipse - Nios II IDE

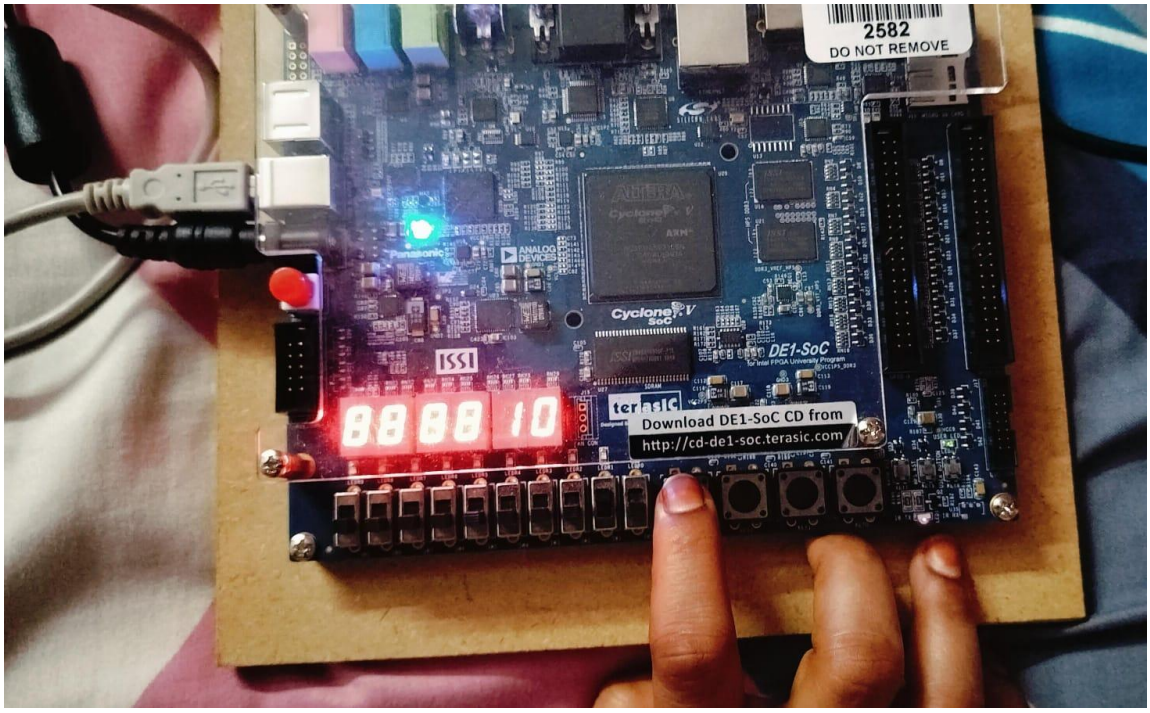


Figure 70: Correct move adds score to the game

```
Pattern Interval set to 10 ticks
New pattern generated: 0x3
Score displayed: 0
Timeout. Penalty applied. Score: 0
Score displayed: 0
New pattern generated: 0xF
Key press detected. Pressed pattern: 0x8, Expected pattern: 0xF
Incorrect! Penalty. Score: 0
Score displayed: 0
New pattern generated: 0x1
Key press detected. Pressed pattern: 0x8, Expected pattern: 0x1
Incorrect! Penalty. Score: 0
Score displayed: 0
Key press detected. Pressed pattern: 0x8, Expected pattern: 0x0
No pattern to match. Ignoring key press.
New pattern generated: 0x4
Key press detected. Pressed pattern: 0x8, Expected pattern: 0x4
Incorrect! Penalty. Score: 0
Score displayed: 0
Key press detected. Pressed pattern: 0x8, Expected pattern: 0x0
No pattern to match. Ignoring key press.
New pattern generated: 0x8
Key press detected. Pressed pattern: 0x8, Expected pattern: 0x8
Correct! Score: 10
Score displayed: 10
New pattern generated: 0x7
Key press detected. Pressed pattern: 0x8, Expected pattern: 0x7
Incorrect! Penalty. Score: 5
Score displayed: 5
New pattern generated: 0xC
Timeout. Penalty applied. Score: 0
Score displayed: 0
New pattern generated: 0x5
```

Figure 71: Nios console displays patterns and scores

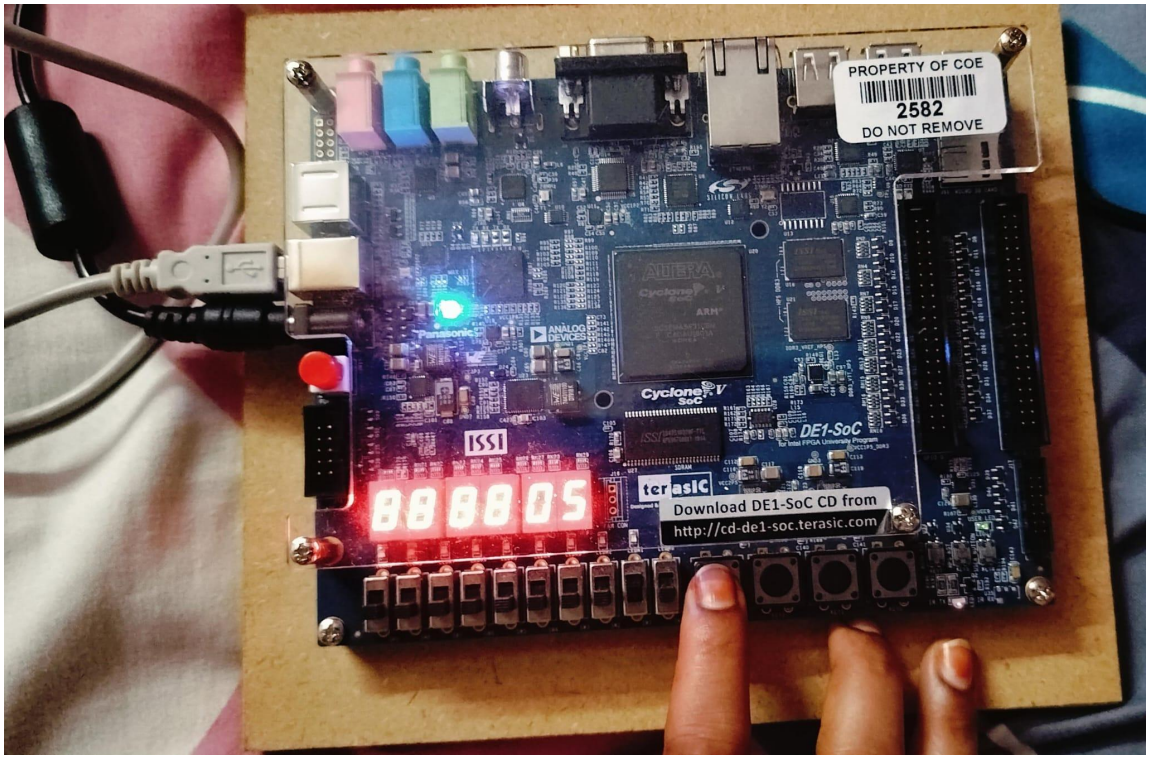


Figure 72: penalty score 5 deducted in next move

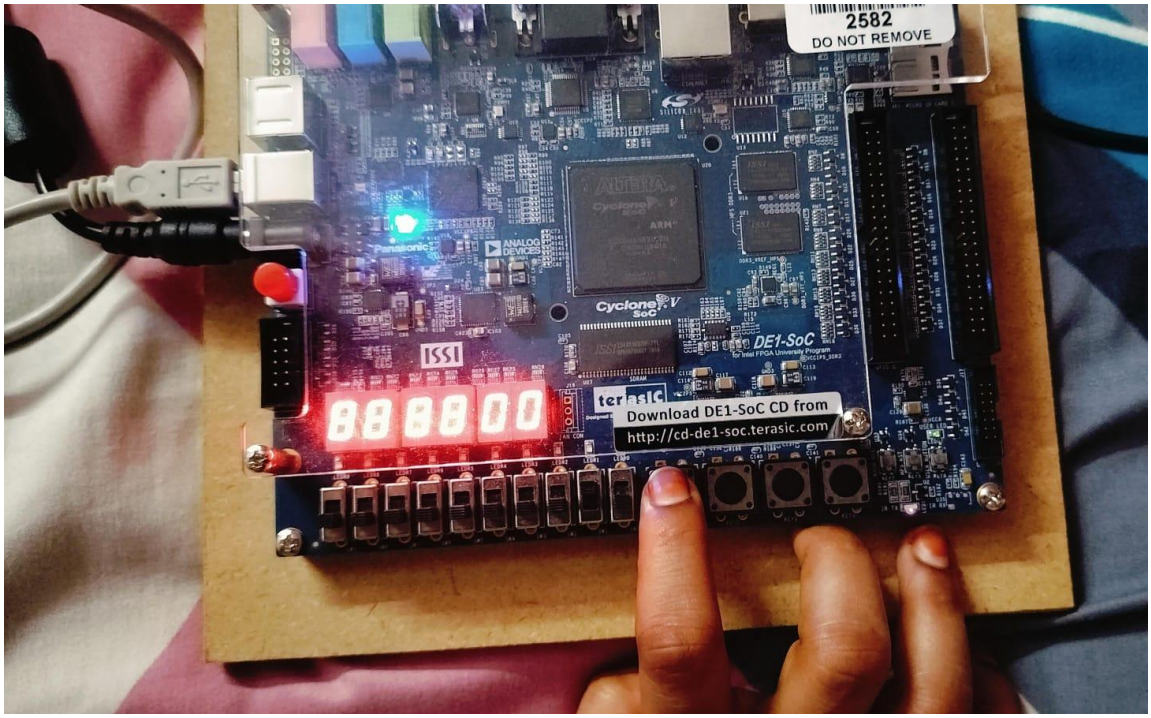


Figure 73: Game score after making moves

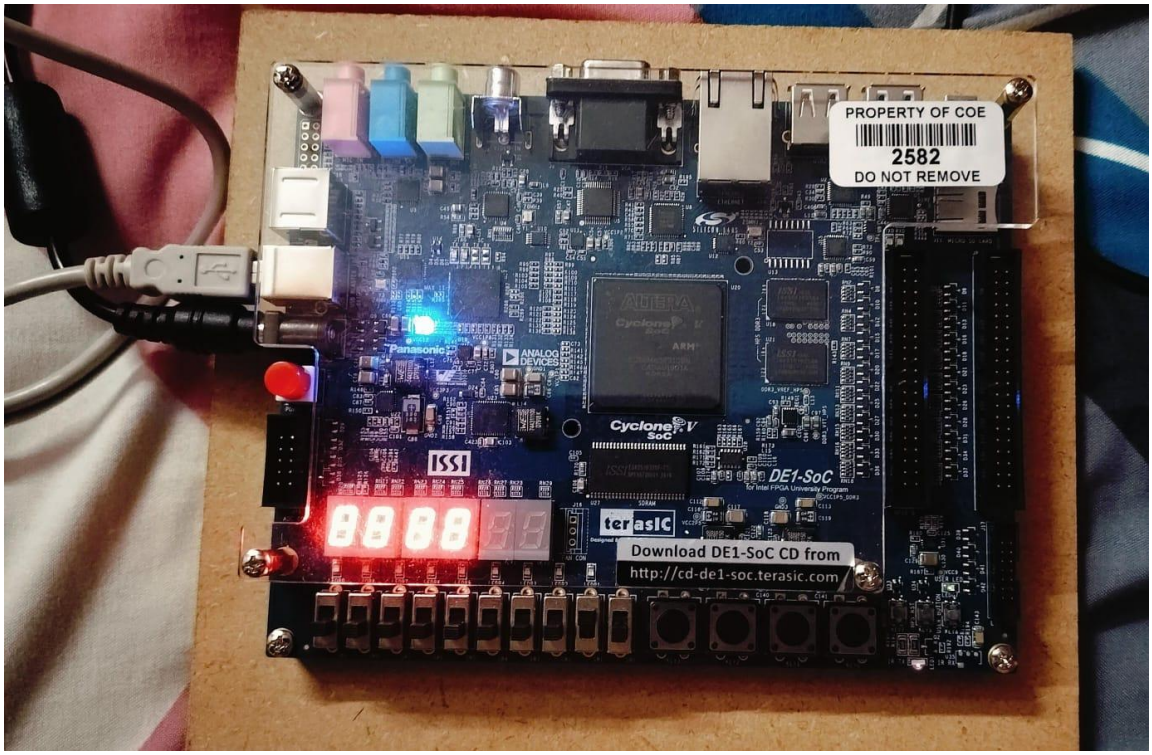


Figure 74: Game is ended and no score on hex display

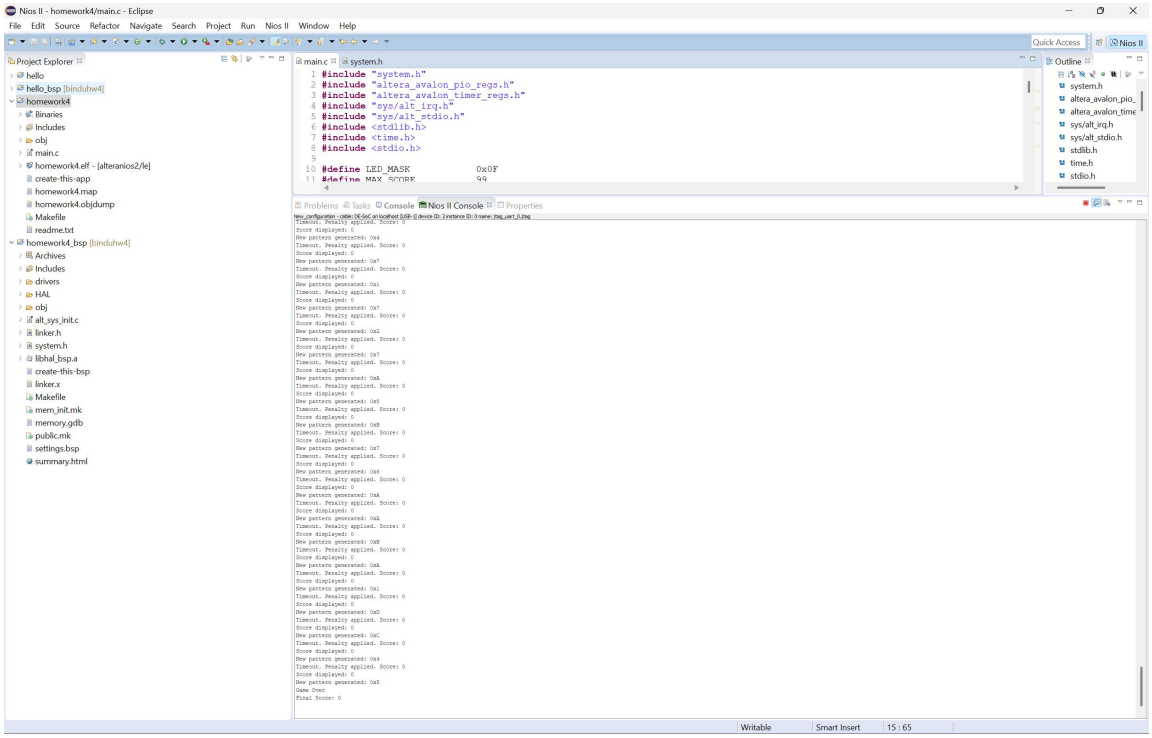


Figure 70: Ecllipse nios shows game ended for user

7 Analysis

Purpose:

The main goal for this Homework 4 was to design and build a complete reaction game on the Intel DE1-SoC FPGA. This project really pushed me to combine hardware and software design, using Platform Designer to set up the custom Nios II system and then writing the C code to make the game work. It was all about getting the LEDs, buttons and timers to talk to each other through the FPGA, making sure everything was driven by interrupts for a good game.

What I Learned:

Working on this assignment taught me a lot about how embedded systems come together. I got hands-on experience with the entire flow, from laying out the hardware components in Platform Designer and compiling everything in Quartus, to writing the actual game logic in Nios II Eclipse. Understanding how the Board Support Package connects the C code to the custom hardware. Plus, getting the interrupts for the timer, keys and switches to work perfectly showed me how to build efficient, event-driven systems that react instantly.

Analysis of Output:

When I finally got the game running, it worked just as planned, which was happy to see. Starting the game with SW1 reset the score and immediately showed the first LED pattern and changing SW0 clearly affected how fast the patterns appeared, proving the difficulty selection worked. The button presses were spot-on; the game reacted instantly, updating the score correctly for hits and applying penalties for misses or timeouts. The timer kept perfect track of the 60-second game and the console output in Eclipse was a lifesaver, showing me exactly what was happening inside the system at every step, which was super helpful for debugging and confirming everything was running smoothly.

8 Results

The development and implementation of the interrupt-driven reaction game on the Intel DE1-SoC FPGA were successfully completed. The custom Nios II embedded system, designed in Platform Designer, integrated seamlessly with the C application code developed in Nios II SBT for Eclipse. All core functionalities were verified: the game initiated correctly via SW1, displayed random LED patterns, processed player input from the push-buttons with accurate scoring (+10 for correct, -5 for incorrect/timeout) and managed game duration via the interval timer. Difficulty selection using SW0 also functioned as intended, altering pattern display speeds. Real-time console output confirmed the system's responsiveness and the precise execution of game logic, demonstrating a fully functional and interactive embedded reaction game.

9 Reflection on the Software Development

What I Learned in homework

The software development for this homework was a crucial part of the project that truly tied everything together. I learned that designing a hardware system in Platform Designer is only the first step; the real challenge and satisfaction came from writing the C code that brought it to life.

Understanding how to use the hal to control the LEDs and read button inputs was a huge learning curve. The interrupt-driven approach was a major takeaway, teaching me that instead of constantly checking for button presses, a well-designed system can efficiently react only when an event occurs. Finally, using the JTAG UART console for real-time debugging was incredibly valuable, as it allowed me to "see" inside the program and confirm that every piece of logic, from score updates to timer ticks, 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 system in Platform Designer and another to correctly link up the interrupt vectors and PIO base addresses in the C code. Getting the interrupt service routines to fire correctly and reliably for both the timer and the keys required a lot of careful register configuration and testing. I also had to contend with the practical issue of switch bounce, which was an unexpected challenge. Implementing a simple software delay within the ISR to debounce the buttons was a critical step to ensure a single physical press didn't trigger multiple interrupts. Ultimately, working through these issues provided a much deeper understanding of how the various layers of the embedded system interact.

10 Conclusion

In conclusion, this project successfully designed and implemented a fully functional, interrupt-driven reaction game on the Intel DE1-SoC FPGA. By leveraging the comprehensive capabilities of Platform Designer, Quartus Prime and the Nios II SBT for Eclipse, a robust embedded system was created that seamlessly integrated custom hardware with application-level software. All game features including real-time score tracking, dynamic pattern generation and timed gameplay were successfully verified. The experience underscored the importance of a well-structured hardware-software co-design approach and demonstrated the power of an interrupt-driven model for building responsive and efficient embedded systems. This project served as a valuable and practical exercise in FPGA-based embedded system development.

11 Appendix

Top level module

```
module homework4top (
    input  wire       CLOCK_50,  // 50 MHz system clock input.
    input  wire [3:0]  KEY,      // 4 push buttons (KEY[3:0]).
            input wire  [9:0]  SW,       // 10 slide switches (SW[9:0]).
    output wire [9:0]  LEDR,     // 10 red LEDs (LEDR[9:0]).
    output wire [6:0]  HEX0,     // 7-segment display 0.
    output wire [6:0]  HEX1,     // 7-segment display 1.
    output wire [6:0]  HEX2,     // 7-segment display 2.
    output wire [6:0]  HEX3,     // 7-segment display 3.
    output wire [6:0]  HEX4,     // 7-segment display 4.
    output wire [6:0]  HEX5,     // 7-segment display 5.

    // SDRAM interface signals.
    output wire [12:0] DRAM_ADDR,    // SDRAM Address.
    output wire [1:0]  DRAM_BA,      // SDRAM Bank Address.
    output wire        DRAM_CAS_N,   // SDRAM Column Address Strobe.
    output wire        DRAM_CKE,     // SDRAM Clock Enable.
    output wire        DRAM_CS_N,    // SDRAM Chip Select.
    inout  wire [15:0] DRAM_DQ,      // SDRAM Data bus.
    output wire        DRAM_LDQM,    // SDRAM Lower Data Mask.
    output wire        DRAM_RAS_N,   // SDRAM Row Address Strobe.
    output wire        DRAM_UDQM,    // SDRAM Upper Data Mask.
    output wire        DRAM_WE_N,    // SDRAM Write Enable.
    output wire        DRAM_CLK      // SDRAM Clock output.
);

    wire pll_locked; // Wire for PLL lock status.

    // Instantiation of the Qsys-generated system module.
    binduhw4 u0 (
        .pll_ref_clk_clk         (CLOCK_50),          // Connects 50 MHz clock to PLL.
        .pll_ref_reset_reset     (~KEY[0]),           // Inverted KEY[0] as active-low system reset.

        // I/O connections to Qsys system.
        .keys_external_connection_export    (KEY),          // Push buttons.
        .leds_external_connection_export    (LEDR),         // Red LEDs.
        .hex0_export             (HEX0),              // HEX0 display.
        .hex1_export             (HEX1),              // HEX1 display.
        .hex2_export             (HEX2),              // HEX2 display.
        .hex3_export             (HEX3),              // HEX3 display.
        .hex4_export             (HEX4),              // HEX4 display.
        .hex5_export             (HEX5),              // HEX5 display.
        .switches_export         (SW),                // Slide switches.

        // SDRAM connections to Qsys 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}),   // Concatenate DQM
signals.
        .sdram_wire_ras_n       (DRAM_RAS_N),
        .sdram_wire_we_n        (DRAM_WE_N),
        .pll_sdram_clk_clk      (DRAM_CLK)              // PLL-generated SDRAM clock.
        // .sdram_pll_locked_export (pll_locked) // PLL locked signal
    );

endmodule
```

Main.c code

```
#include "system.h"                 // Includes hardware definitions
generated by Qsys.
#include "altera_avalon_pio_regs.h" // Provides macros for Parallel I/O
(PIO) register access.
#include "altera_avalon_timer_regs.h" // Provides macros for timer
register access.
#include "sys/alt_irq.h"            // Functions for interrupt handling.
#include "sys/alt_stdio.h"          // Lightweight standard I/O for
Nios II console.
#include <stdlib.h>                 // Standard library for general
utilities, including random number generation.
#include <time.h>                   // For time-related functions, used
to seed the random number generator.
#include <stdio.h>                  // Standard I/O functions, like
printf.

// --- Game Configuration Constants ---
#define LED_MASK            0x0F    // Mask to target the lower 4 bits
of the LEDs (LEDR[3:0]).
#define MAX_SCORE           99      // Maximum score displayable on two
7-segment digits.
#define GAME_DURATION_TICKS 600     // Total game duration in 100ms
ticks (60 seconds).
#define DEBOUNCE_DELAY_MS   50      // Millisecond delay for button
debouncing.

const int freq[3] = {10, 5, 3};     // Possible pattern intervals in
100ms ticks (1s, 0.5s, 0.3s).

// 7-segment display patterns for digits 0-9 (common-anode).
const alt_u8 hex_digits[10] = {
    0xC0, 0xF9, 0xA4, 0xB0, 0x99,
    0x92, 0x82, 0xF8, 0x80, 0x90
};

// --- Global Volatile Variables ---
// 'volatile' keyword ensures these variables are always read
from/written to memory,
// preventing compiler optimizations that could affect ISR interaction.
volatile int game_running = 0;      // Flag: 1 if game is active, 0
otherwise.
volatile int score = 0;             // Current player's score.
volatile int ticks = 0;             // Counter for total 100ms game
ticks.
```

```
volatile int pattern_ticks = 0;      // Counter for 100ms ticks since
the last LED pattern change.
volatile int pattern_interval = 10; // Current duration (in 100ms ticks)
an LED pattern is displayed.
volatile int current_pattern = 0;    // The binary pattern currently
displayed on LEDs (0 if no pattern).

// --- Utility Functions ---
// Simple blocking delay function for a specified number of ms.
void delay_ms(int ms) {
    volatile int i;
    for (i = 0; i < ms * 10000; ++i); // Approximate loop count for
delay.
}
// Displays the current score on the HEX1 and HEX0 7-segment displays.
void display_score(int s) {
    int ones = s % 10;           // Extract the ones digit.
    int tens = (s / 10) % 10;    // Extract the tens digit.
    // Write the corresponding hex digit patterns to the PIO bases for
HEX0 and HEX1.
    IOWR_ALTERA_AVALON_PIO_DATA(HEX0_BASE, hex_digits[ones]);
    IOWR_ALTERA_AVALON_PIO_DATA(HEX1_BASE, hex_digits[tens]);
    printf("Score displayed: %d\n", s); // Output score to console for
debugging.
}

// Generates a new random pattern for LEDR[3:0] and displays it.
void generate_pattern() {
    // Loop to ensure the generated pattern is not 0 (all LEDs off).
    do {
        current_pattern = (rand() % 15) + 1; // Generates a random
value from 1 to 15 (0x01-0x0F).
    } while(current_pattern == 0);
    IOWR_ALTERA_AVALON_PIO_DATA(LEDS_BASE, current_pattern); // Write
the pattern to the LED PIO.
    printf("New pattern generated: 0x%X\n", current_pattern); // Log
new pattern to console.
}

// --- Interrupt Service Routines (ISRs) ---

// Timer Interrupt Service Routine (ISR) for TIMER_0.
// This ISR is triggered every 100ms and manages game timing, penalties,
and pattern generation.
void timer_isr(void* context) {
    IOWR_ALTERA_AVALON_TIMER_STATUS(TIMER_0_BASE, 0); // Clear the
timer's interrupt status bit.

    if (!game_running) return; // Exit if the game is not active.

    ticks++;        // Increment overall game duration counter.
    pattern_ticks++; // Increment counter for current pattern's display
time.

    // Check if the total game duration has elapsed.
    if (ticks >= GAME_DURATION_TICKS) {
        game_running = 0; // Stop the game.
```

```
        alt_printf("Game Over\n"); // Display "Game Over" on console.
        printf("Final Score: %d\n", score); // Show final score.
        IOWR_ALTERA_AVALON_PIO_DATA(LEDS_BASE, 0); // Turn off all LEDs.
        // Display 'OFF' state on HEX displays (all segments on).
        IOWR_ALTERA_AVALON_PIO_DATA(HEX0_BASE, 0xFF);
        IOWR_ALTERA_AVALON_PIO_DATA(HEX1_BASE, 0xFF);
        return; // Exit ISR.
    }

    // Check if the current pattern has timed out.
    if (pattern_ticks >= pattern_interval) {
        // Apply penalty only if a pattern was actively displayed and
missed.
        if (current_pattern != 0) {
            score -= 5; // Deduct points for timeout.
            if (score < 0) score = 0; // Ensure score doesn't go below
zero.
            printf("Timeout. Penalty applied. Score: %d\n", score);
            display_score(score); // Update displayed score.
        }

        generate_pattern(); // Generate and display a new pattern.
        pattern_ticks = 0;  // Reset pattern timer for the new pattern.
    }
}

// Push Button (KEY[0:3]) Interrupt Service Routine (ISR).
// Triggered when any of the KEY[0:3] buttons are pressed.
void key_isr(void* context) {
    // Clear the edge capture register to acknowledge the interrupt.
    int edge_capture = IORD_ALTERA_AVALON_PIO_EDGE_CAP(KEYS_BASE);
    IOWR_ALTERA_AVALON_PIO_EDGE_CAP(KEYS_BASE, 0);

    if (!game_running) { // Ignore key presses if the game is not
active.
        return;
    }

    delay_ms(DEBOUNCE_DELAY_MS); // Apply software debounce delay.

    int key_state = IORD_ALTERA_AVALON_PIO_DATA(KEYS_BASE); // Read
current state of all keys.
    // Convert active-low key state to active-high pattern for bits 0-3.
    int pressed_pattern = (~key_state) & 0x0F;

    if (pressed_pattern == 0) { // Ignore if no key is currently
pressed after debouncing.
        return;
    }

    printf("Key press detected. Pressed pattern: 0x%X, Expected pattern:
0x%X\n", pressed_pattern, current_pattern);

    // Only process a key press if a pattern is currently displayed.
    if (current_pattern == 0) {
        printf("No pattern to match. Ignoring key press.\n");
        return;
    }
```

```
 // Check if the pressed pattern matches the expected LED pattern.
    if (pressed_pattern == current_pattern) {
        score += 10; // Correct match: add 10 points.
        if (score > MAX_SCORE) score = MAX_SCORE; // Clamp score at
MAX_SCORE.
        printf("Correct! Score: %d\n", score);
    } else {
        score -= 5; // Incorrect match: deduct 5 points.
        if (score < 0) score = 0; // Clamp score at 0.
        printf("Incorrect! Penalty. Score: %d\n", score);
    }

    display_score(score); // Update the score on the 7-segment displays.

    IOWR_ALTERA_AVALON_PIO_DATA(LEDS_BASE, 0); // Clear LEDs to
indicate response registered.
    current_pattern = 0; // Mark pattern as handled.
    pattern_ticks = 0; // Reset pattern timer for the next pattern.
}

// Switch (SW1) Interrupt Service Routine (ISR).
// Triggered by a change in state of SW1 to start/restart the game.
void switch_isr(void* context) {
    int edge = IORD_ALTERA_AVALON_PIO_EDGE_CAP(SWITCHES_BASE); // Read
edge capture register.
    IOWR_ALTERA_AVALON_PIO_EDGE_CAP(SWITCHES_BASE, 0); // Clear the
edge capture flag.

    if (edge & 0x02) { // Check if the interrupt was triggered by SW1
(bit 1).
        game_running = 1; // Set game to active.
        ticks = 0;        // Reset overall game timer.
        score = 0;        // Reset score.
        pattern_ticks = 0; // Reset pattern timer.

        // Read SW0 (bit 0) to determine difficulty (0=Easy, 1=Hard).
        int difficulty_switch =
IORD_ALTERA_AVALON_PIO_DATA(SWITCHES_BASE) & 0x01;
        // Randomly select a base pattern interval from the 'freq'
array.
        pattern_interval = freq[rand() % 3];

        if (difficulty_switch) { // If hard mode is selected.
            pattern_interval /= 2; // Halve the pattern interval.
            if (pattern_interval == 0) pattern_interval = 1; // Ensure
interval is at least 1 tick.
        }

        printf("\n=== GAME STARTED ===\n"); // Console message for game
start.
        printf("Selected Difficulty (SW0): %d (0=Easy, 1=Hard)\n",
difficulty_switch);
        printf("Pattern Interval set to %d ticks\n", pattern_interval);

        generate_pattern(); // Generate and display the first game
pattern.
        display_score(score); // Display initial score (00).
    }
}
// --- Main Program ---
// The entry point of the application.
```

```c
int main() {
    // Seed the random number generator using the low 16 bits of the
timer's current snapshot.
    // This provides a somewhat unique seed each time the board is
reset.
    srand(IORD_ALTERA_AVALON_TIMER_SNAPL(TIMER_0_BASE));

    IOWR_ALTERA_AVALON_PIO_DATA(LEDS_BASE, 0); // Ensure all LEDs are
off at startup.
    display_score(0); // Initialize HEX displays to '00'.

    // Timer 0 Configuration: Set period and control register for 100ms
interrupts.
    IOWR_ALTERA_AVALON_TIMER_PERIODL(TIMER_0_BASE, 0x4240); // Lower 16
bits of period.
    IOWR_ALTERA_AVALON_TIMER_PERIODH(TIMER_0_BASE, 0x0098); // Upper 16
bits of period.
    IOWR_ALTERA_AVALON_TIMER_CONTROL(TIMER_0_BASE, 0x7);    // Enable
timer, continuous mode, interrupt-on-timeout.

    // Register the timer_isr with the interrupt controller.
    alt_ic_isr_register(TIMER_0_IRQ_INTERRUPT_CONTROLLER_ID,
TIMER_0_IRQ, timer_isr, NULL, NULL);

    // KEY[0:3] Interrupt Setup: Configure PIO for interrupts from push
buttons.
    IOWR_ALTERA_AVALON_PIO_IRQ_MASK(KEYS_BASE, 0x0F); // Enable
interrupt mask for KEY[0:3].
    IOWR_ALTERA_AVALON_PIO_EDGE_CAP(KEYS_BASE, 0);    // Clear any
pending edge capture flags.
    alt_ic_isr_register(KEYS_IRQ_INTERRUPT_CONTROLLER_ID, KEYS_IRQ,
key_isr, NULL, NULL); // Register key_isr.

    // SW1 Interrupt Setup: Configure PIO for interrupt from Switch 1.
    IOWR_ALTERA_AVALON_PIO_IRQ_MASK(SWITCHES_BASE, 0x02); // Enable
interrupt mask for SW1 only.
    IOWR_ALTERA_AVALON_PIO_EDGE_CAP(SWITCHES_BASE, 0);    // Clear any
pending edge capture flags.
    alt_ic_isr_register(SWITCHES_IRQ_INTERRUPT_CONTROLLER_ID,
SWITCHES_IRQ, switch_isr, NULL, NULL); // Register switch_isr.

    printf("System Initialized. Press SW1 to start the game.\n"); //
Inform user how to start.

    while (1); // Infinite loop: The program idles here while ISRs
handle all game logic.
}
```

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 = 23:23:51  July 17, 2025
#
# -------------------------------------------------------------------------- #
#
# Notes:
#
# 1) The default values for assignments are stored in the file:
#                binduhw4_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 homework4top
set_global_assignment -name ORIGINAL_QUARTUS_VERSION 18.1.0
set_global_assignment -name PROJECT_CREATION_TIME_DATE "23:23:51  JULY 17,
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 LVTTL" -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 LVTTL" -to LEDR[9]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to LEDR[8]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to LEDR[7]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to LEDR[6]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -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 LVTTL" -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 LVTTL" -to LEDR[3]
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -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 LVTTL" -to LEDR[4]
set_instance_assignment -name IO_STANDARD "3.3-V LVCMOS" -to LEDR
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 binduhw4/synthesis/binduhw4.v
set_global_assignment -name QIP_FILE binduhw4/synthesis/binduhw4.qip
set_global_assignment -name VERILOG_FILE homework4top.v
set_instance_assignment -name PARTITION_HIERARCHY root_partition -to | -section_id Top
```