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

TECHNICAL REPORT

Assignment: Assignment 3

Course Title: ECE 278 - Embedded Systems

Instructor: Dr. Reza Raeisi

Student Name: Bindu Sree Chandu

INSTRUCTOR SECTION

Comments: _____

_____

_____

_____

Final Grade: _____

Contents page

1 Objective

.

The goal of this assignment is to create a embedded system using the DE1-SoC board and Nios II processor. To design and implement a comprehensive interval timer system demonstrating precise time measurement, timeout detection, and periodic event generation through low-level HAL API functions and polling techniques.

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
- 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 software was developed using Nios II SBT for Eclipse. The .sopcinfo file from Platform Designer was used to generate the BSP, providing the necessary HAL drivers.

The C program implemented the following objectives:

- **Part A: Stopwatch Mode**
    - **Action:** Measures how long KEY2 is held.
    - **Mechanism:** Polls KEY2 for press/release; uses alt_timestamp() for precise start/end time capture.
    - **Output:** Calculates elapsed microseconds; displays on HEX displays and LEDR (LSBs).
- **Part B: Timeout Detection**
    - **Action:** Detects 5 seconds of user inactivity.
    - **Mechanism:** Continuously monitors current timer value; resets 'last activity' timer on KEY3 press (polling).
    - **Output:** Lights LEDR9 if timeout occurs; KEY3 press clears LEDR9 and restarts timer.
- **Part C: Period Measurement and Repeat**
    - **Action:** Measures duration between two KEY1 presses and automatically repeats a cycle.
    - **Mechanism:** First KEY1 press stores time; second KEY1 press captures time and computes duration.
    - **Output:** Displays measured duration on HEX displays; blinks LEDR7 to indicate automatic 5-second cycle repetitions.

After building the project, the .elf file was loaded onto the DE1-SoC board and all functions were verified successfully.

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.

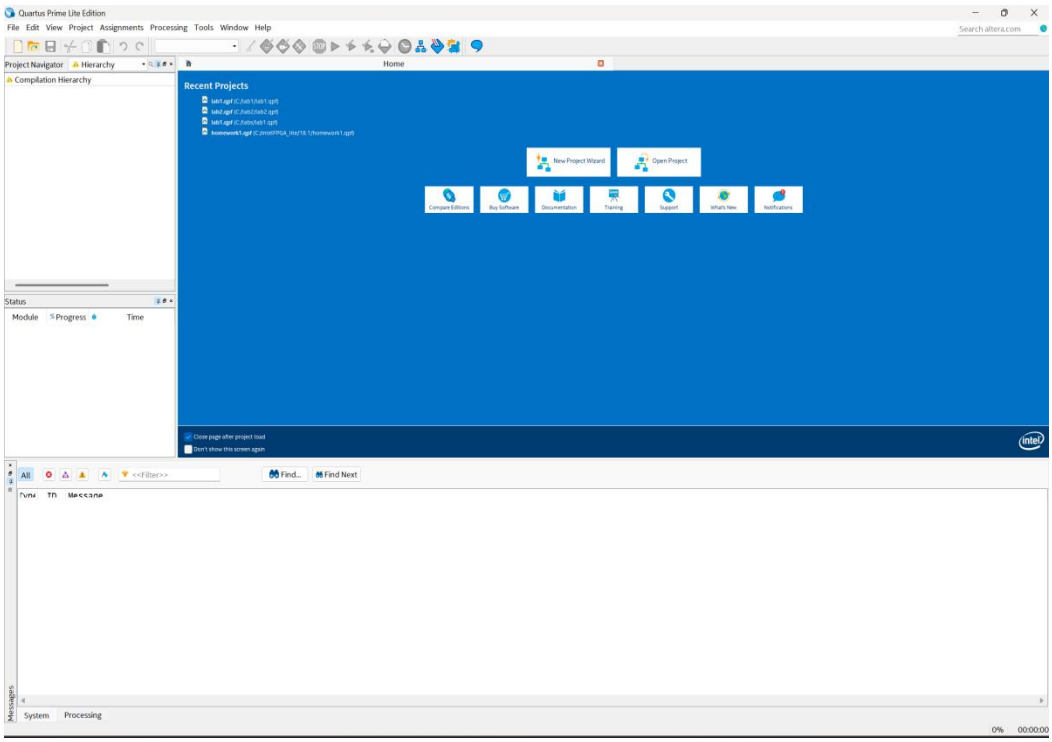


Figure 2: Accessing the New Project Wizard from the Quartus File menu.

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

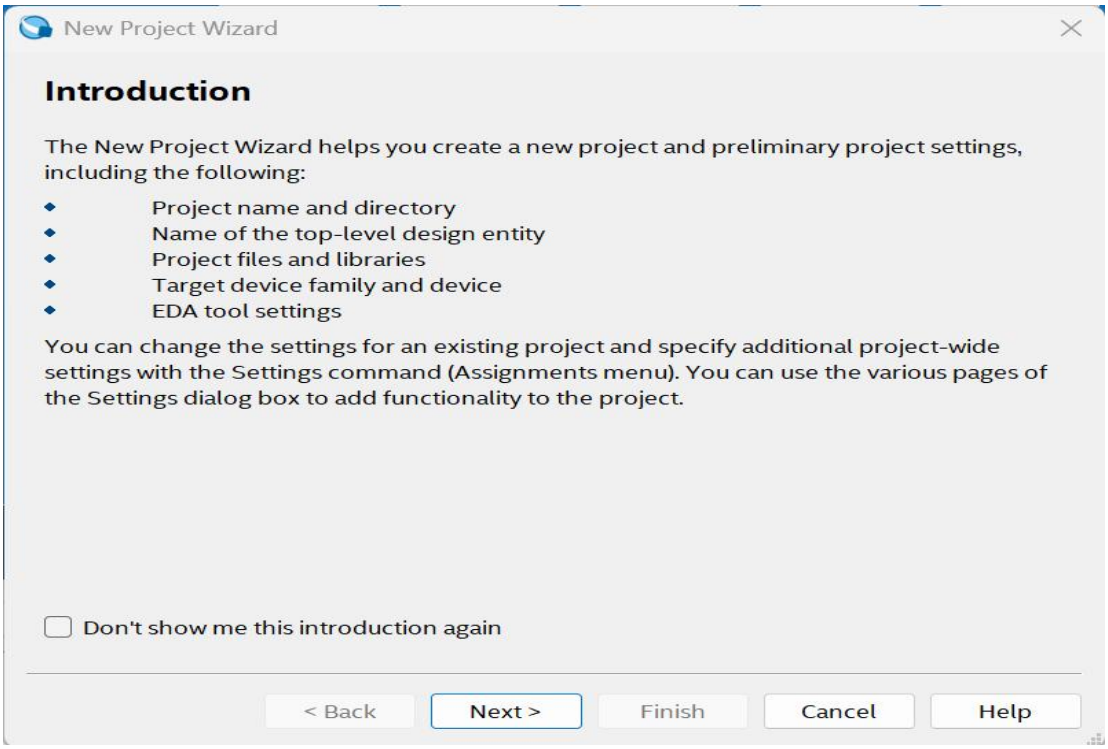


Figure 3: Introduction screen of the New Project Wizard, outlining key setup parameters.

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.

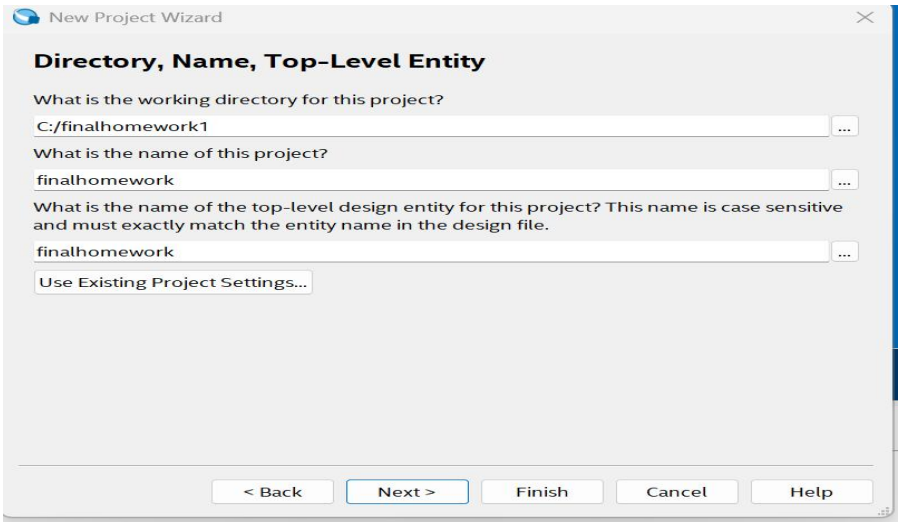


Figure 4: Defining the project directory, name and top-level design entity.

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

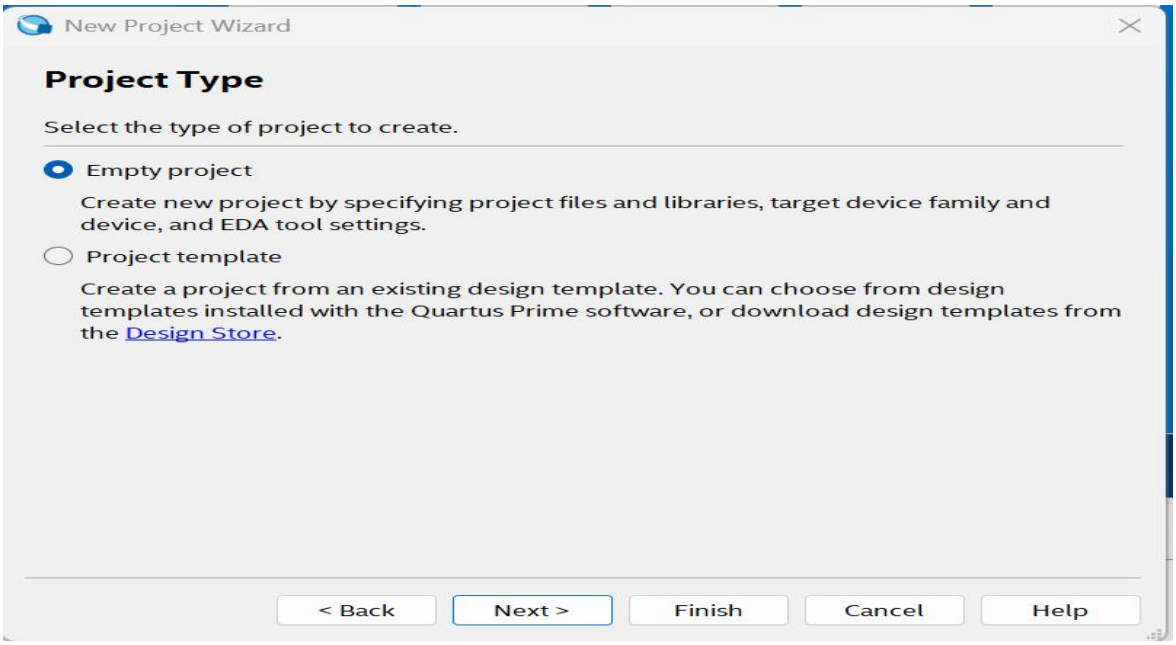


Figure 5: Choosing the "Empty Project" option for a custom setup.

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.

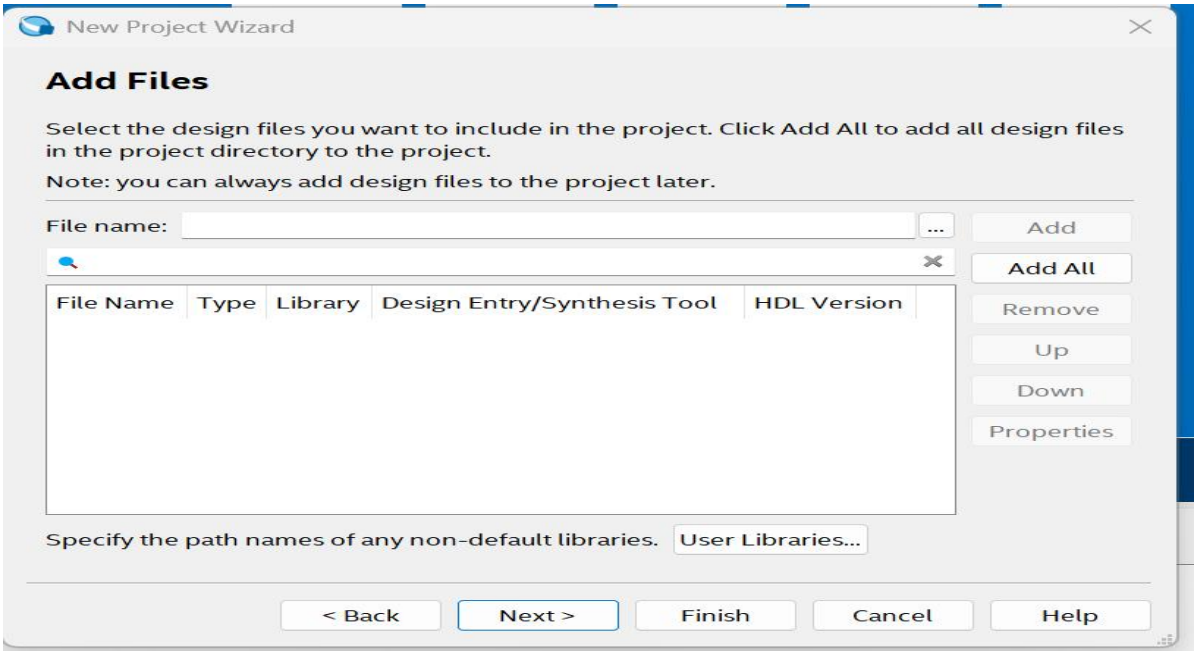


Figure 6: Skipping the file addition step for now, to be handled manually after Qsys generation.

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.

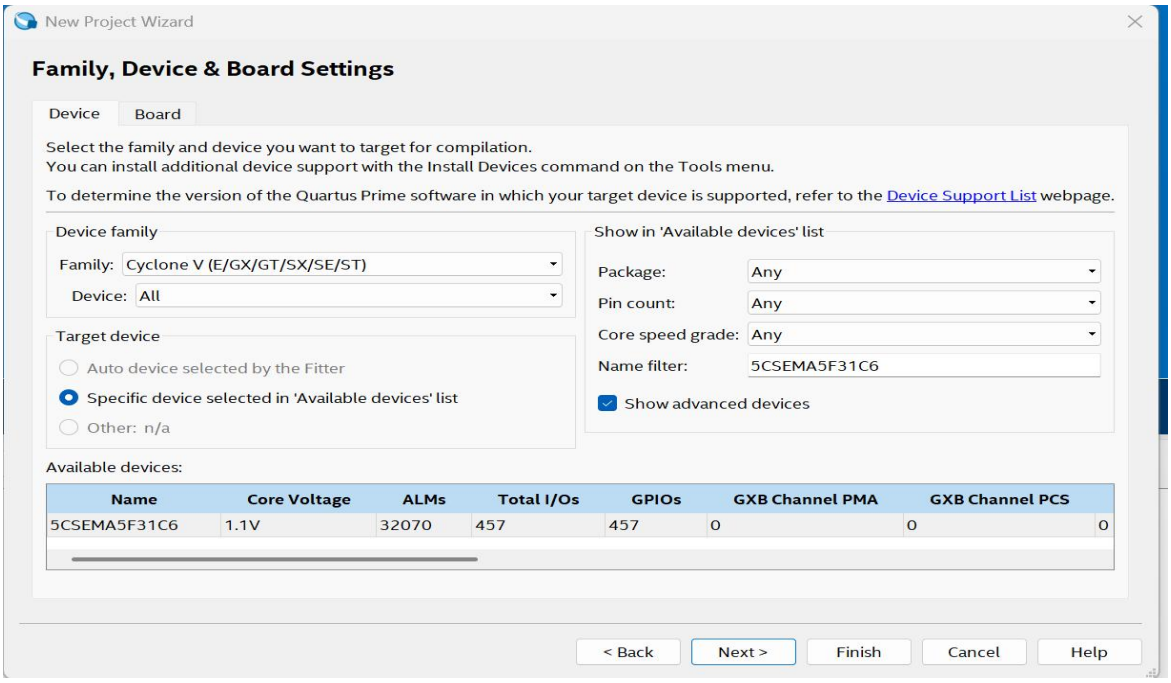


Figure 7: Selecting the Cyclone V device used on the DE1-SoC FPGA board.

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

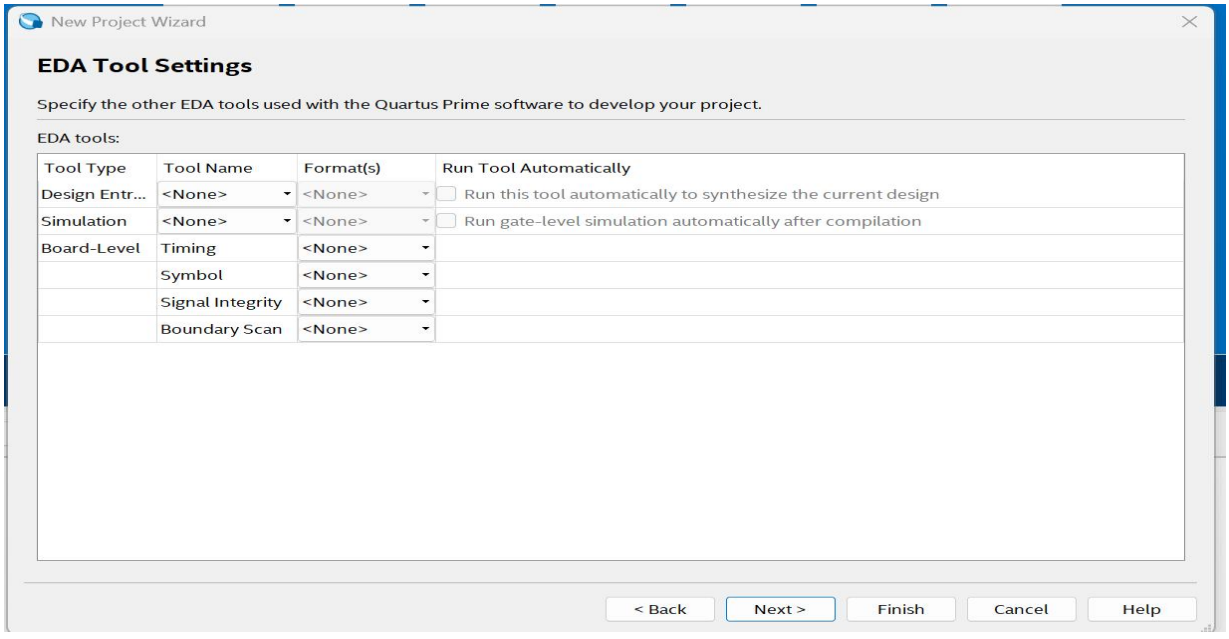


Figure 8: Keeping default settings for EDA tool configuration.

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.

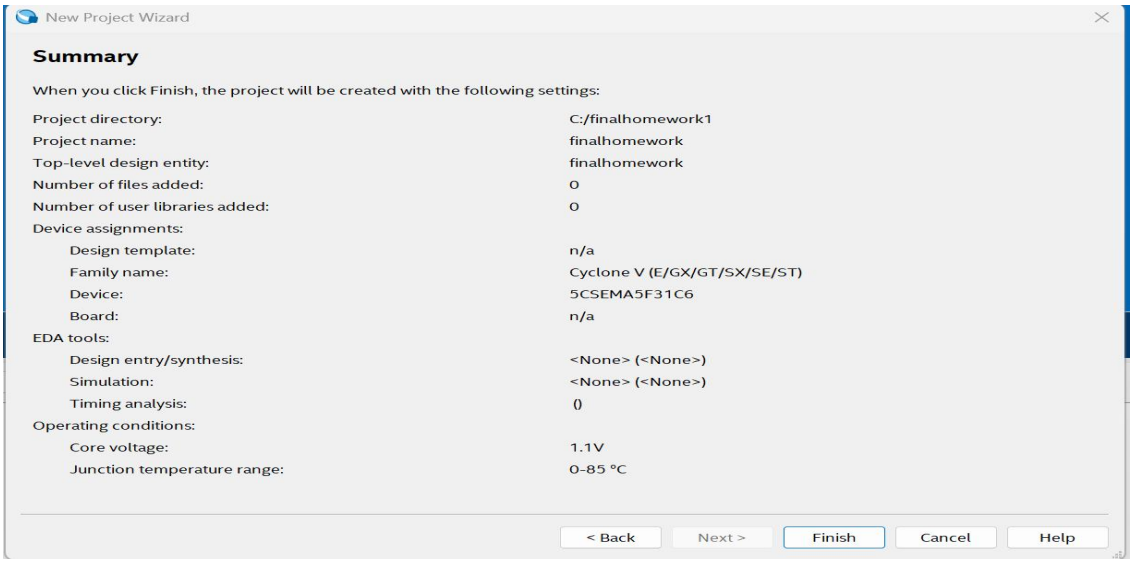


Figure 9: Summary screen confirming all project configuration settings before final creation.

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.

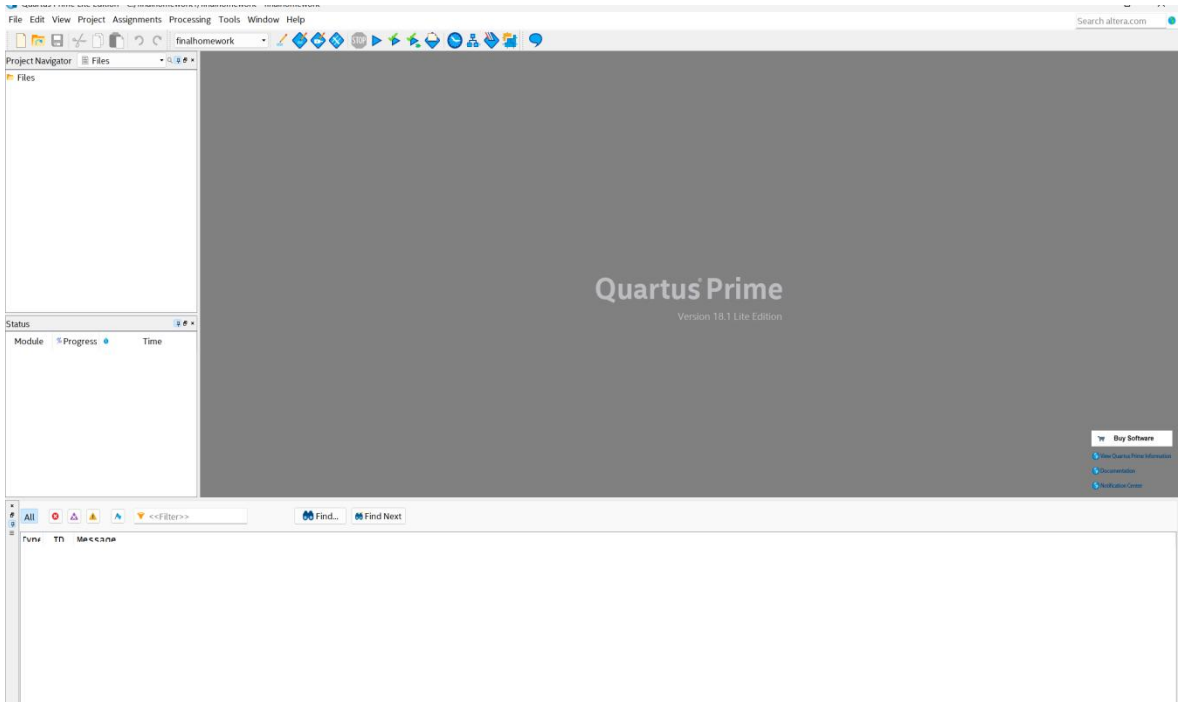


Figure 10: Quartus Prime main interface after project creation, ready for system integration and further development.

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.

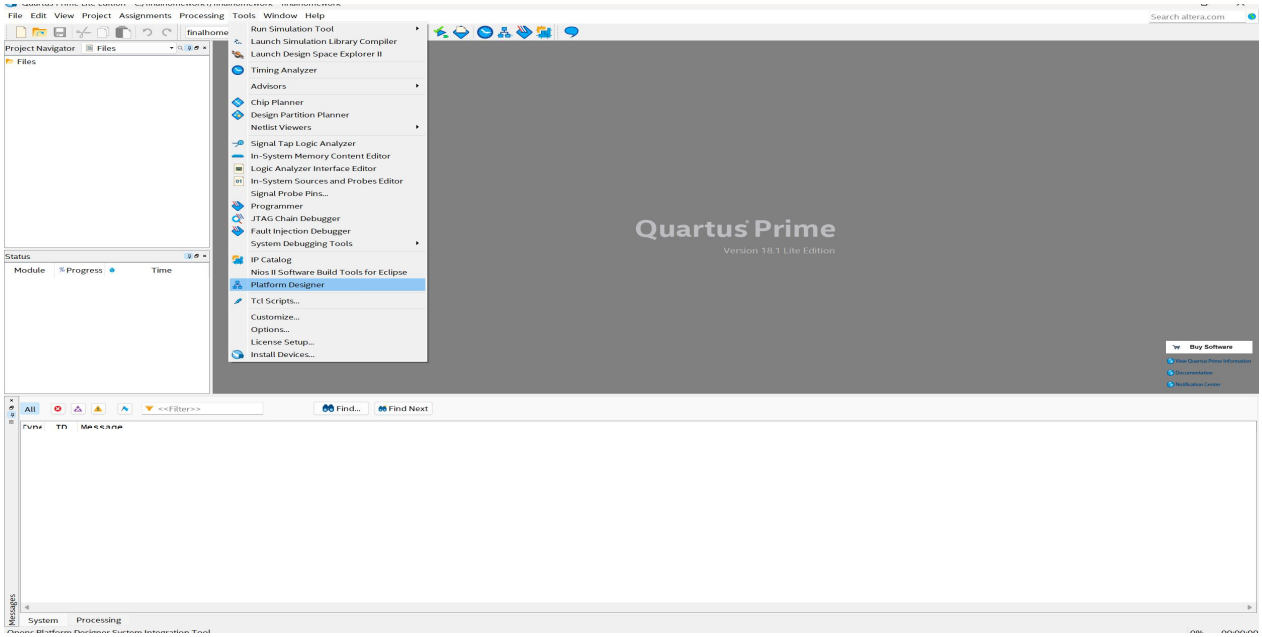


Figure 11: Opening Platform Designer from the Quartus Tools menu.

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.

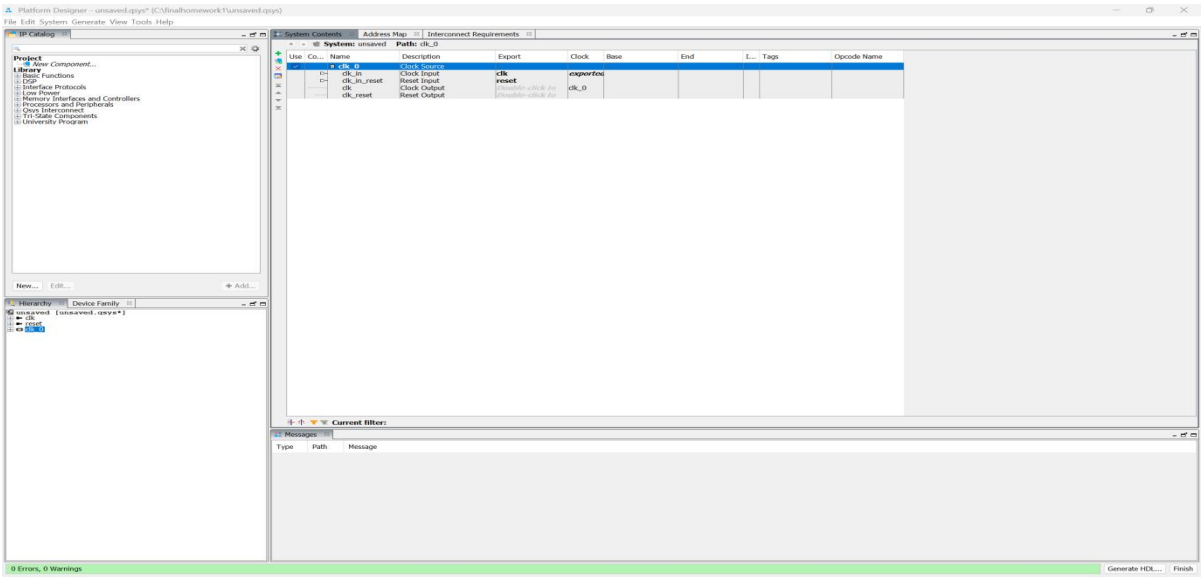


Figure 12: canvas space in qsys, Clock Source component added to the system. Removed this later

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.

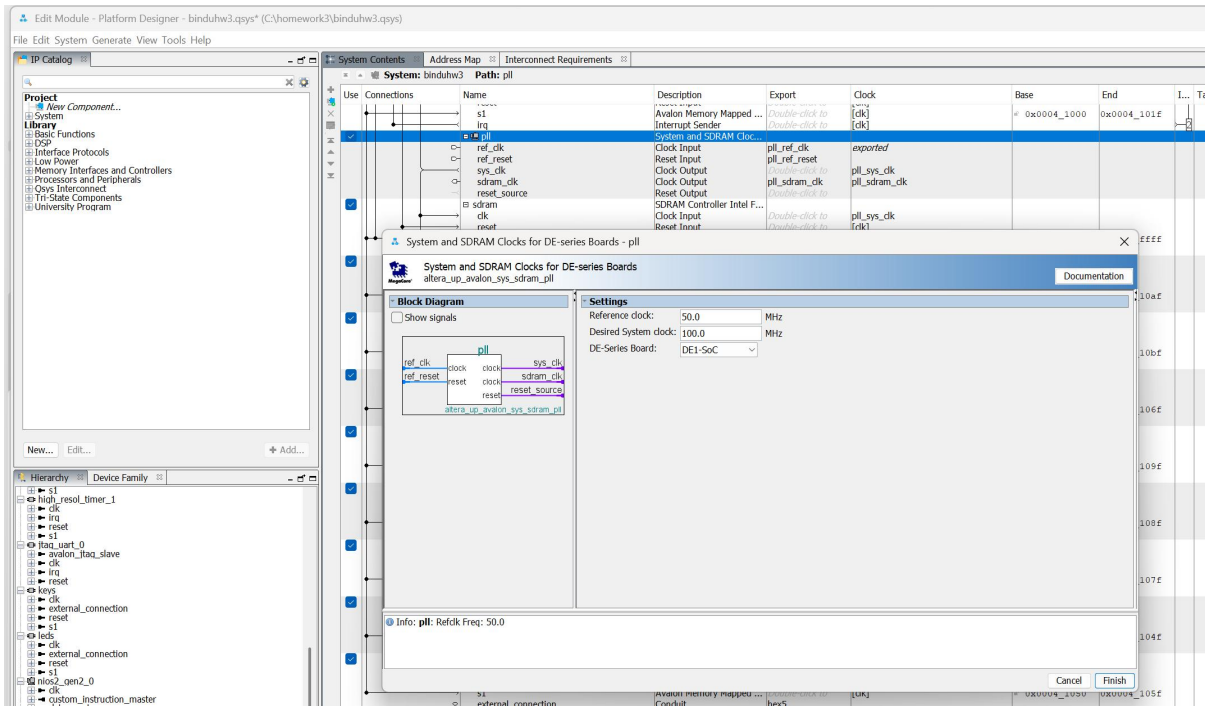


Figure 13: Setting the pll Clock frequency.

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

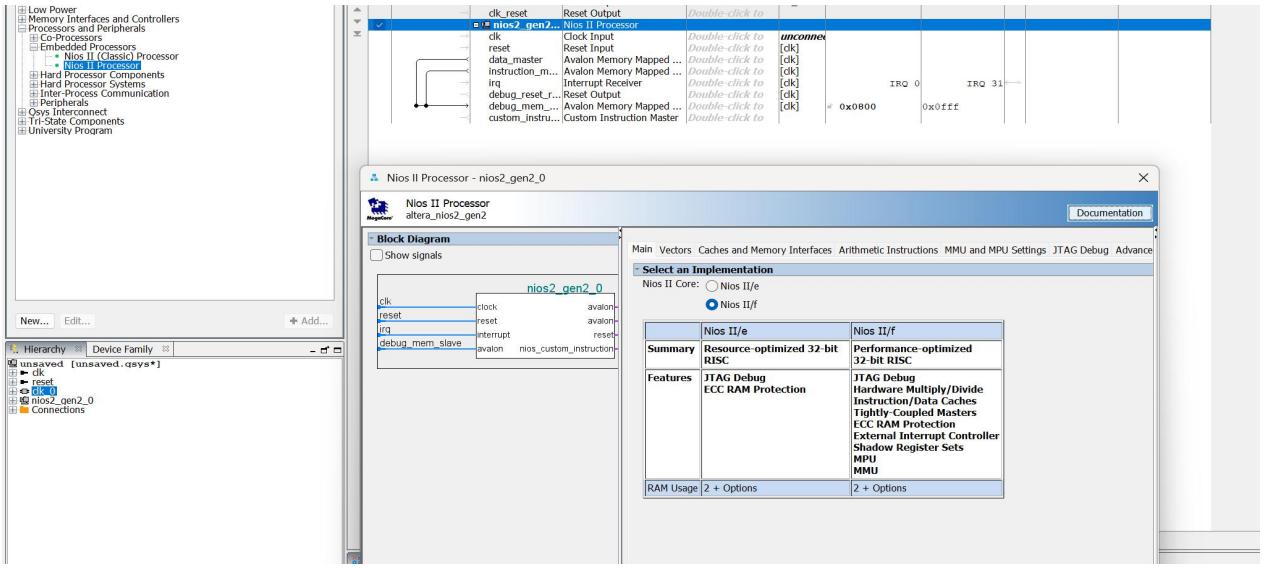


Figure 14: Selecting the Nios II Processor from the IP Catalog and adding into design

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.

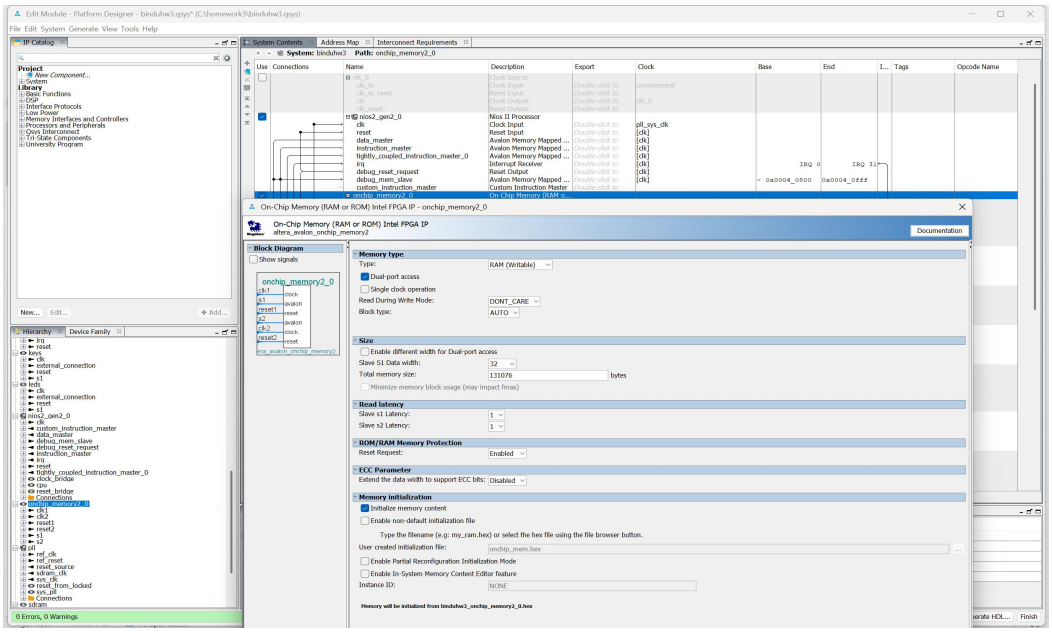


Figure 15: On-Chip Memory Configuration

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.

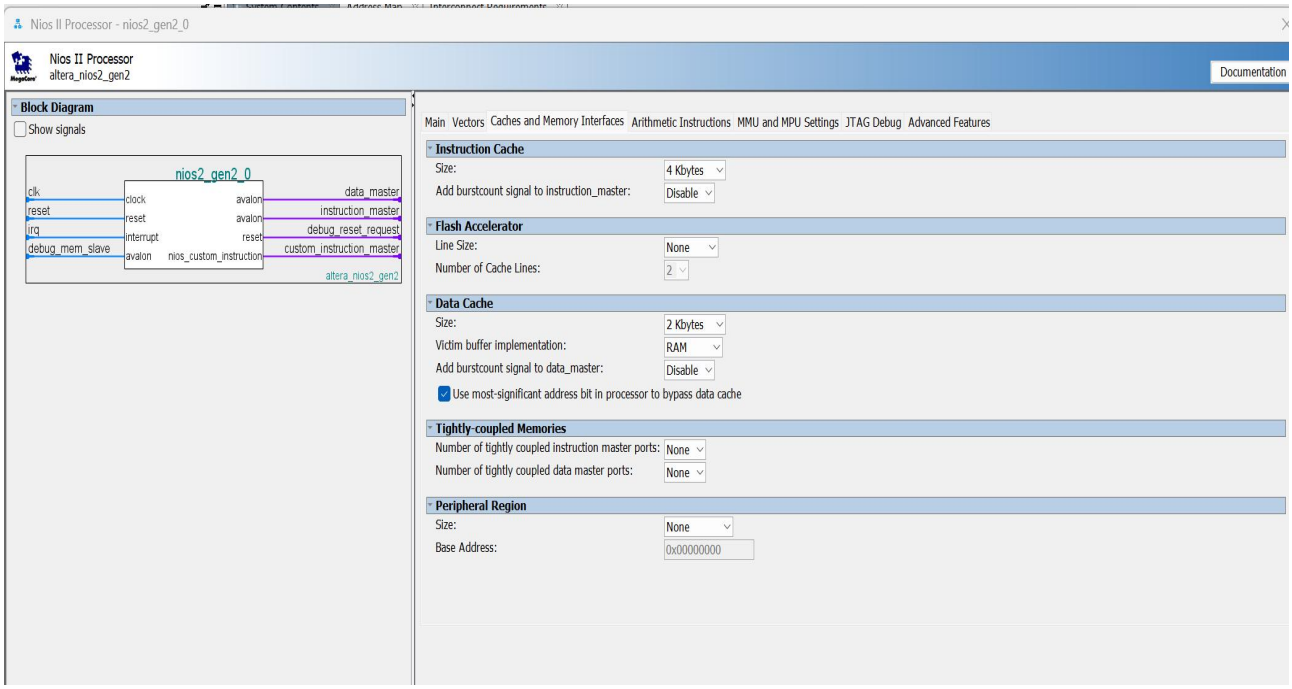


Figure 16: Configuring Cache for Nios II Processor

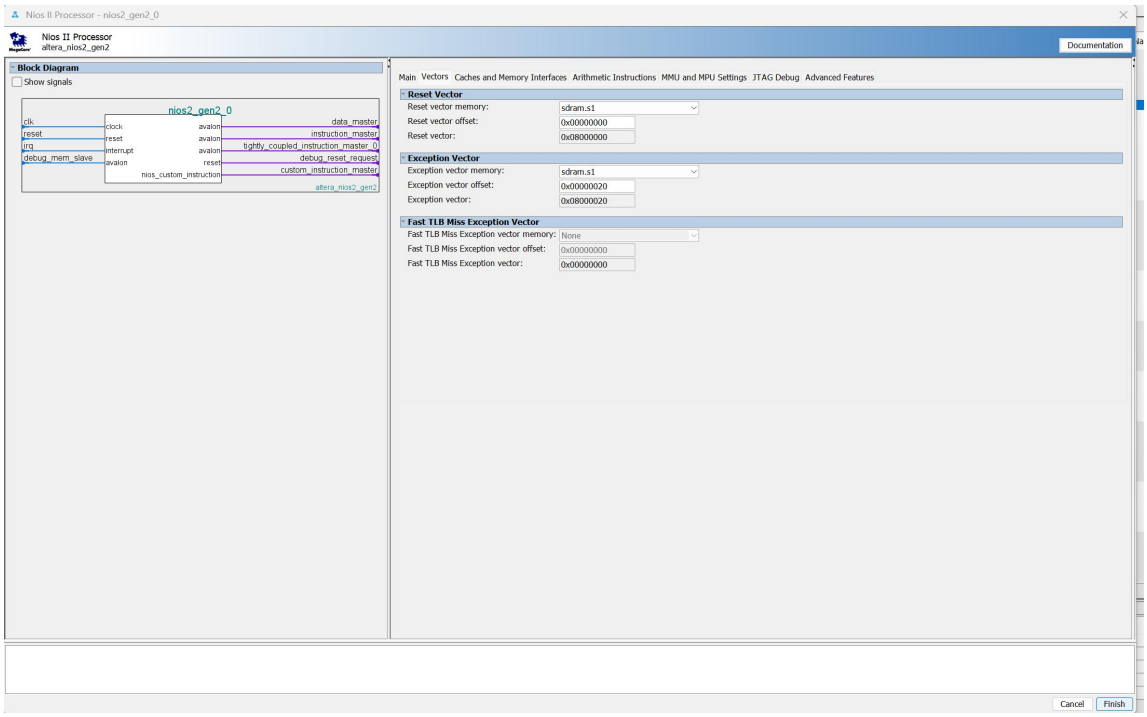


Figure 17: Setting Reset and Exception Vectors

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.

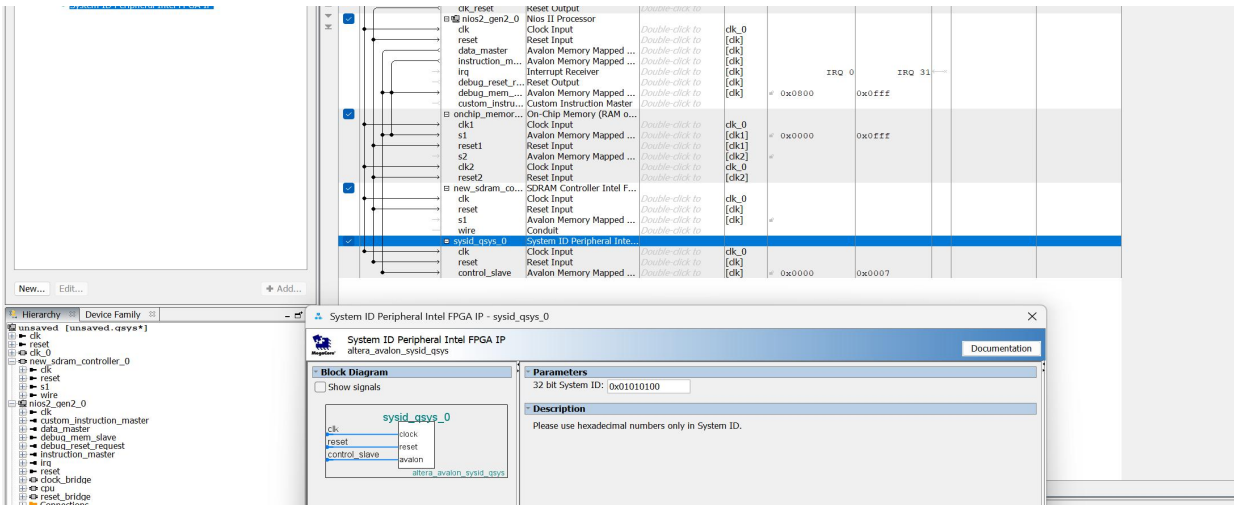


Figure 18: Configuring system ID peripheral

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.

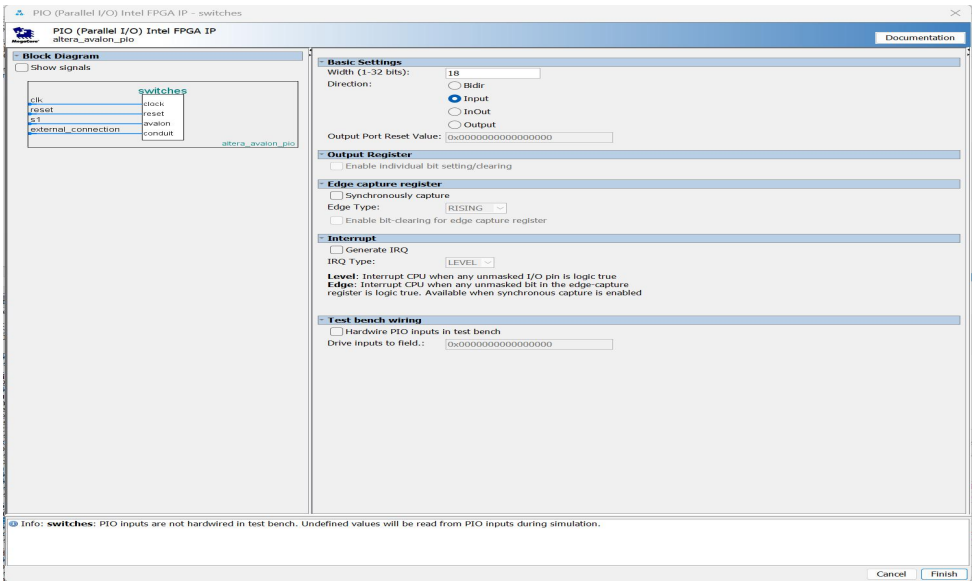


Figure 19: Configuring switches

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.

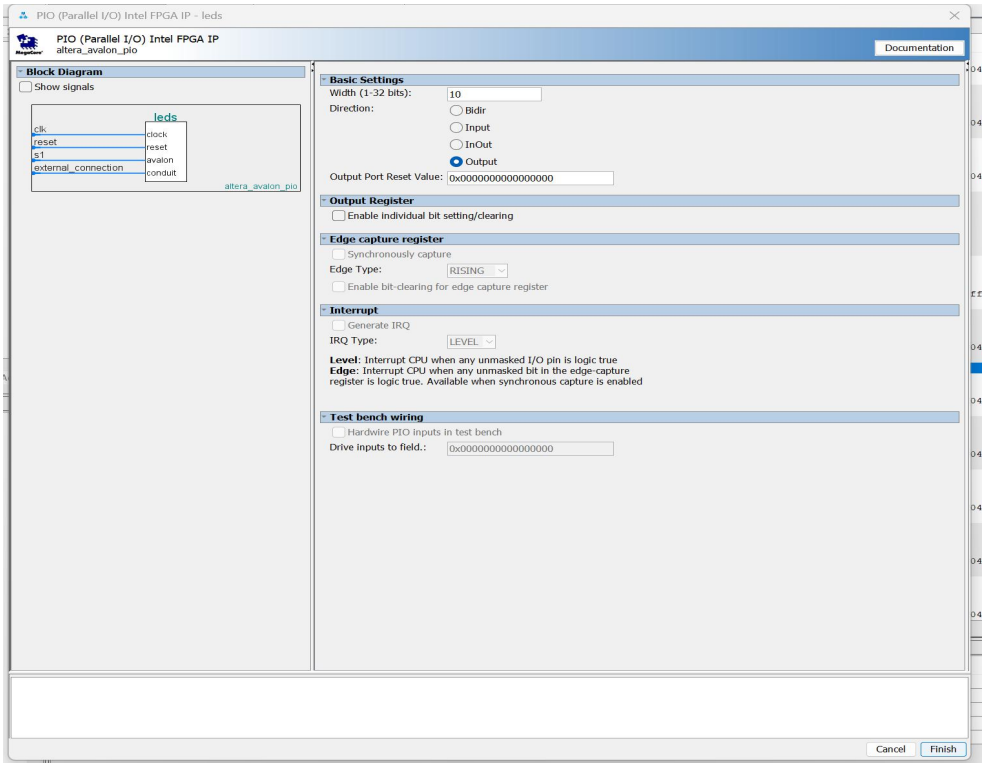


Figure 20: Configuring LEDR PIO

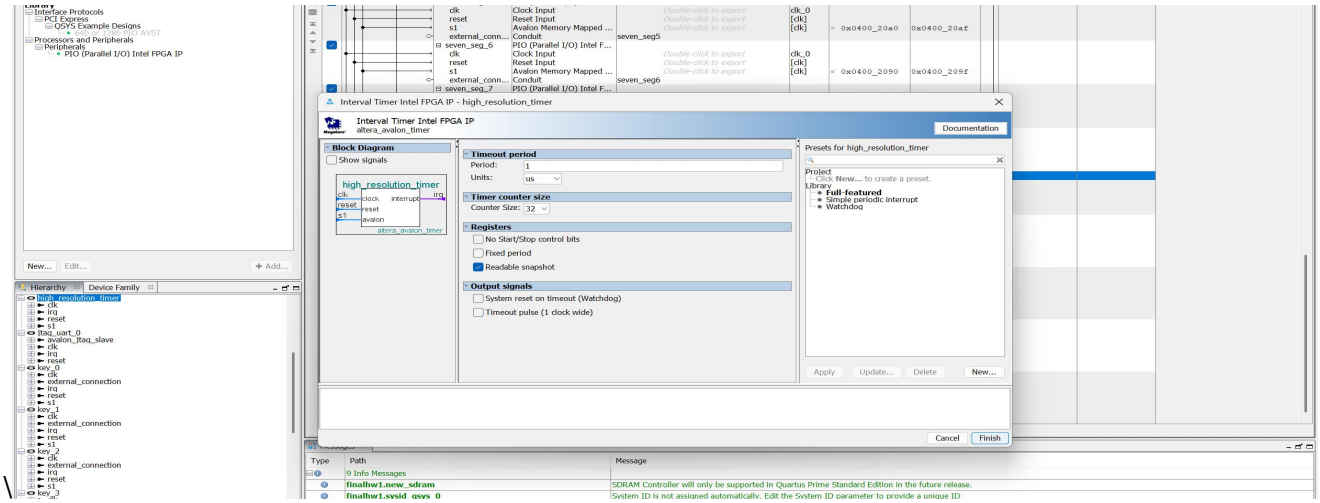


Figure 21: Configuring High Resolution Timer in Platform Designer

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.

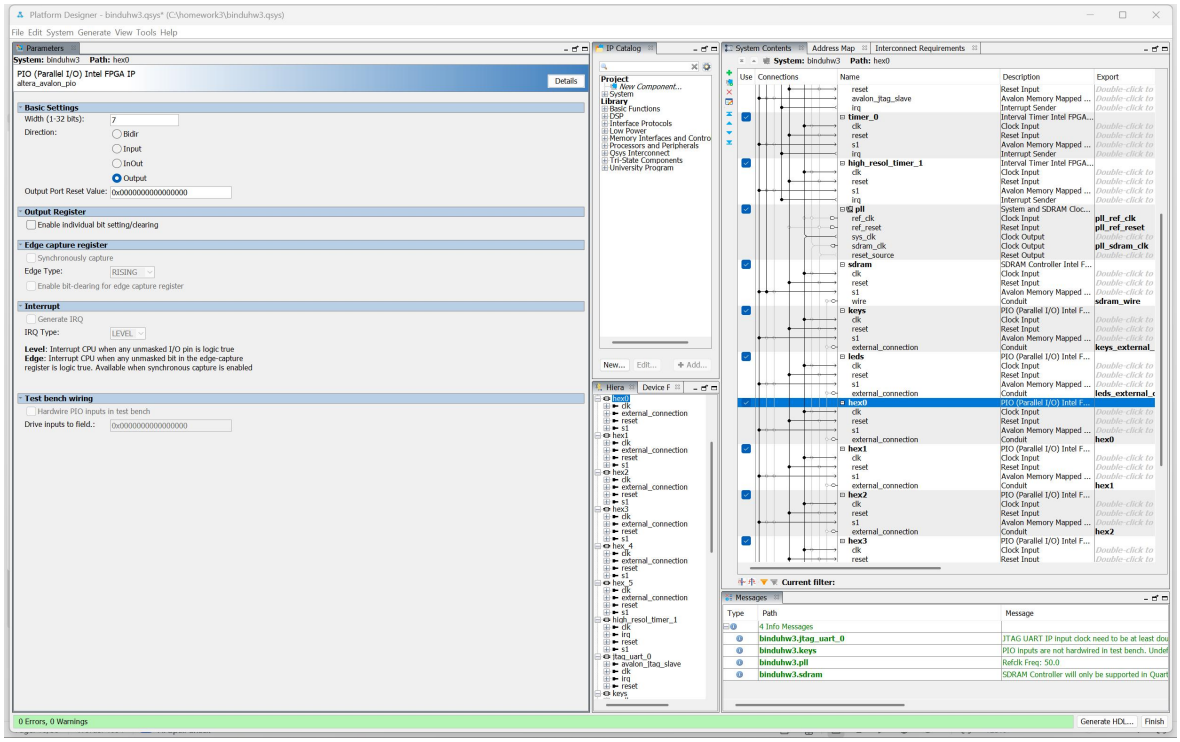


Figure 22: Configuration of Seven Segment Display

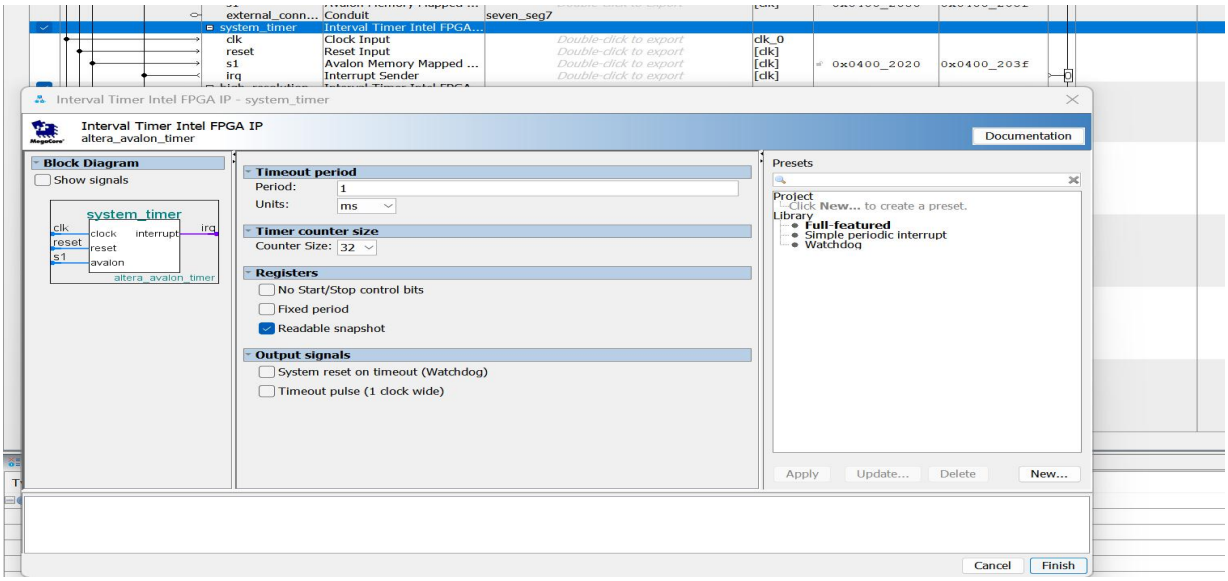


Figure 23: Adding and Configuring System Timer

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.

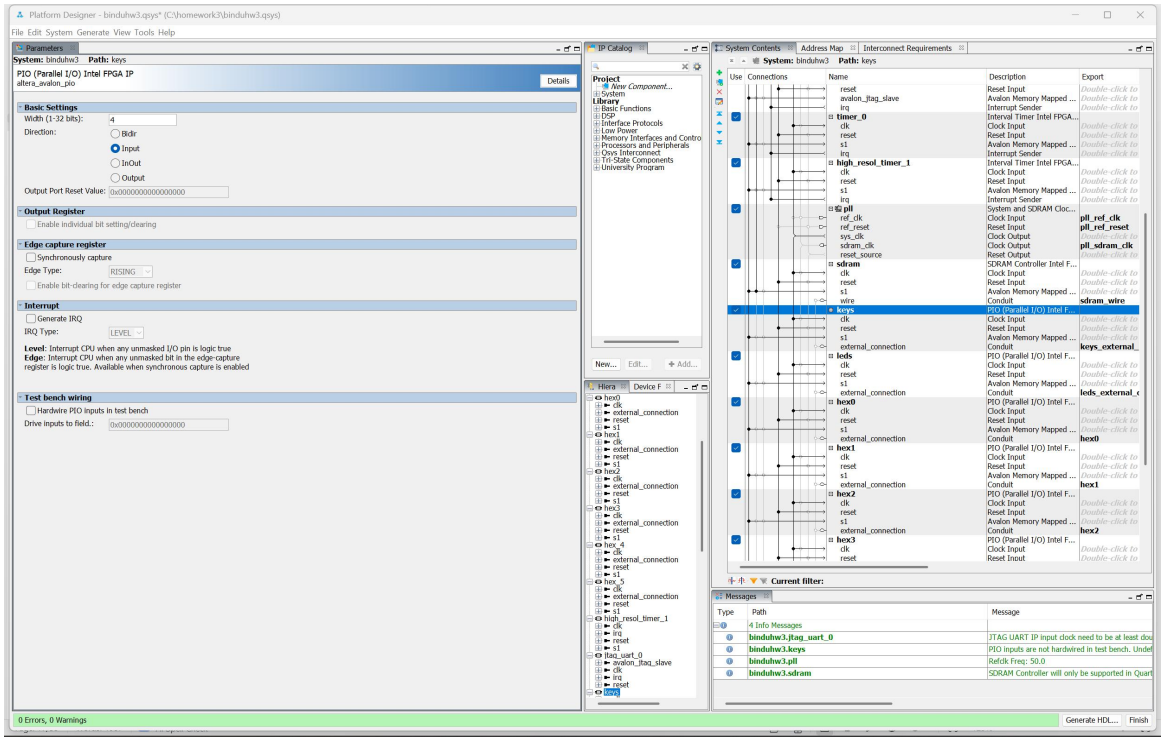


Figure 24: Configuring keys as input (PIO)

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.

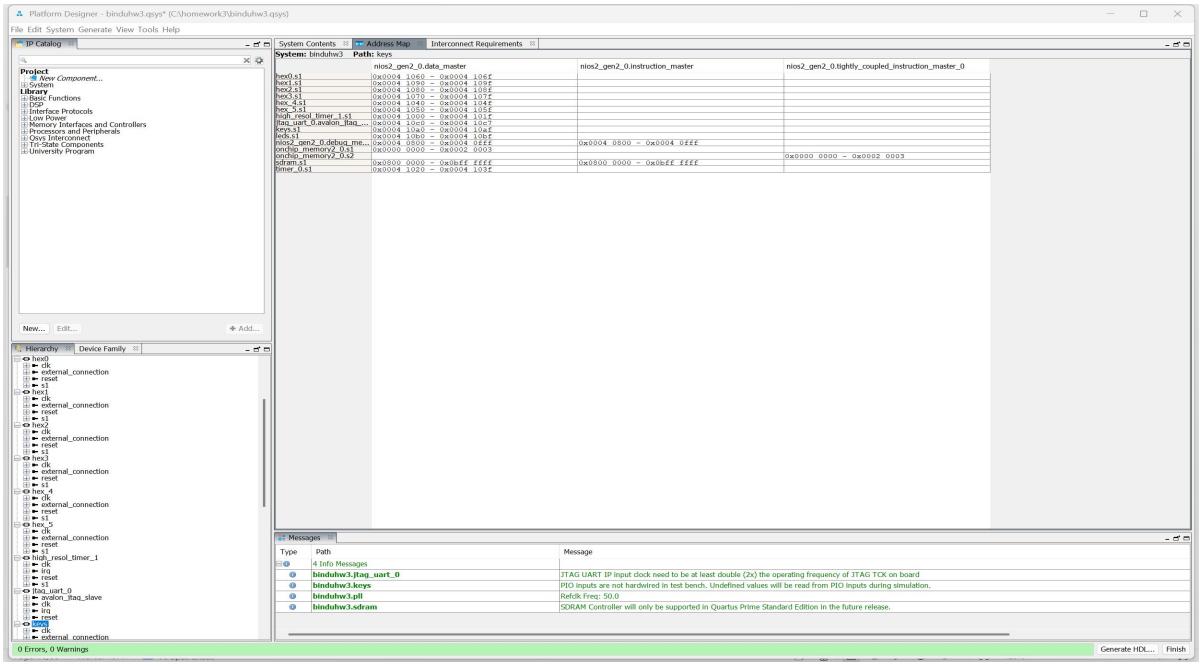
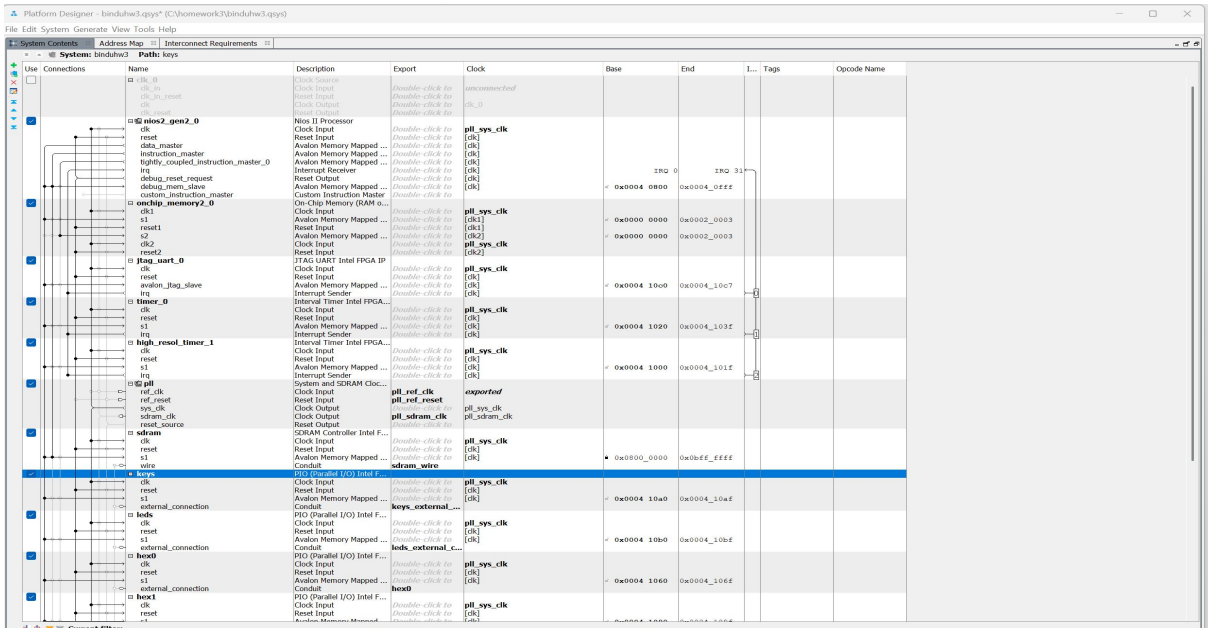


Figure 25: Memory allocation doesn't overlap. Auto assign base address

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



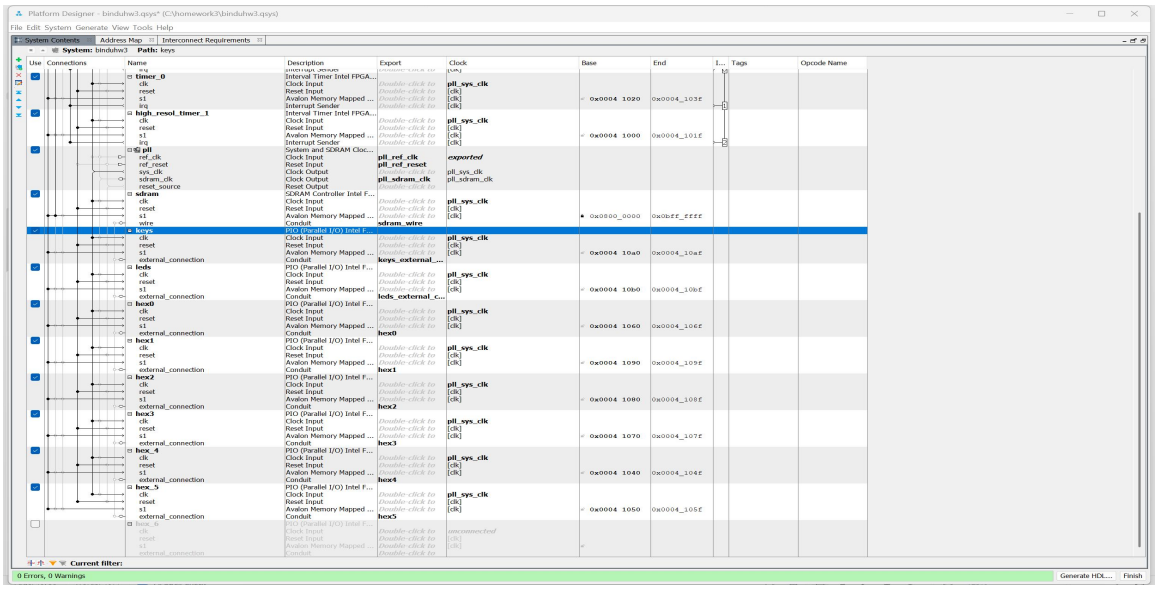


Figure 26: qsys design layout

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.

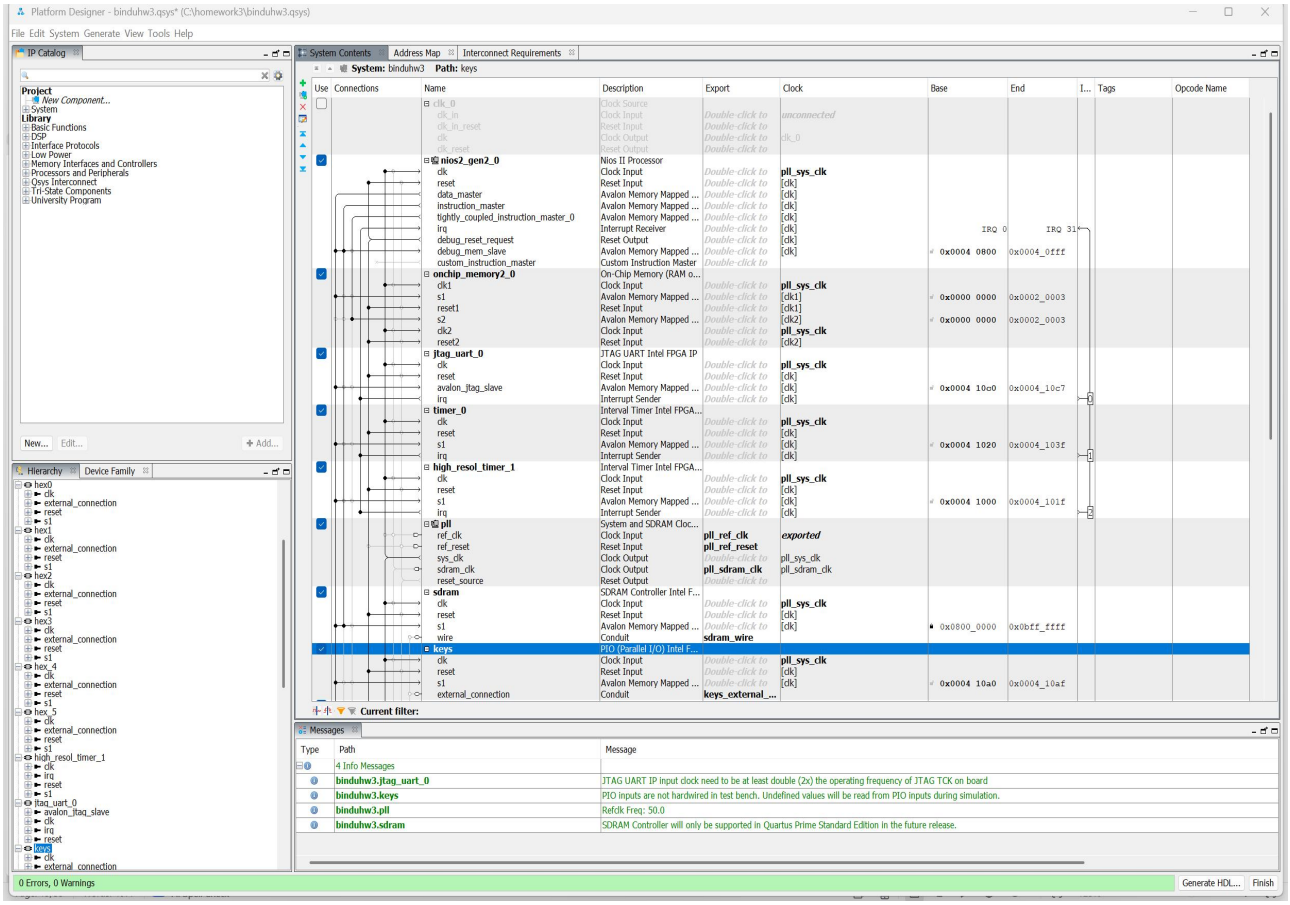


Figure 27: Final Connections and Summary in Platform Designer

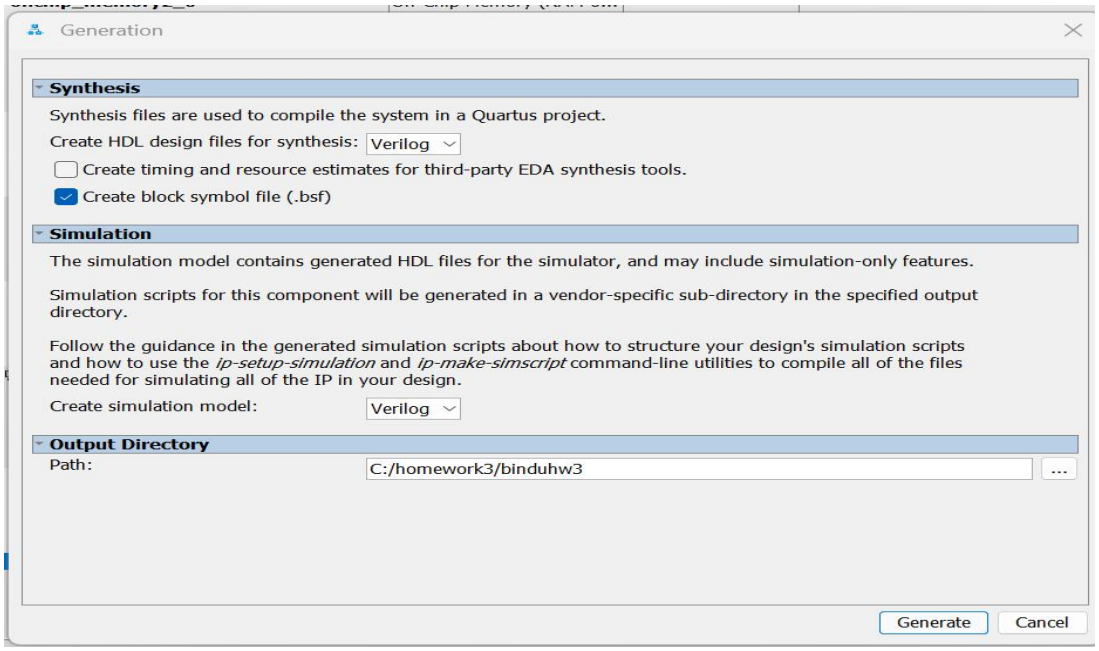


Figure 28: Generate HDL

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.

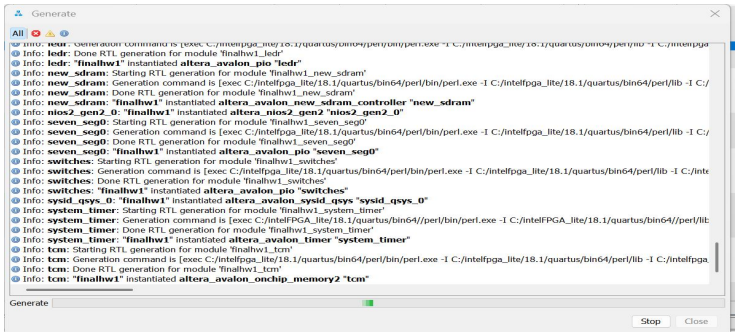


Figure 29: Generation process in progress

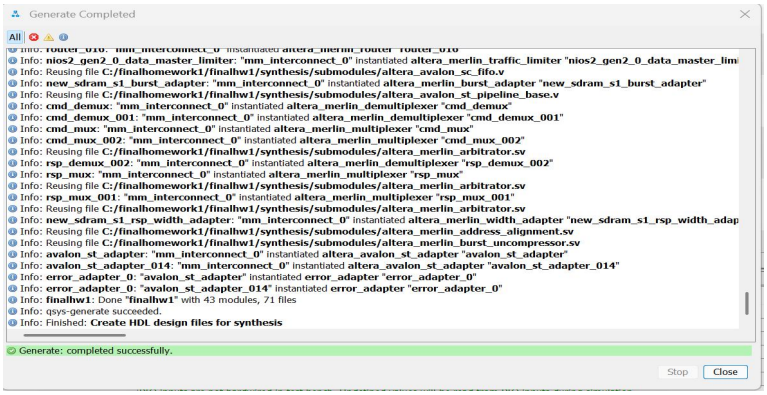


Figure 30: Generation completed successfully

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.

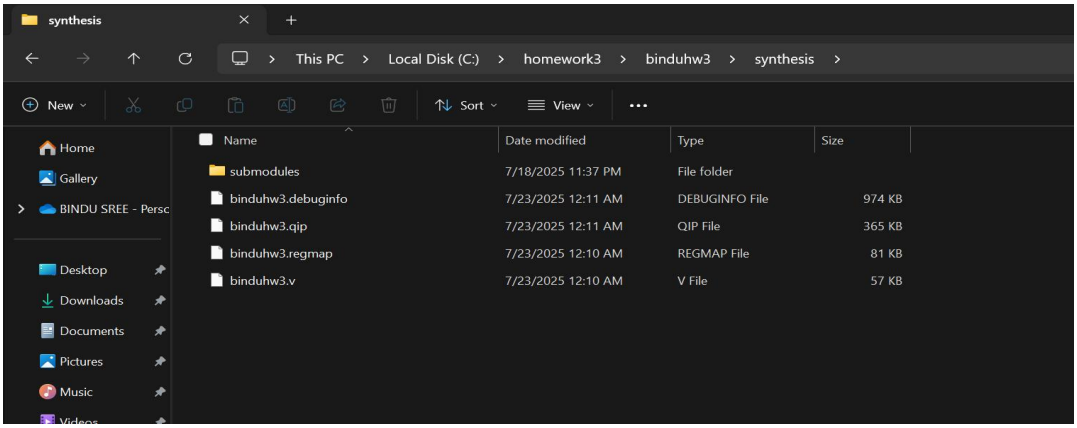


Figure 31: Files created in project file directory

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.

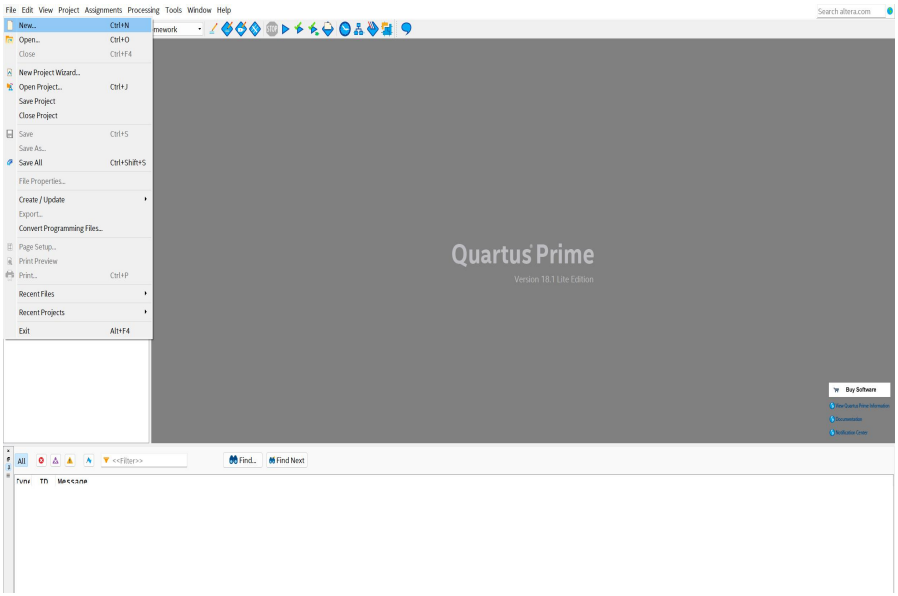


Figure 32: Creating new file from Quartus File Menu

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.

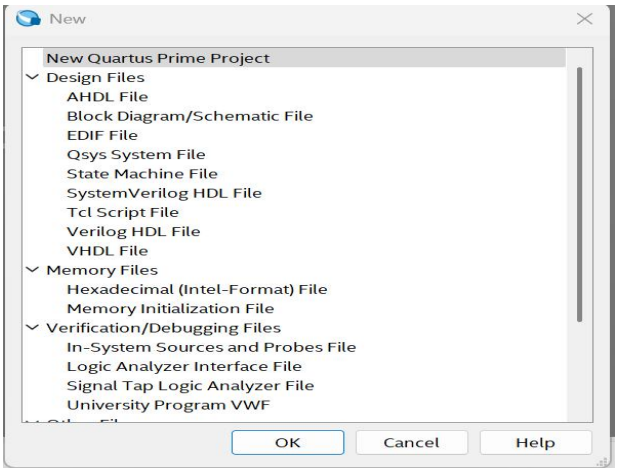


Figure 33: New File Types Selection

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.

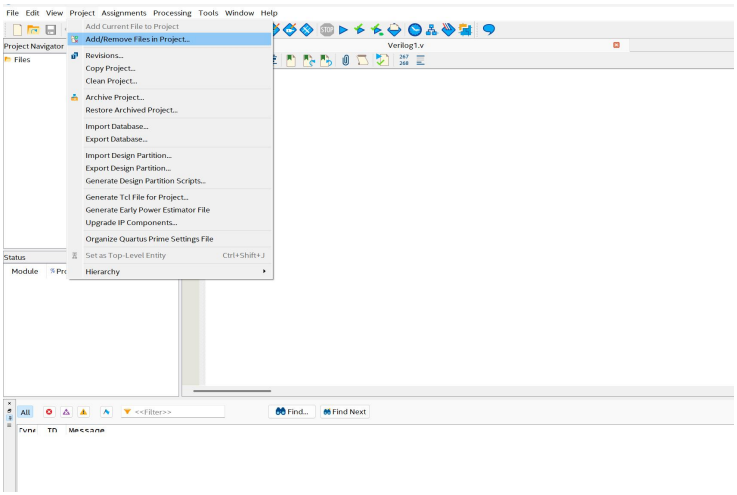


Figure 34: Adding Files to the Quartus Project

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

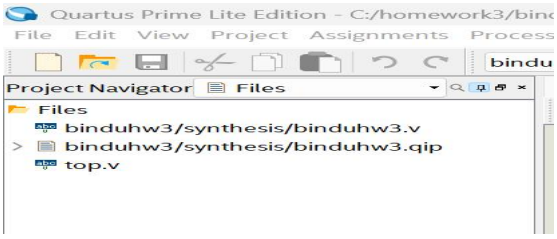


Figure 35: In project navigator, these loaded files can be seen

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

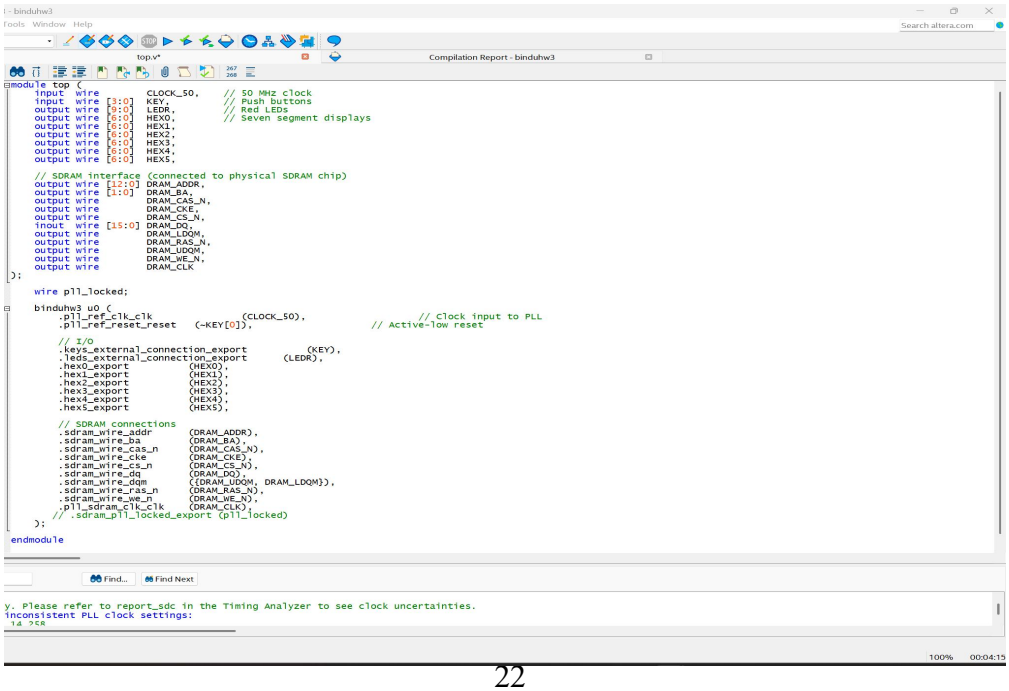


Figure 36: Top-Level Module Declaration in Verilog

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

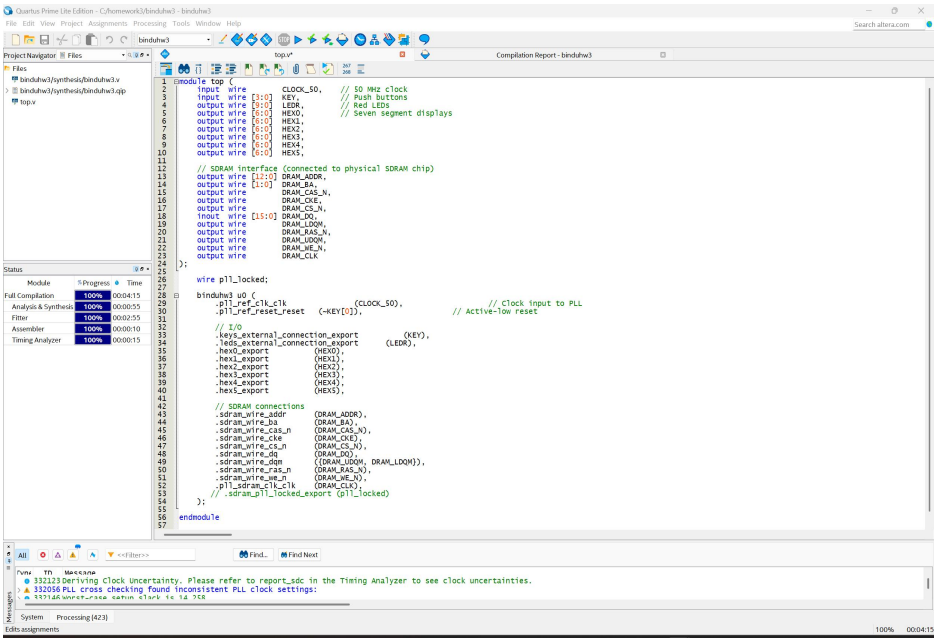


Figure 37: Success compilation for analysis block, no syntax errors

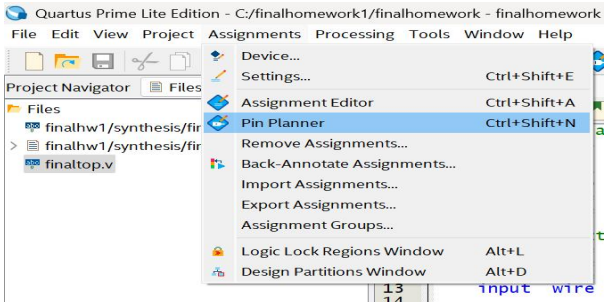
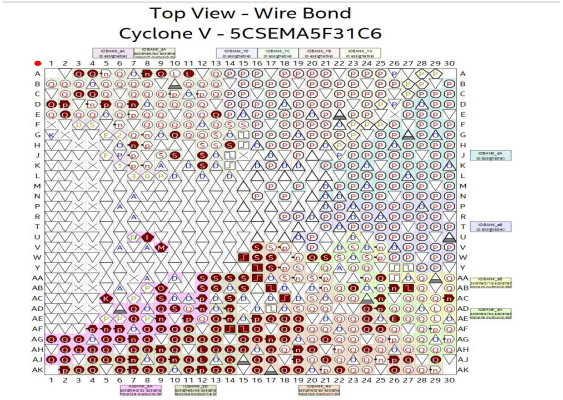


Figure 38: Accessing Pin Planner via Assignments Menu

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.



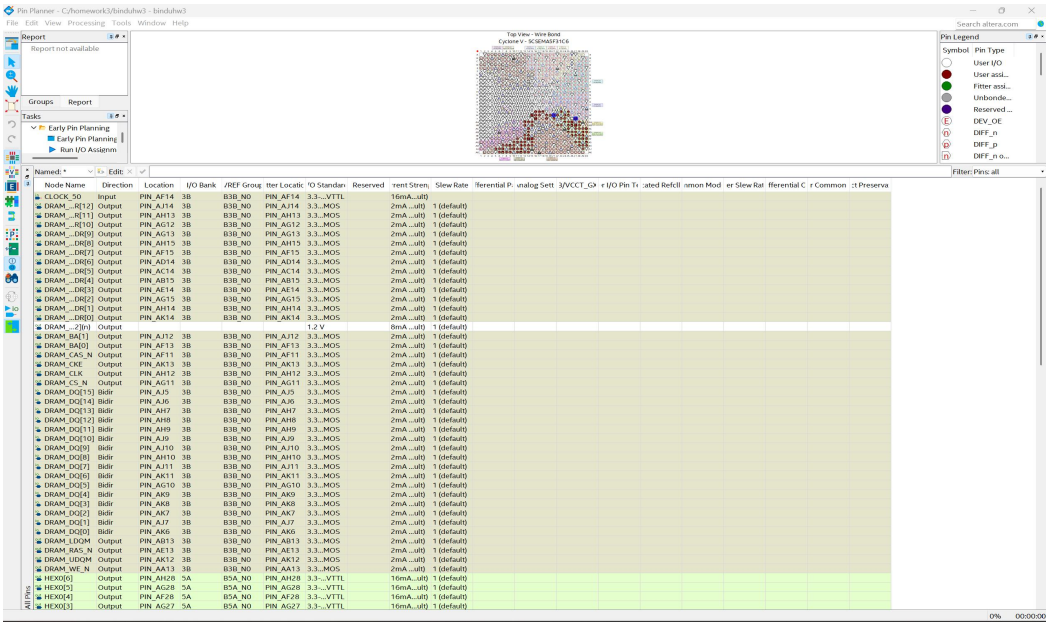


Figure 39: Pin Planner with Assigned I/O Pins

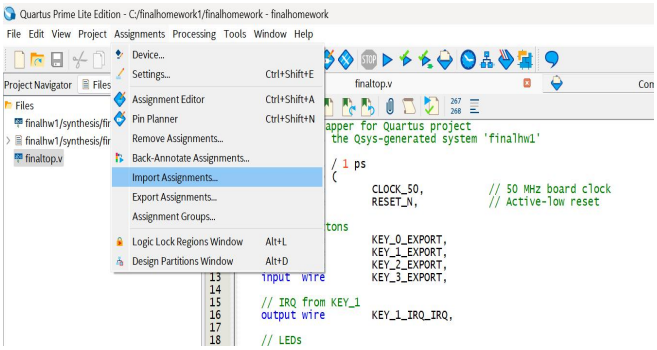


Figure 40: Import Assignments for pins

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

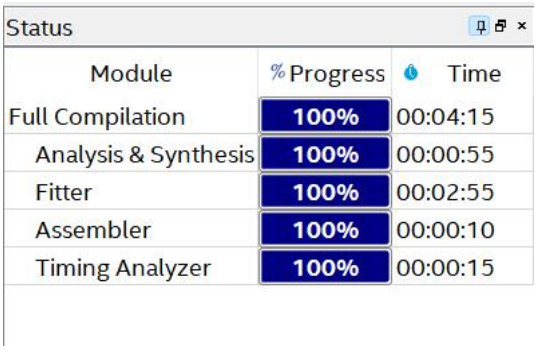


Figure 41: Status panel shows 100%

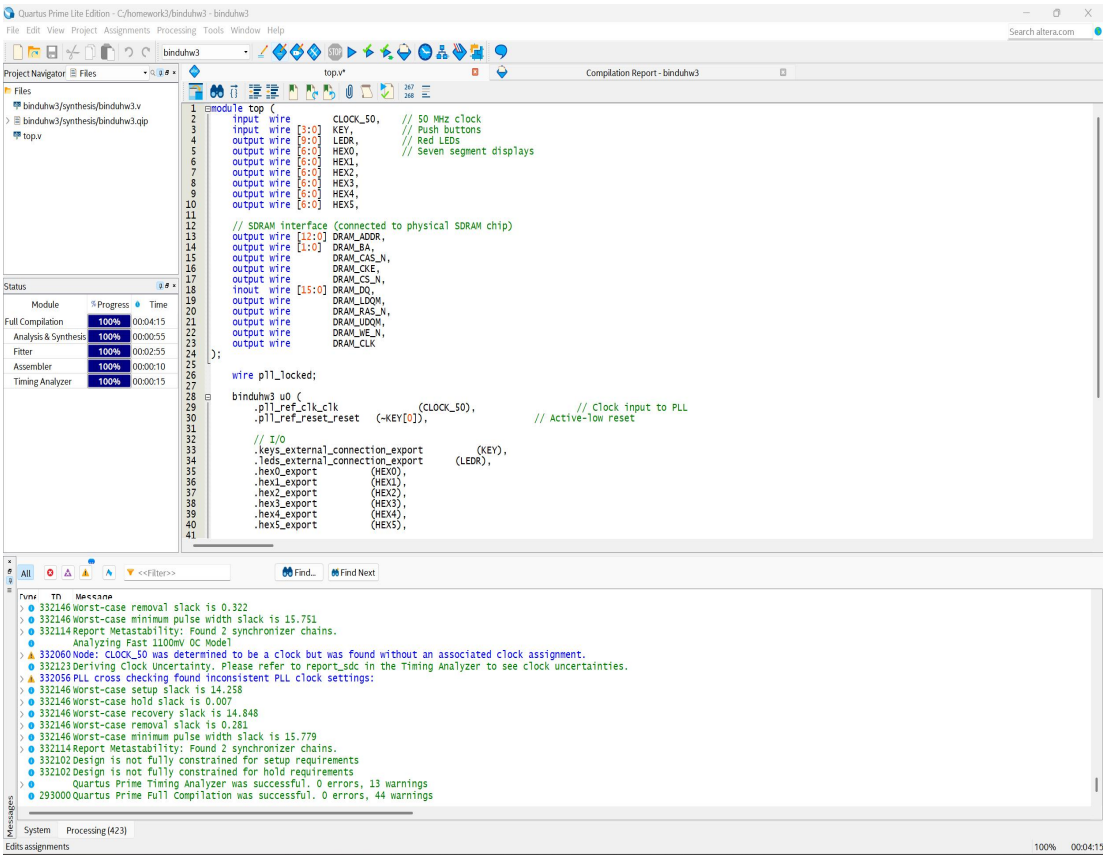


Figure 42: Successful Compilation and Analysis in Quartus Prime

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.

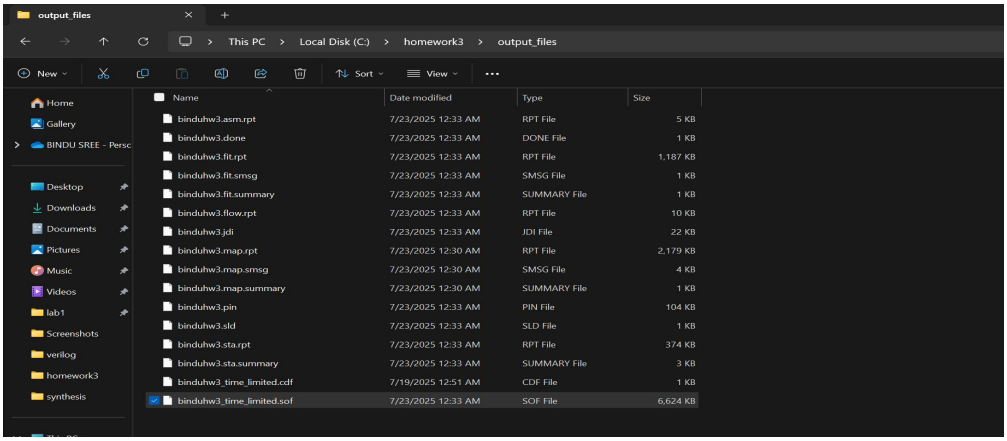


Figure 43: .sof files generated in file manager

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.

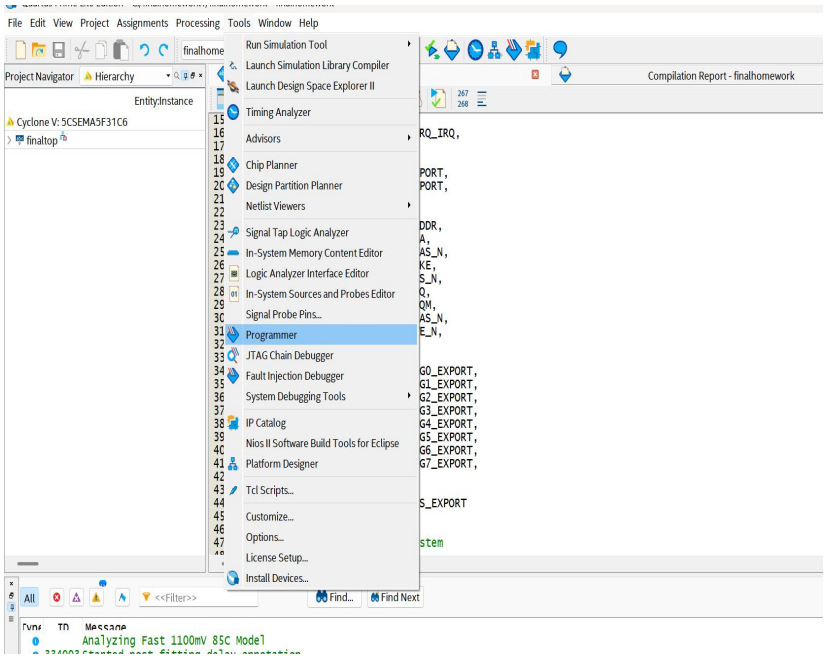


Figure 44: Launching the Programmer Tool in Quartus Prime

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.

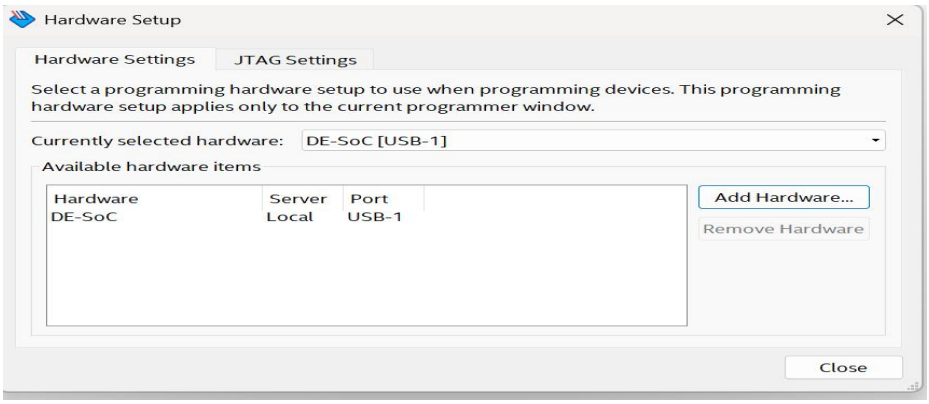


Figure 45: Hardware Setup Selection in Quartus Prime Programmer

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

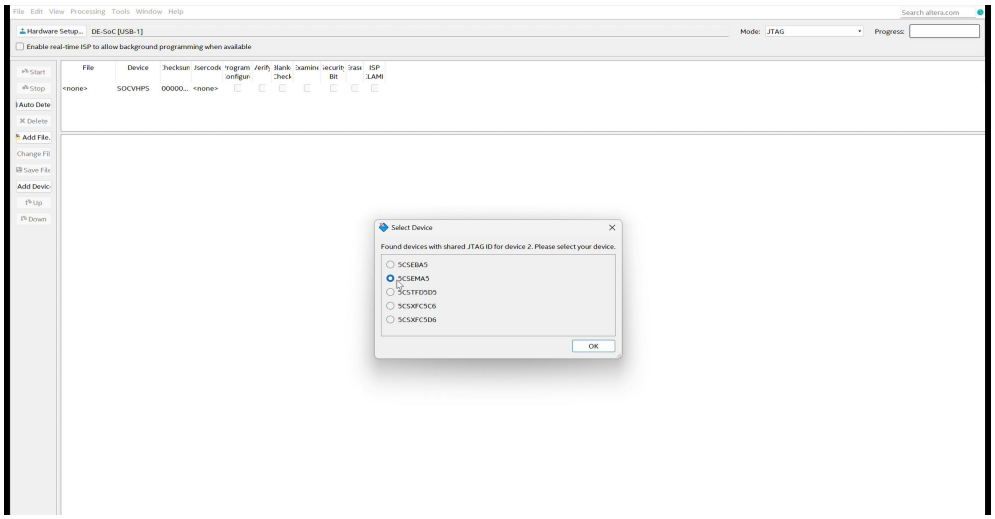


Figure 46: Device selection in Quartus Prime Programmer

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.

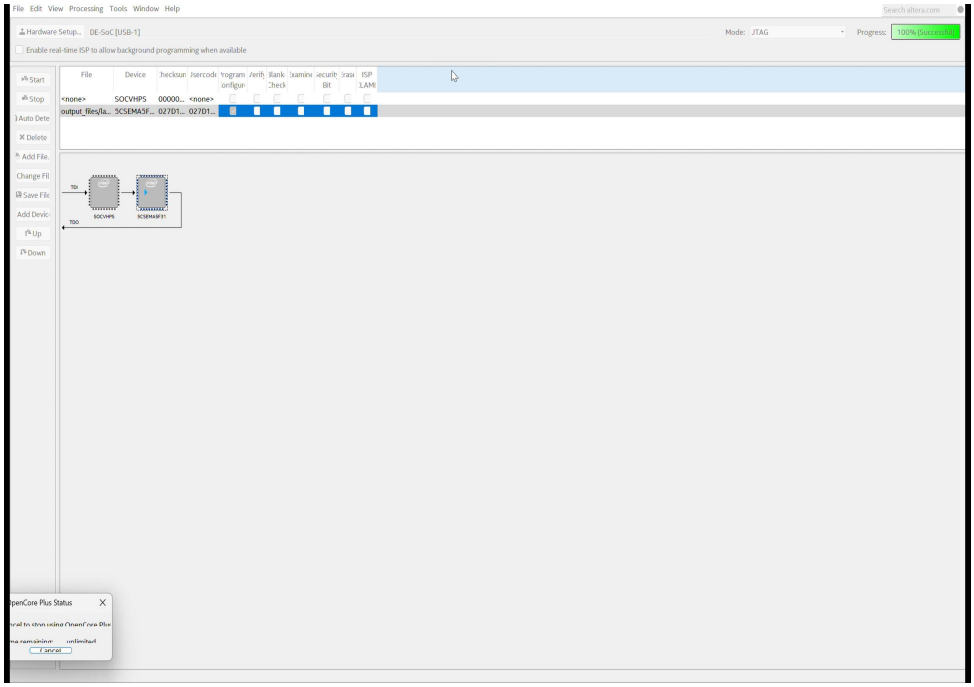


Figure 47: Successful FPGA Programming in Quartus Programmer

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

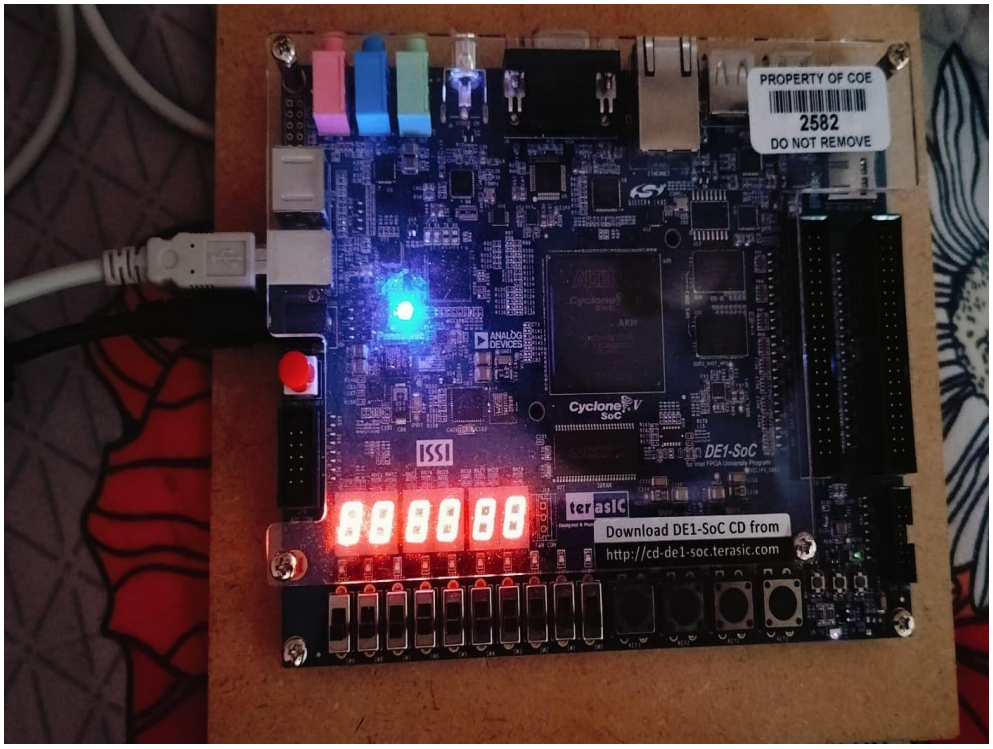


Figure 48: FPGA board after programming .sof file

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.

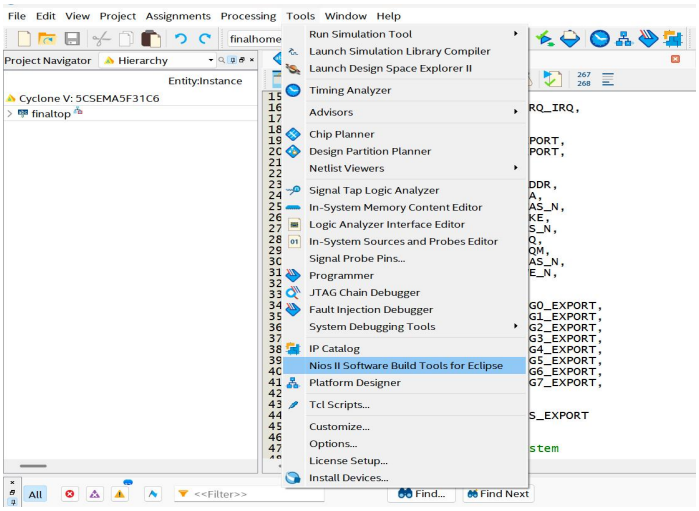


Figure 49: Opening Nios II Software Build Tools for Eclipse

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 3 separate projects for each part. I did this same steps for creating application and bsp projects to all parts.

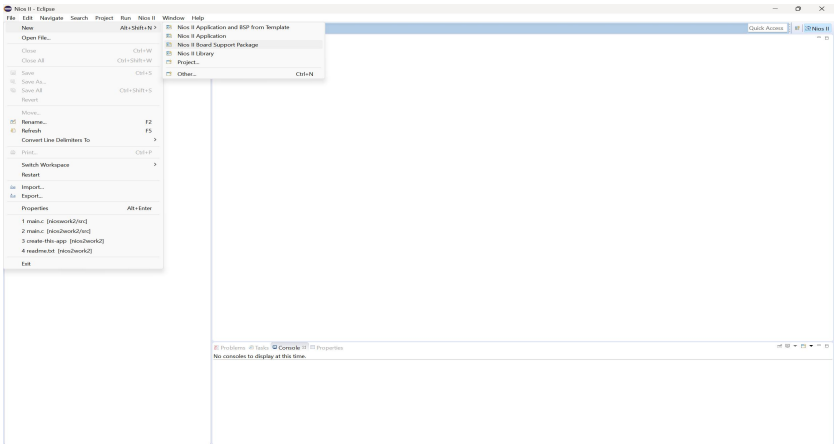


Figure 50: Creating new application project and the BSP

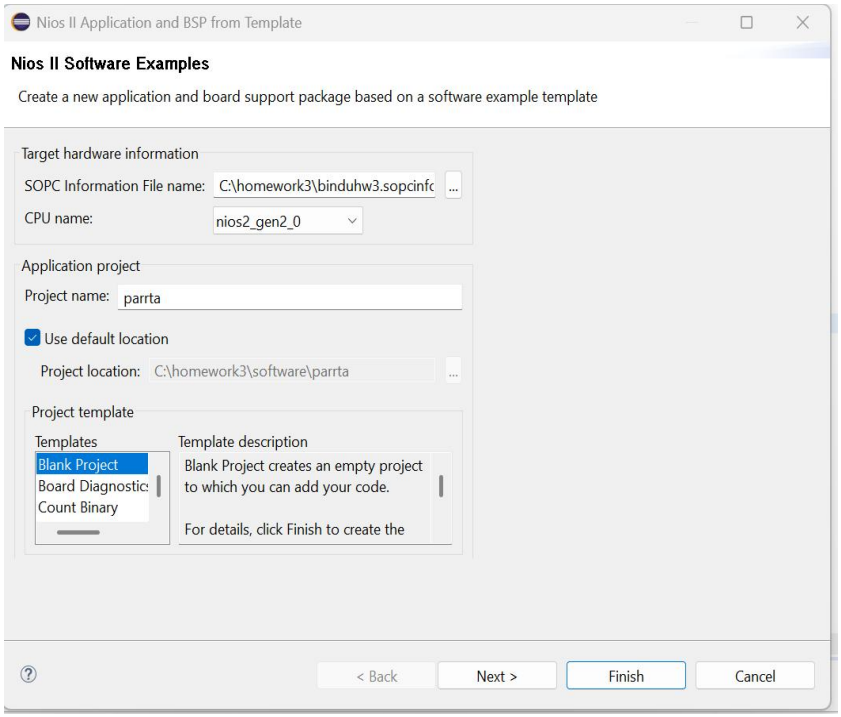


Figure 51: In the Nios II Application and BSP from Template window

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.

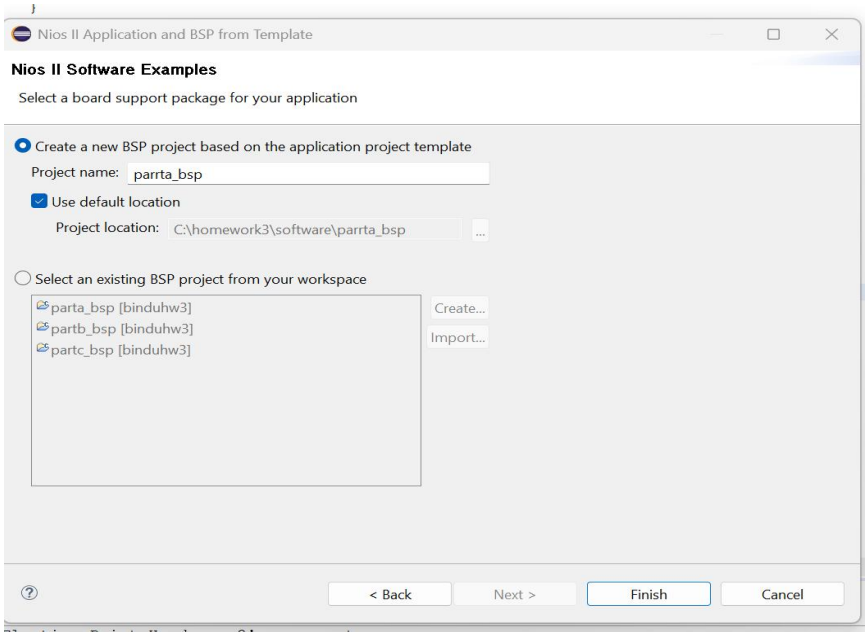
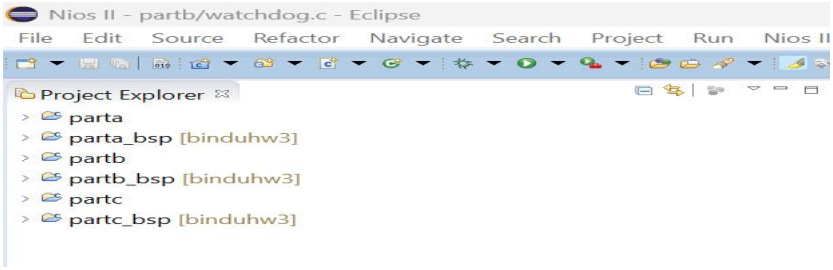


Figure 52: For creating bsp project

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

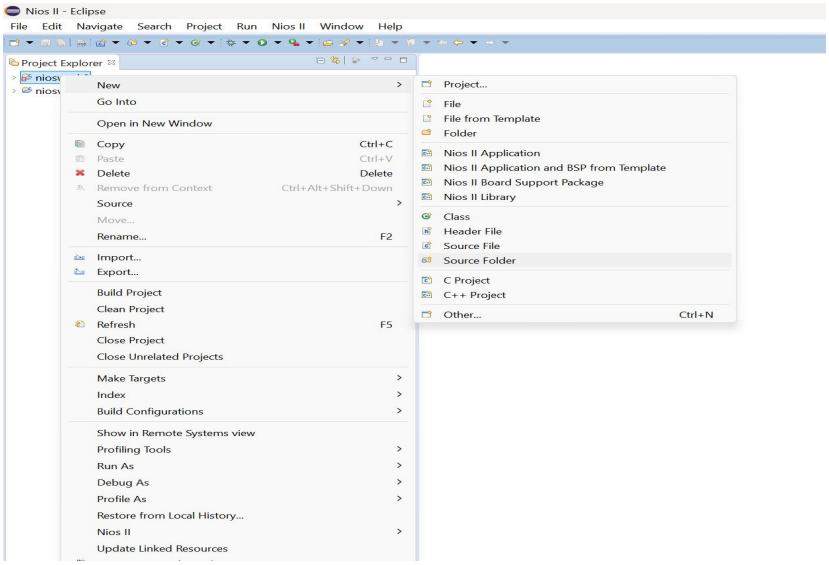


Figure 53: For creating source folder

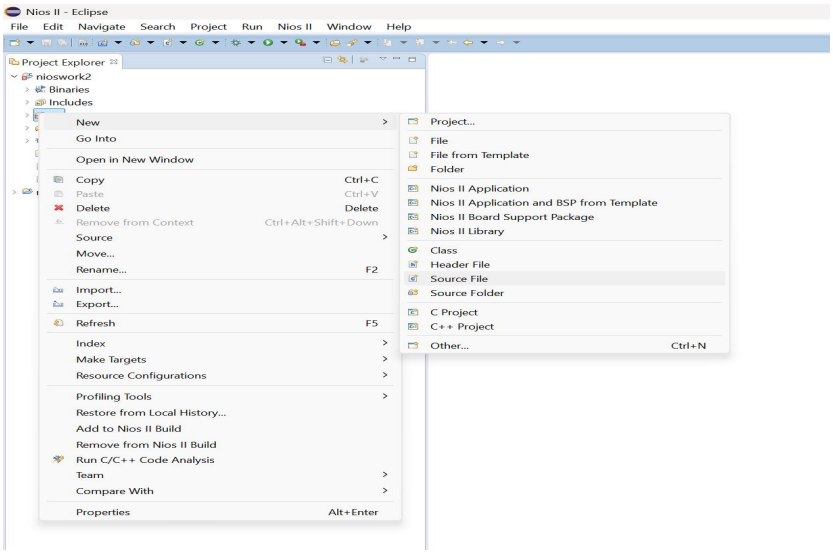


Figure 54: For creating source file

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 for 3 parts.

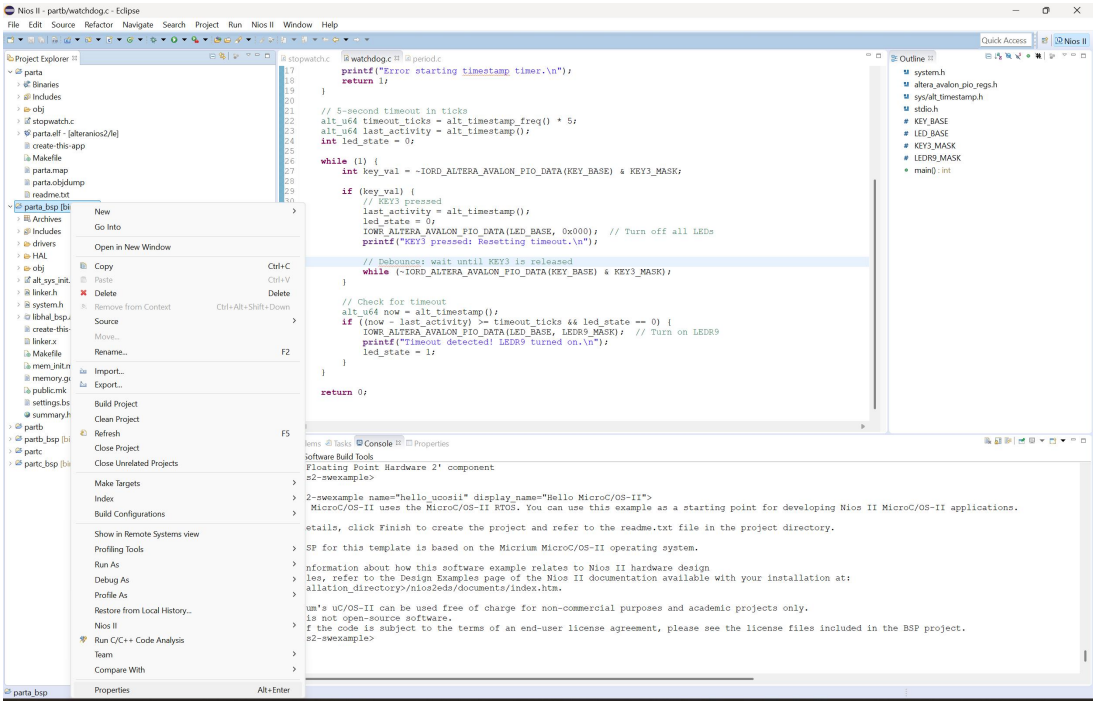


Figure 55: Right-clicking on the bsp project in Eclipse provides access to BSP settings.

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.

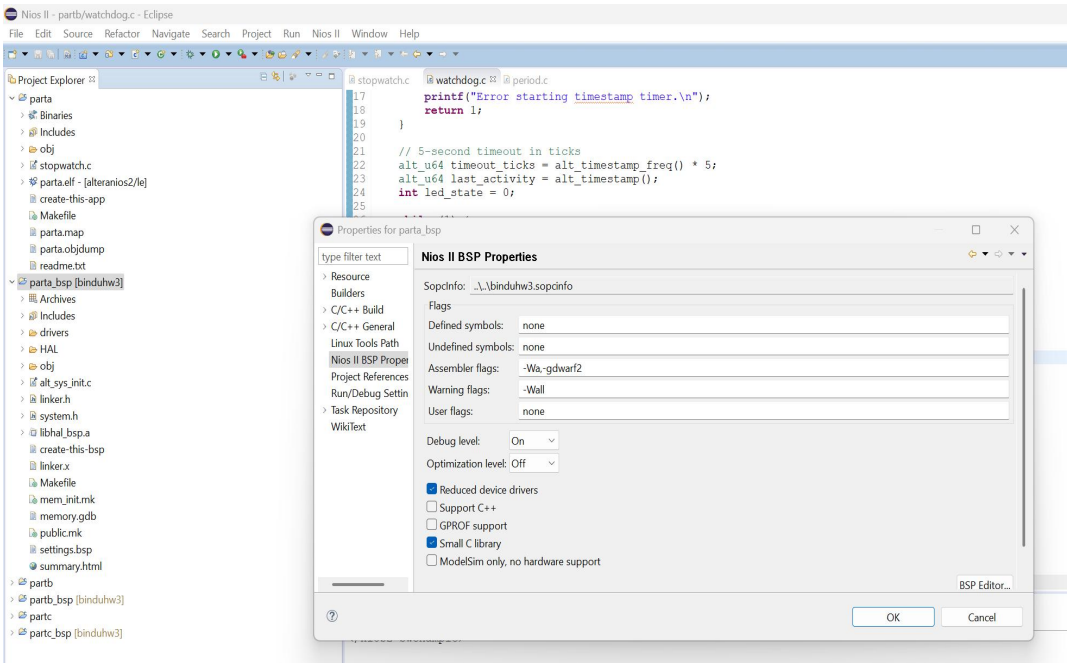


Figure 56: Nios BSP properties pat b

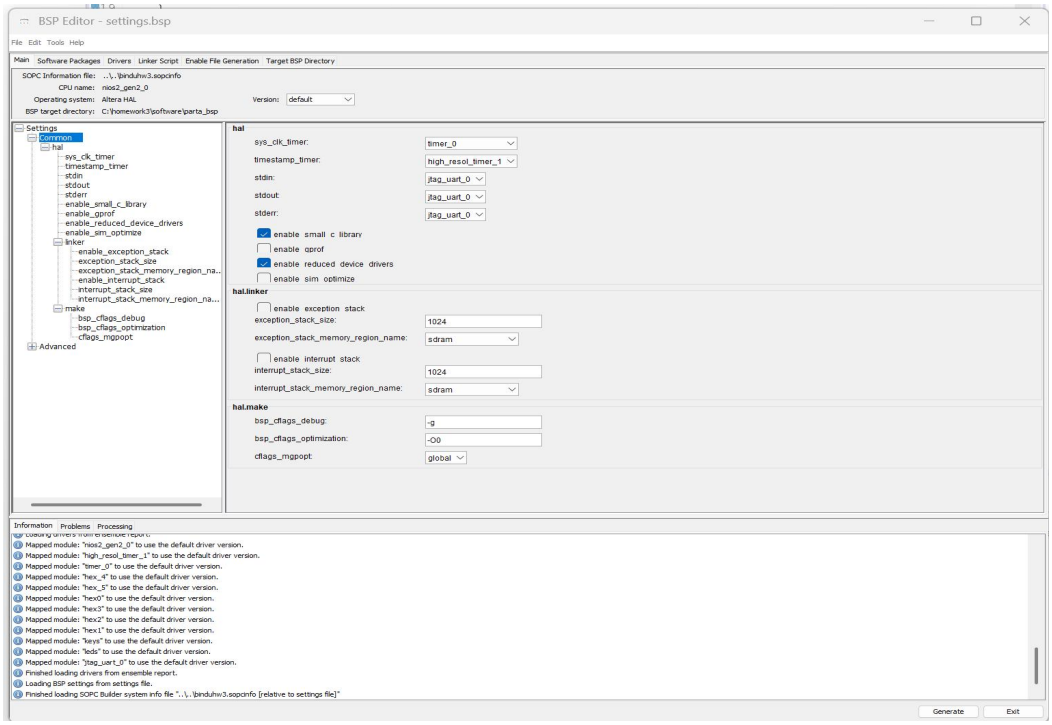


Figure 57: Nios BSP editor

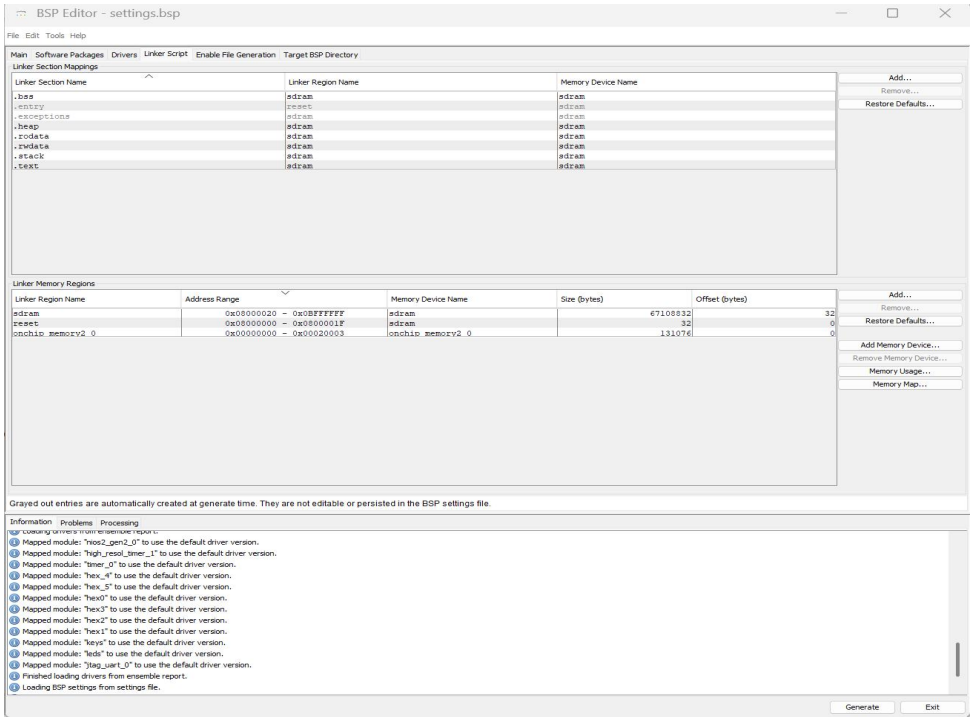


Figure 58: Nios II BSP linker scripts and click on generate

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

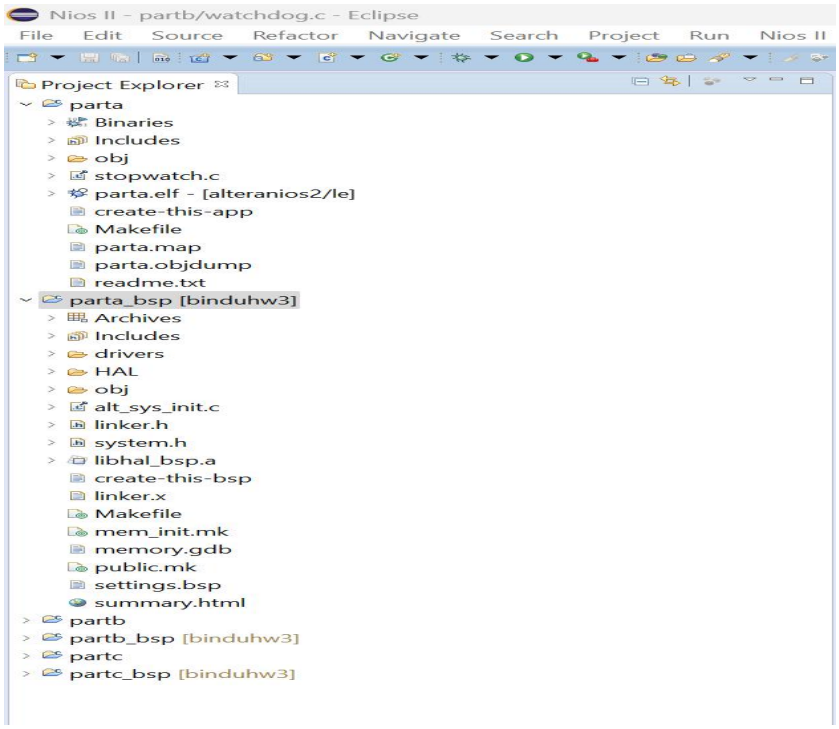


Figure 59: Artifacts of projects in Nios II

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.

For part a , generate the bsp and then build the projects and run the application.

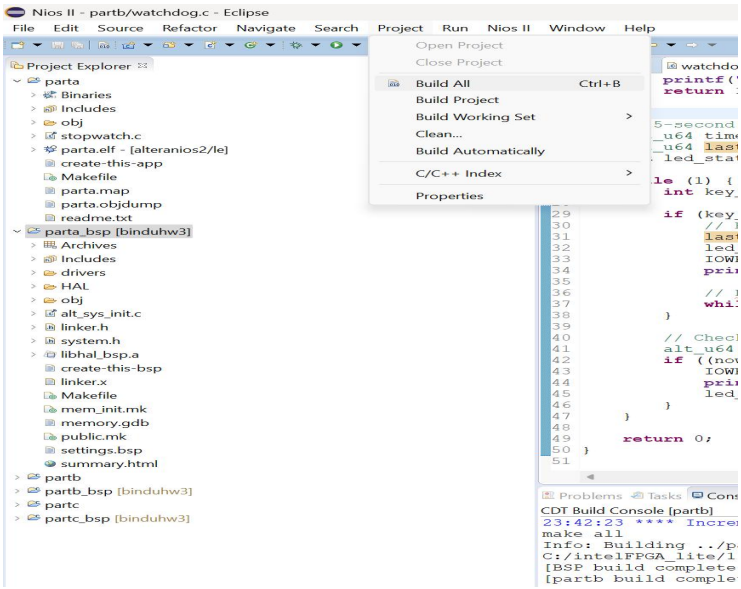


Figure 60: Building projects

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

Figure 61: Build complete succesfully for bsp part a 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.

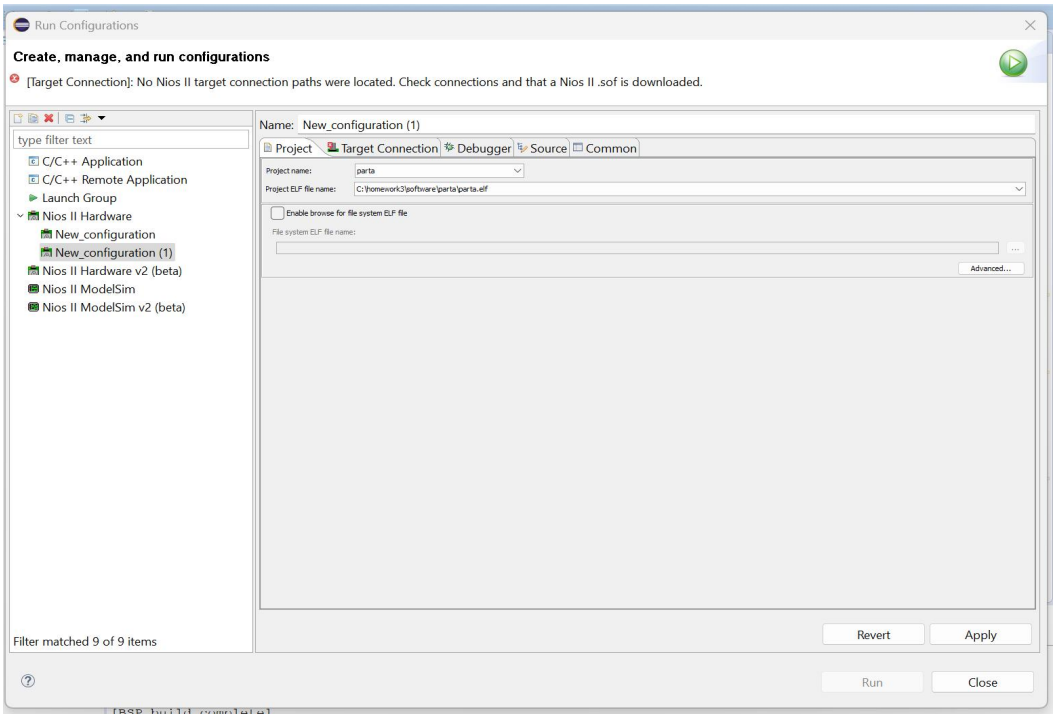


Figure 63: Run Configurations – part a Project Tab

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

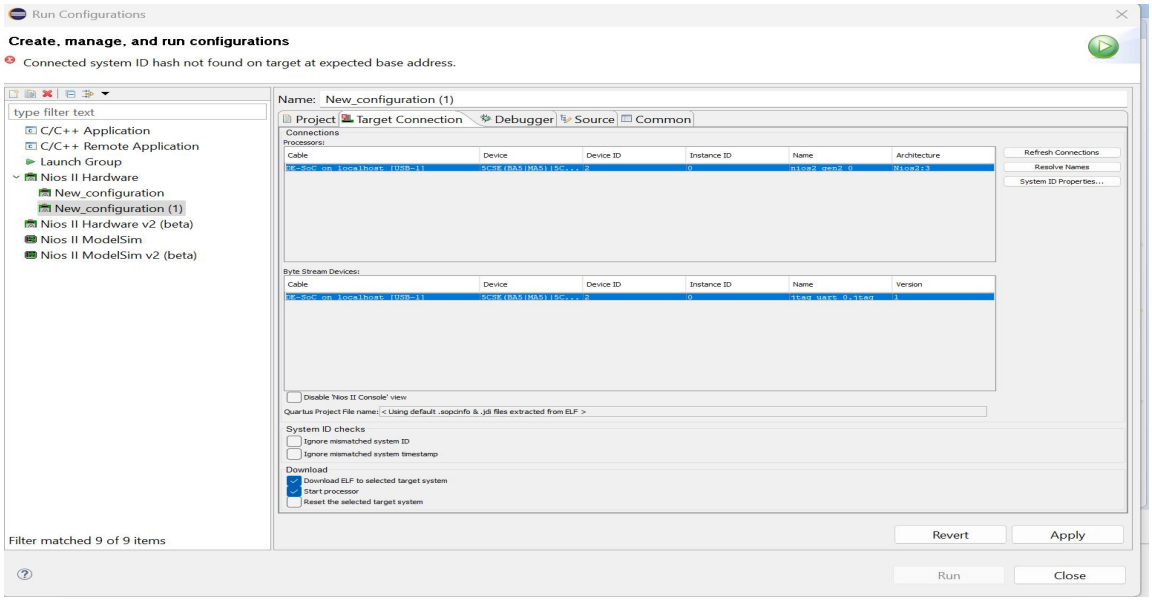


Figure 64: parta Run Configurations – Target Connection Tab

Clicked apply and then run the part a project for stopwatch mode. As soon as you run, it prints message in nios console to press key 2.

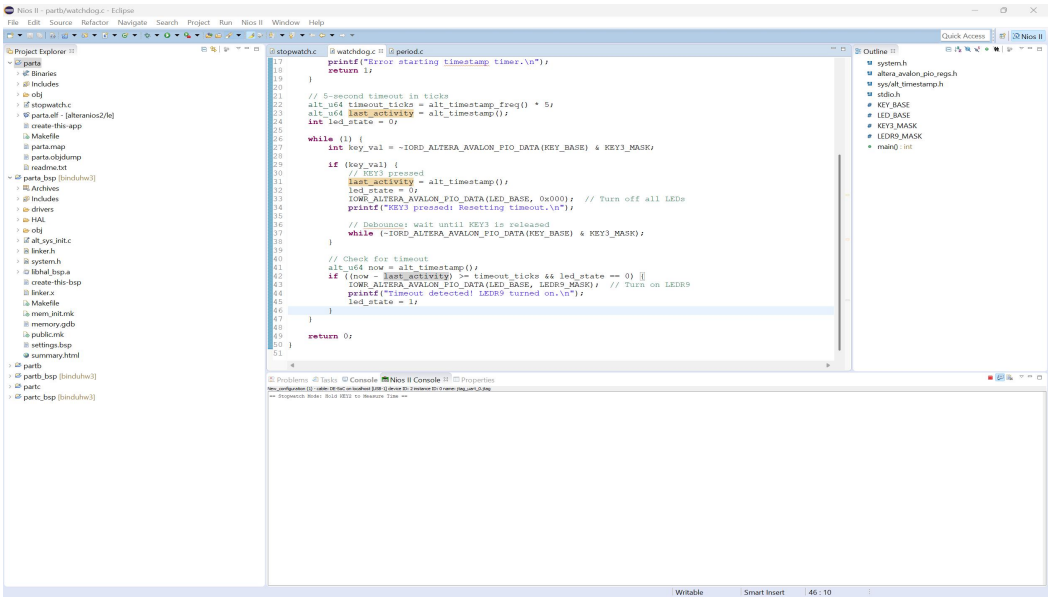


Figure 65: Eclipse IDE – Nios II Console and Source Code

After pressing on key2, I could see the ticks noted in nios console.

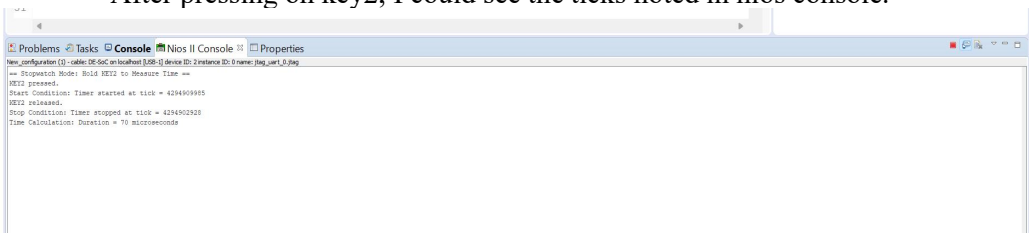


Figure 66: Eclipse IDE – Nios II Console

The time duration is printed on nios console and also on the board's seven segment display. LEDS 1,2, and 6 are also turned on after pressing the key2 at once.

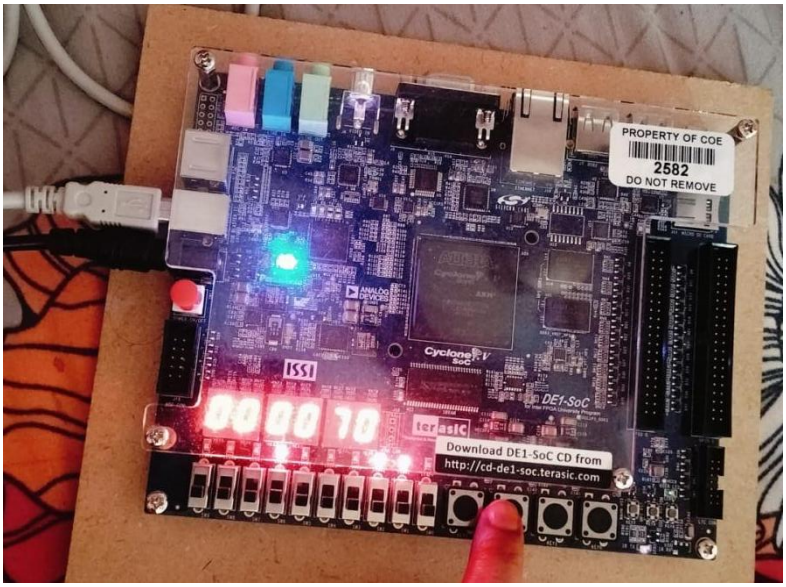


Figure 67: Press key on board and display output on display and LEDs response

Second time pressed the key2, and again time is noted in console and on board.

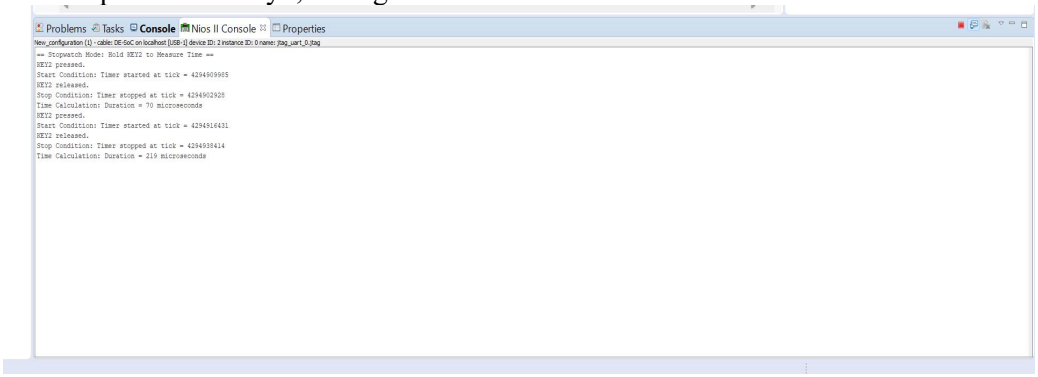


Figure 68 : Eclipse IDE – Nios II Console

Figure 69: second Press key on board and display output on display and LEDs response

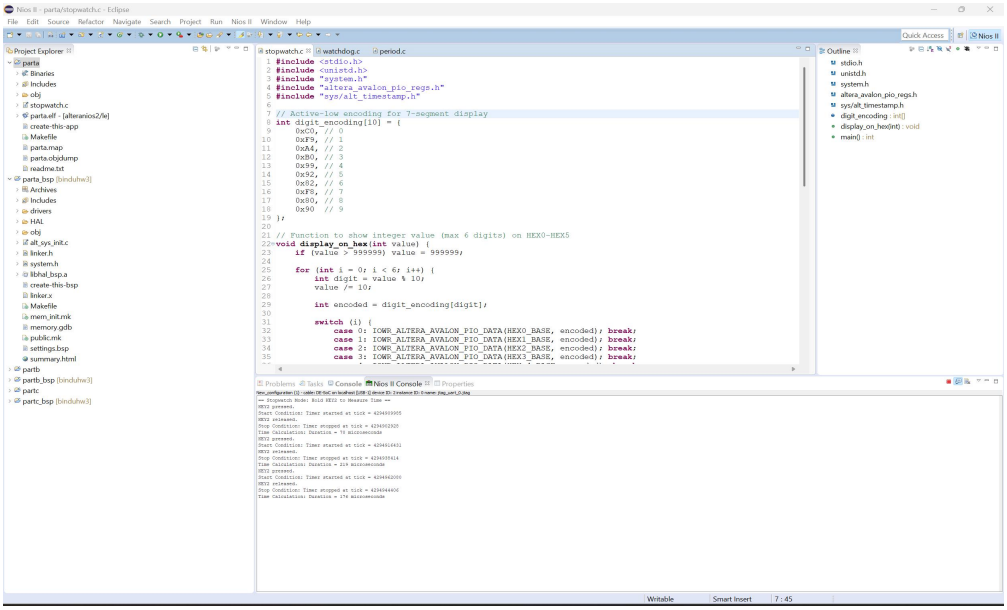


Figure 70: Part a Nios Output view nios console

For part b, this demonstrates the ability to detect user inactivity and trigger a specific action after a predefined period, simulating a watchdog function.

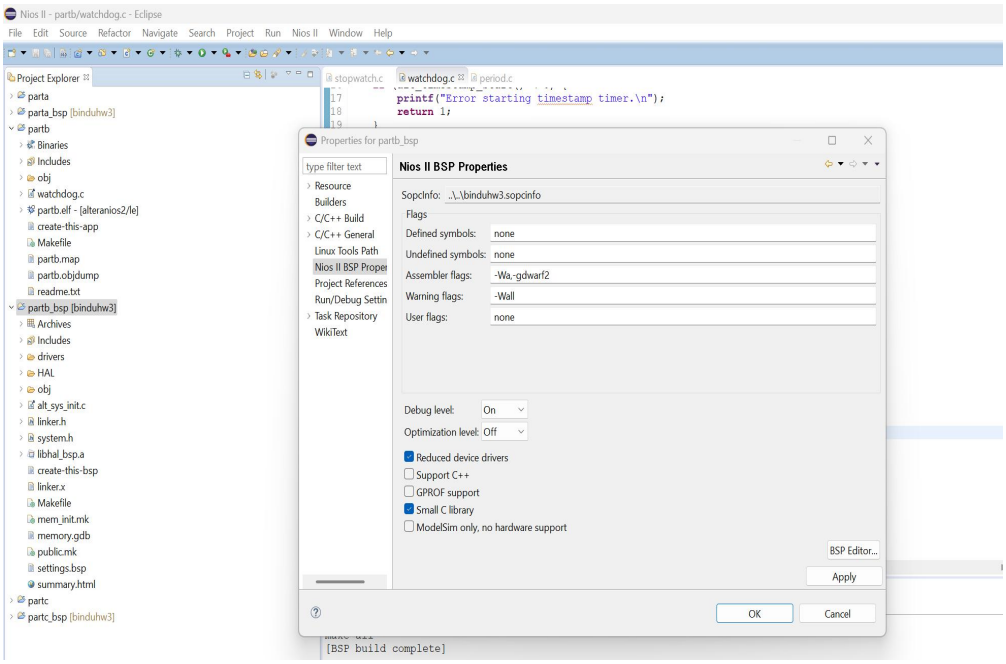


Figure 71: Nios BSP properties

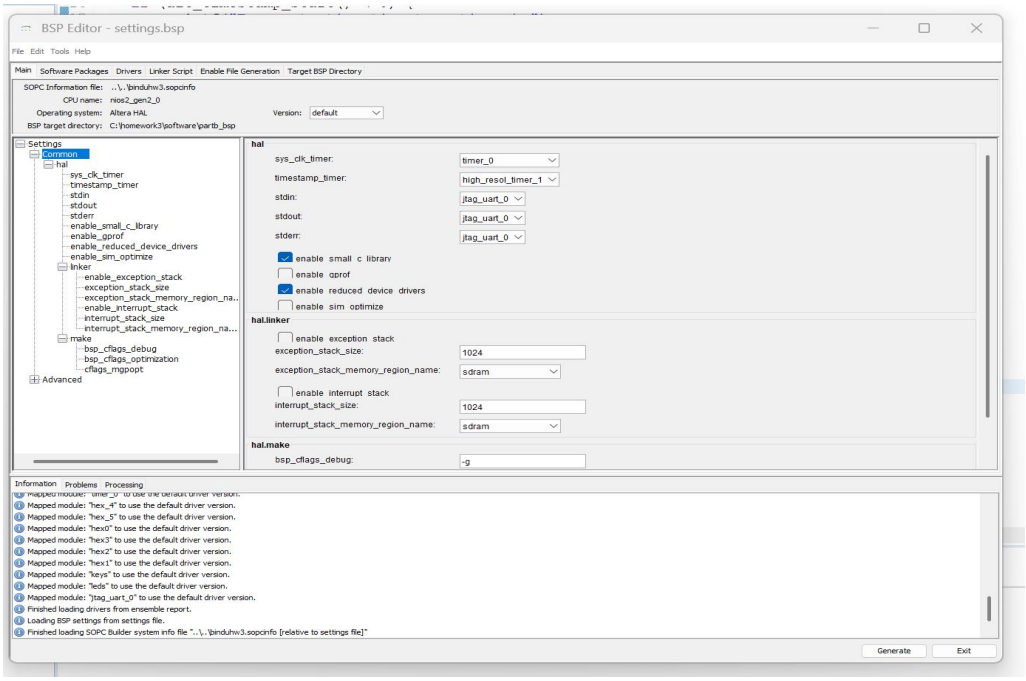


Figure 72: Nios BSP Editor

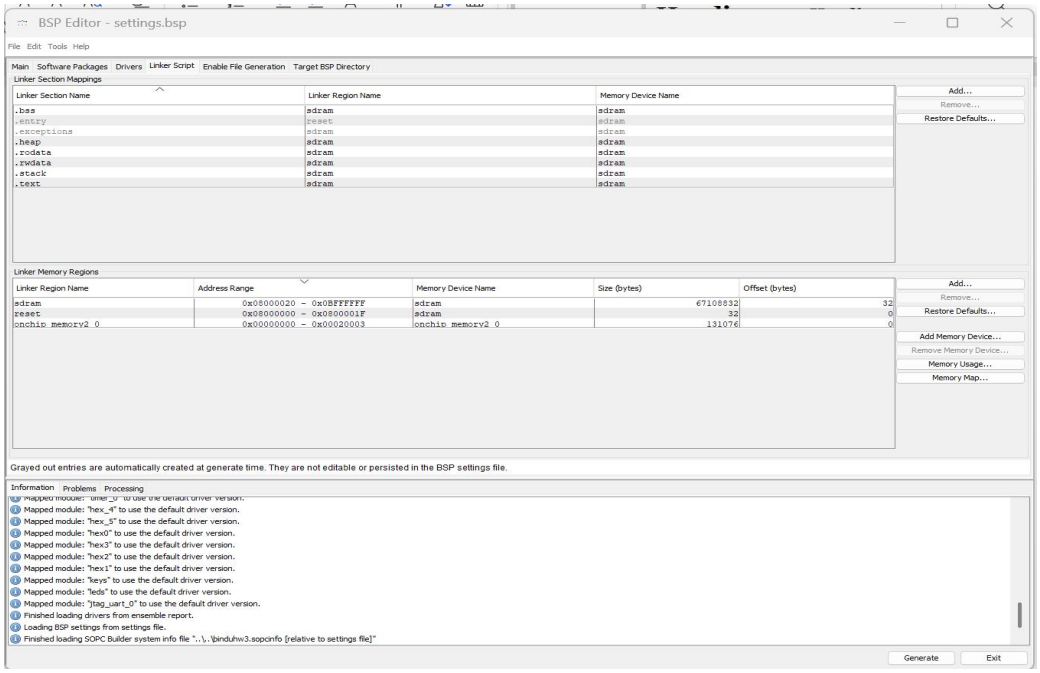


Figure 73: Nios BSP Editor and linker scripts

Click on generate and exit from bsp editor and save the nios BSP settings.

Build the part b application and bsp projects by right click and build option.

Figure 74: partb_bsp project build finished

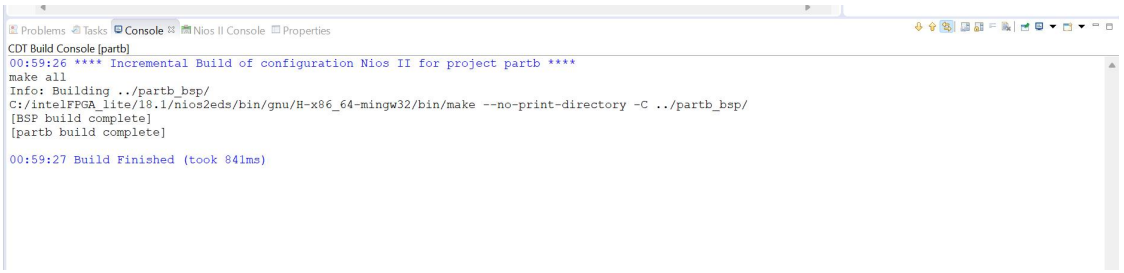


Figure 75: partb project build finished

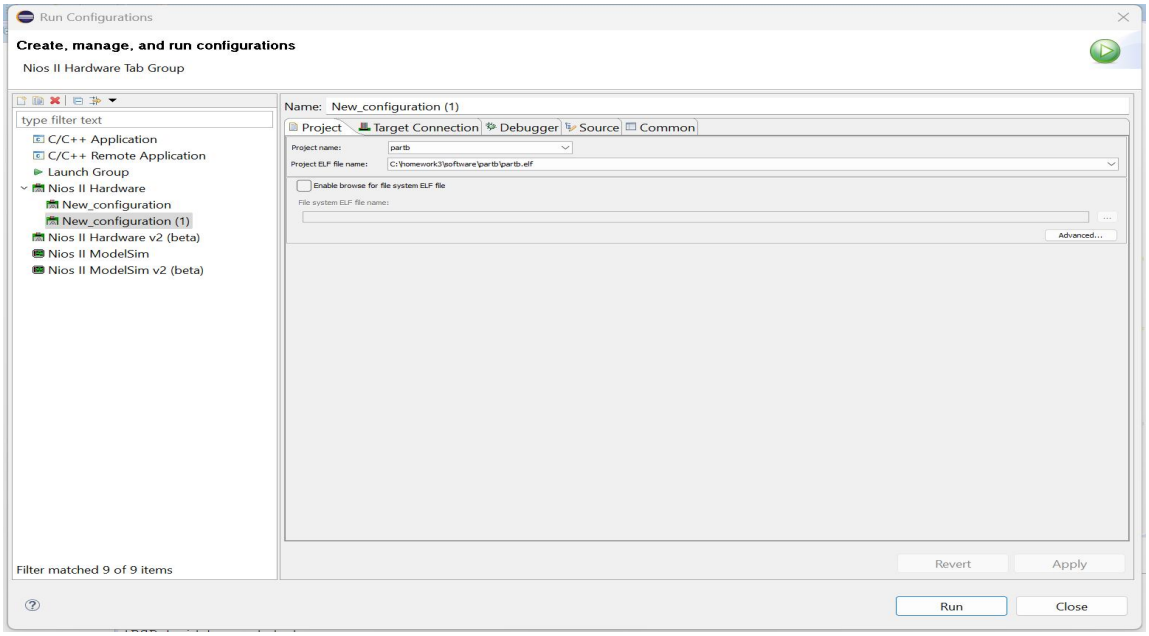


Figure 76: partb run configurations

After running the part b, this is printed on console that timeout detected and also led light LEDR9 also turned on in FPGA board. I observed that after 5 seconds, LEDR9 is turned on.

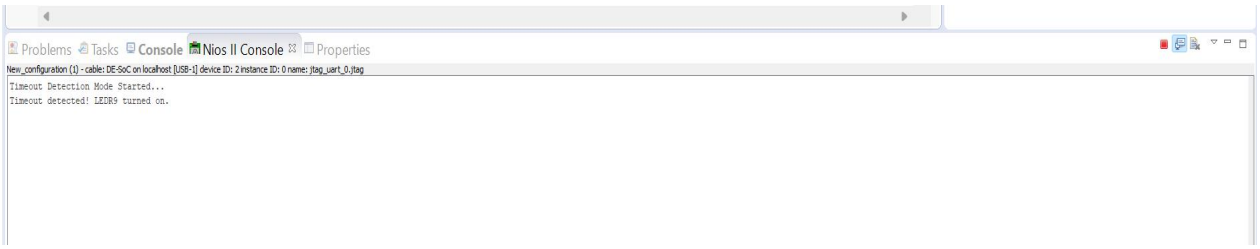


Figure 77: Eclipse IDE – Nios II Console output after running project

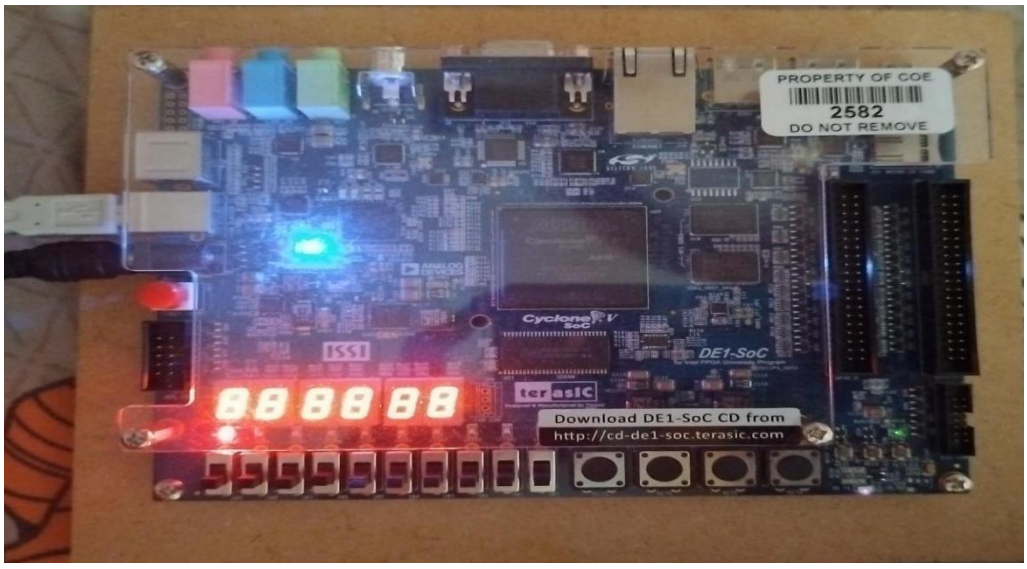


Figure 78: FPGA Board LEDR9 is turned on after running file

Key 3 is pressed, then the led light got turned off and after 5 secs, it turned on again. And output is seen in console.

Figure 79: Eclipse IDE – Nios II Console output after Key 3 press

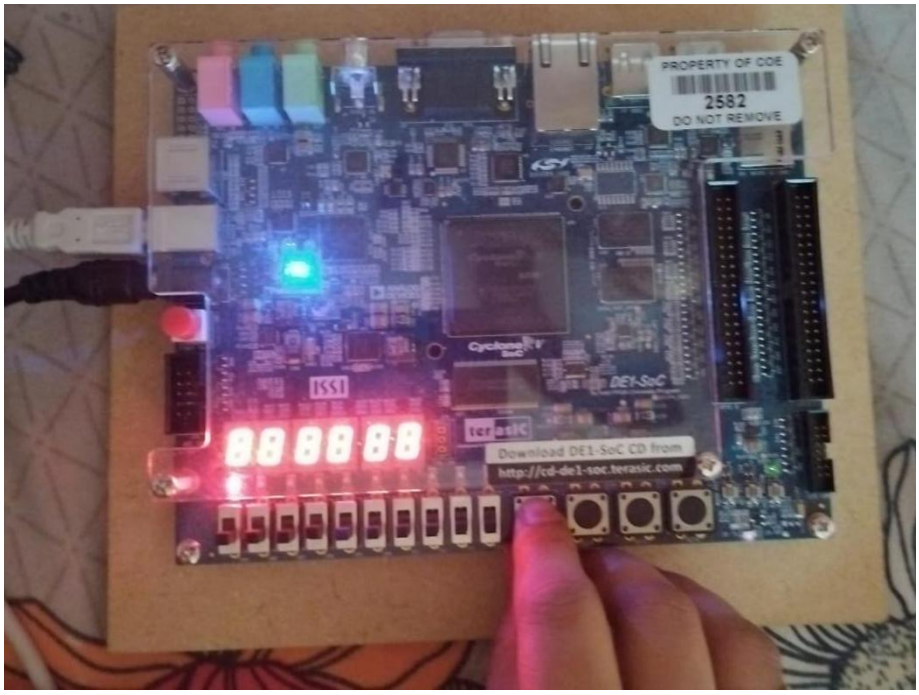


Figure 80: FPGA Board LEDR 9 is turned off after key3 press and turned on in 5 sec

Part c :

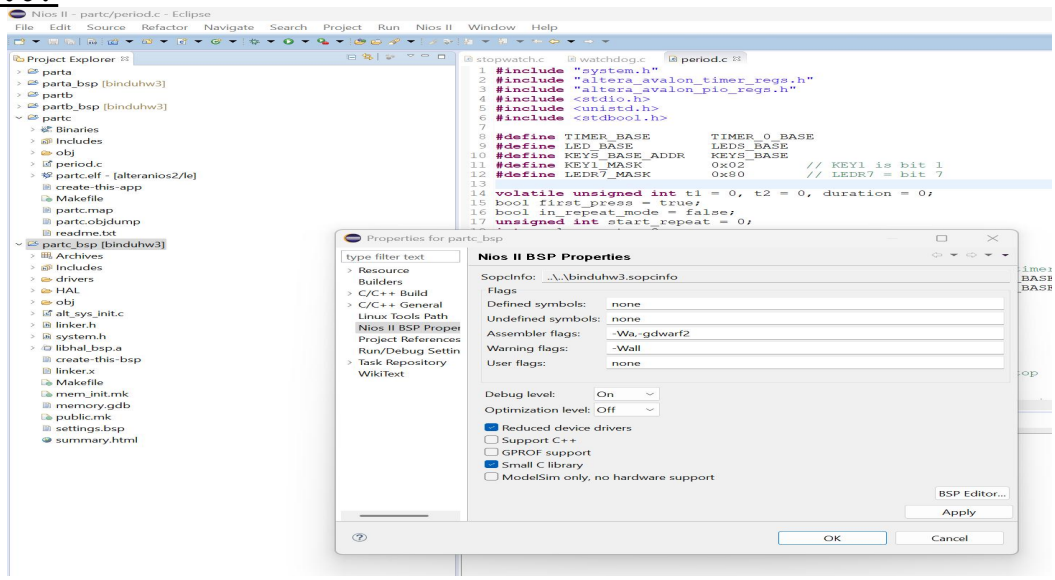


Figure 81: NIOS BSP settings

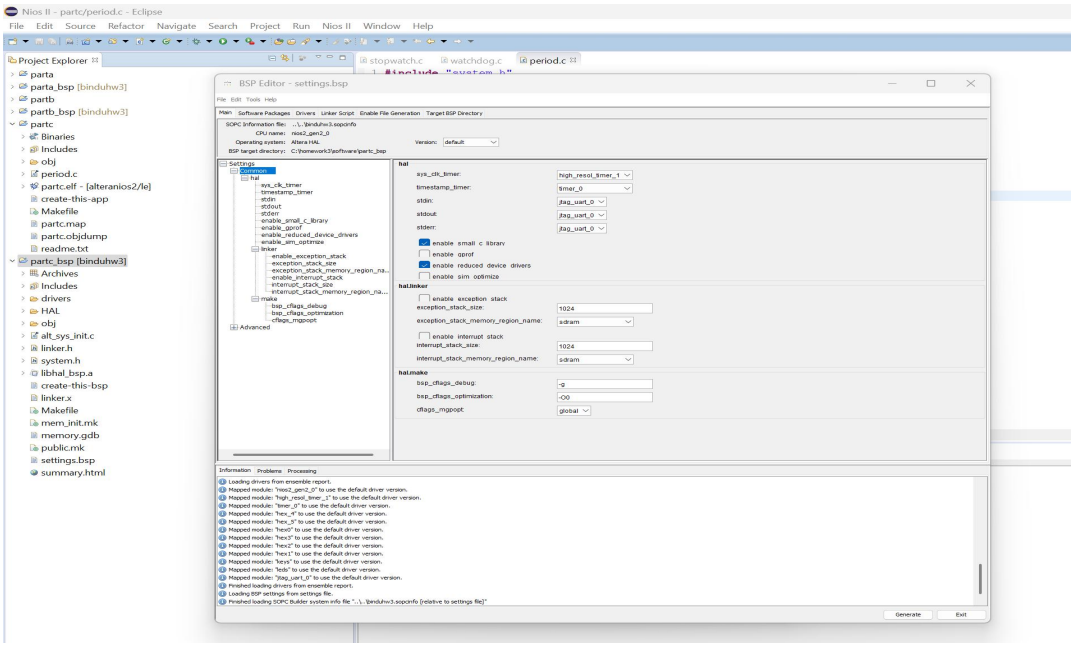


Figure 82: NIOS BSP editor part c

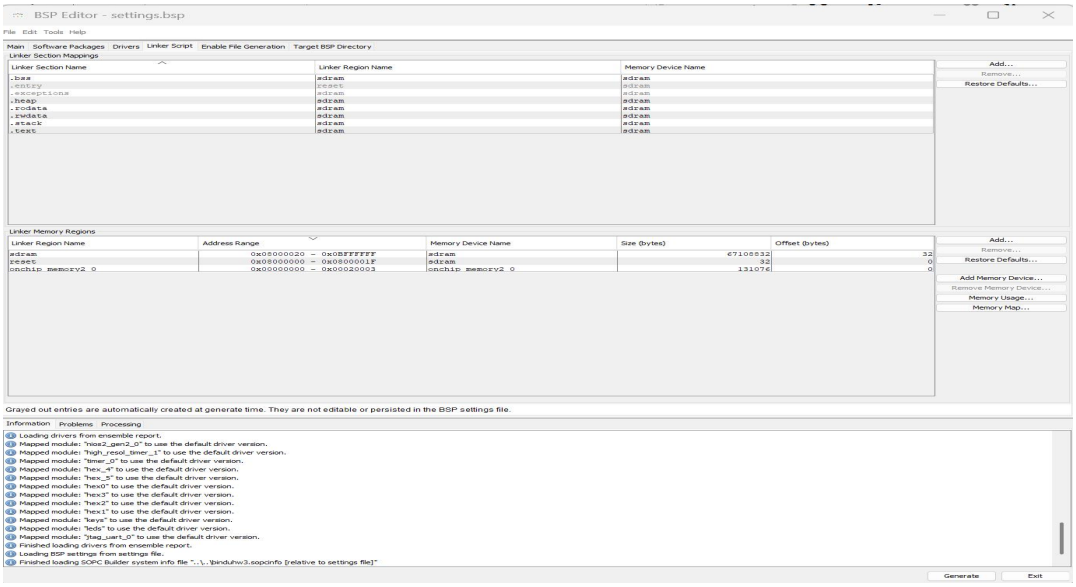


Figure 83: NIOS BSP linker scripts and generate

After this BSP editor exit and apply changes and click on OK. Then build the application and BSP project of part c.

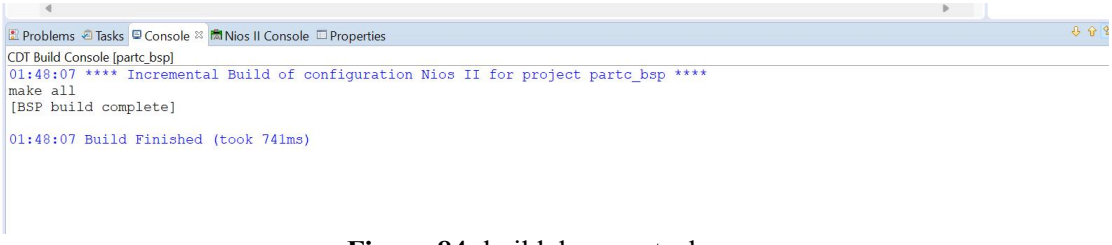


Figure 84: build done partc_bsp

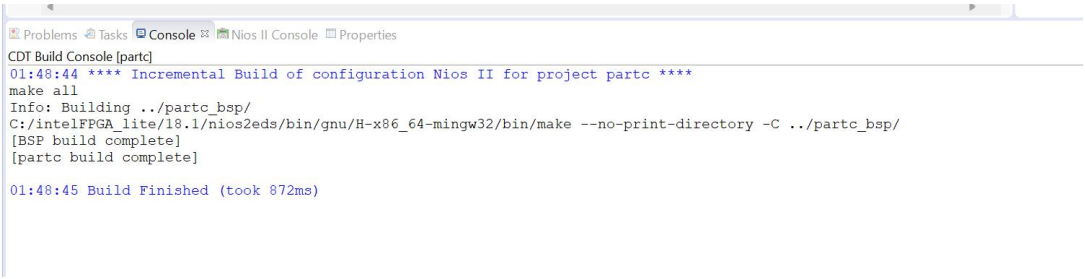


Figure 85: build done partc

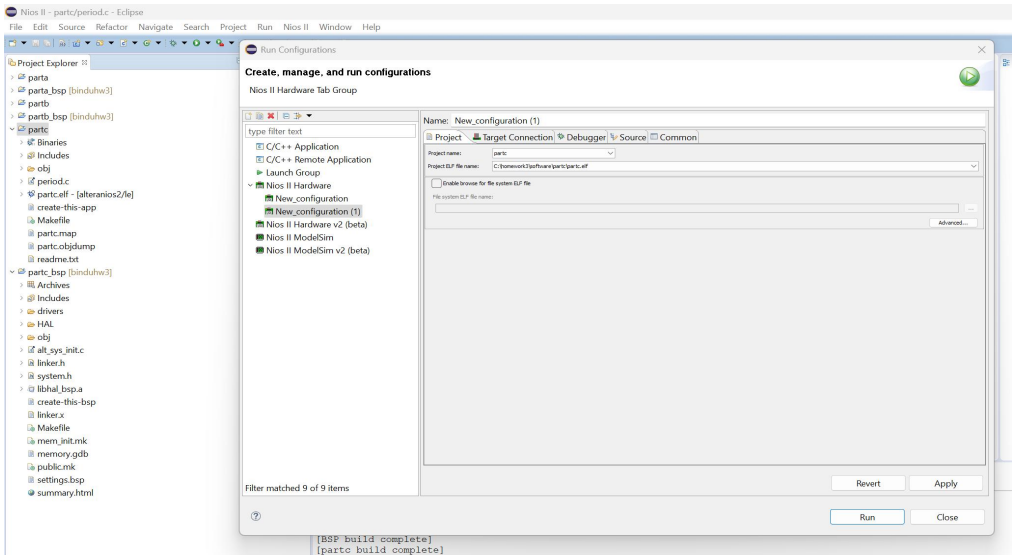


Figure 86: build done partc_bsp

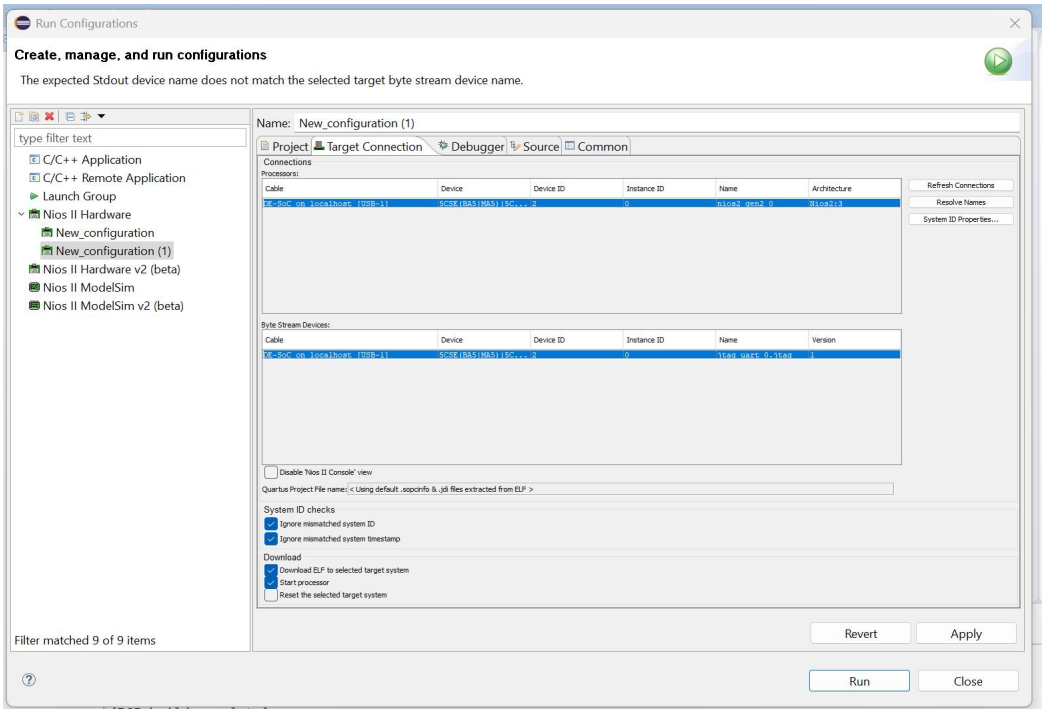


Figure 87: part c run configurations

After running the partc, this is printed in nios console

Figure 88: Part c nios console output after hitting run.

After pressing the key1, on board it notes the time and again I pressed the key1, so it noted the second press time. And it calculates the measured time duration for both the presses and turns on ledr7.

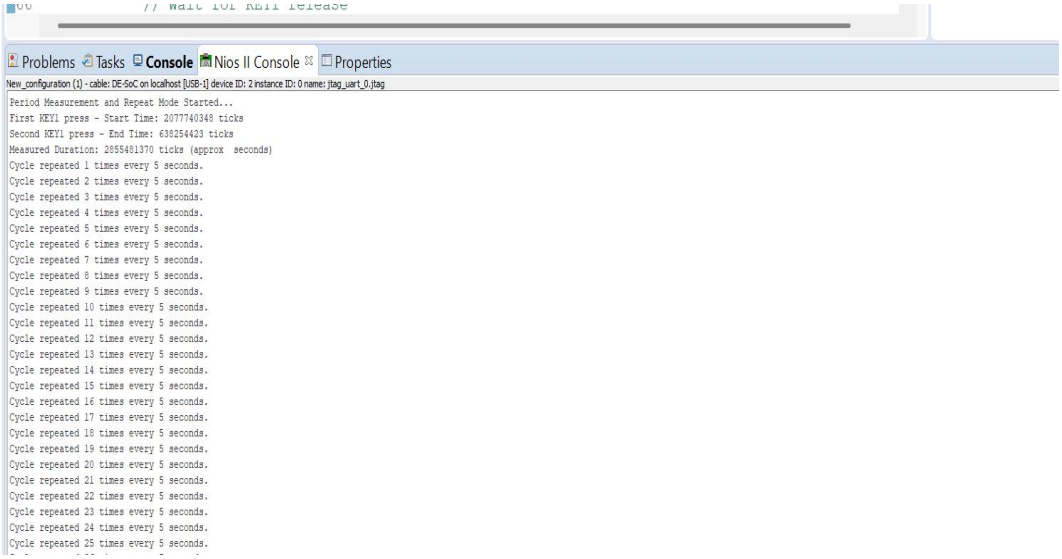


Figure 89: Part c nios console output after pressing double times key1 and nios II output

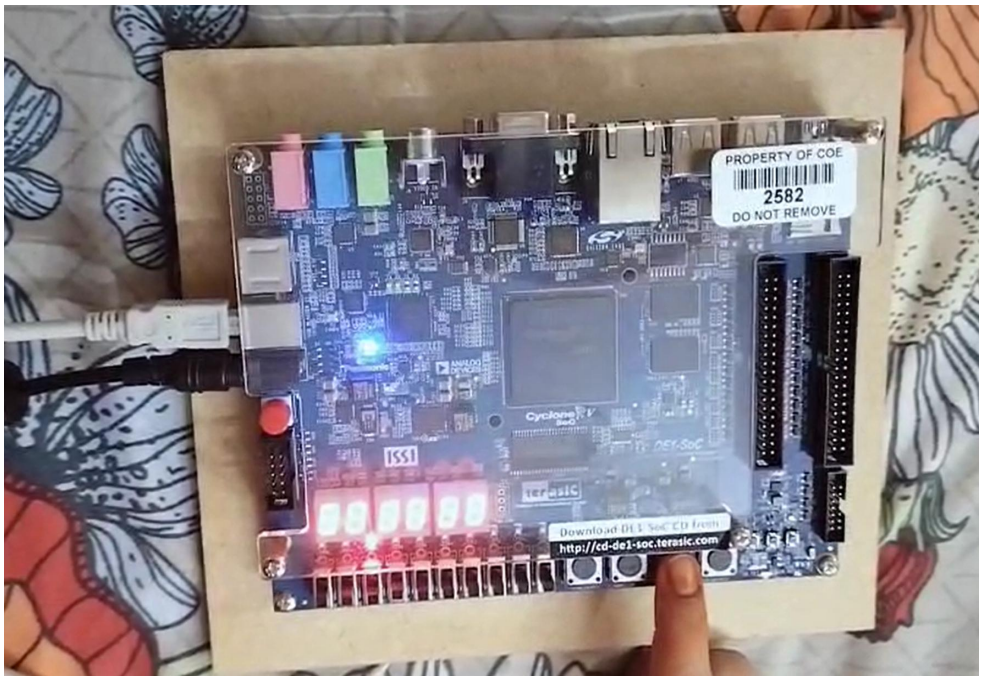


Figure 90: Part c Output in FPGA board and turned on LED

7 Analysis

1. Stopwatch Mode (Part A)

Purpose:

The purpose of the Stopwatch Mode was to measure the duration for which KEY2 is held down, and to display this duration in microseconds on both the Nios II console and the seven-segment displays. This functionality is crucial in demonstrating precise time measurement using hardware timers and software polling.

What I Learned:

In this part, I learned how to use the alt_timestamp() function, which utilizes the high-resolution timer on the FPGA to measure time in microseconds. I also gained experience in polling for user input (KEY2 press/release) and managing hardware timers effectively in an embedded system.

Analysis of Output:

The stopwatch mode worked as expected. When KEY2 was pressed, the system started recording the elapsed time, and the result was displayed both in the Nios II console and on the board's seven-segment displays. LED1, LED2, and LED6 were also ON to show the active state. However, I initially encountered a small precision issue in measuring the time duration. This was resolved by adjusting the polling frequency, which made the measurements more accurate.

2. Timeout Detection (Part B)

Purpose:

The purpose of the Timeout Detection feature was to detect a 5-second period of inactivity. If no user input occurs within 5 seconds, the system lights up LEDR9 as an indicator of timeout. Pressing KEY3 resets the inactivity timer.

What I Learned:

This part helped me understand how to use a system timer to track elapsed time and implement basic timeout logic. I also learned the importance of managing hardware timers for detecting inactivity and using polling to ensure the system remains responsive.

Analysis of Output:

The Timeout Detection successfully detected a 5-second inactivity period. After 5 seconds of no activity, LEDR9 was activated. Pressing KEY3 reset the timer and turned off LEDR9, which worked as expected. This demonstrated the ability to use a single timer for multiple functionalities—both as a stopwatch and a timeout detector.

3. Period Measurement and Repeat (Part C)

Purpose:

The goal of Part C was to measure the time duration between two presses of KEY1, compute the time difference, and repeat the cycle every 5 seconds. The duration is displayed on the seven-segment displays, and LEDR7 blinks to indicate the system is in the repeat cycle.

What I Learned:

In this part, I learned how to measure time intervals between two distinct events (KEY1 presses) and automatically repeat an action based on those intervals. This also taught me how to manage periodic actions, including hardware timer resets and output control via LEDs.

Analysis of Output:

The system correctly measured the interval between two KEY1 presses and displayed the time duration on the seven-segment displays. LEDR7 blinked every 5 seconds, indicating that the system was repeating the cycle automatically. The logic for capturing the first and second press was implemented correctly, and the system accurately displayed the measured time interval.

8 Results

This experiment successfully validated the Nios II timer system on the DE1-SoC FPGA.

Part A (Stopwatch): Accurately measured and displayed KEY2 hold durations in microseconds on HEX displays and LEDs, confirming high-resolution time capture.

Part B (Timeout): Reliably detected 5-second inactivity periods, illuminating LEDR9, with KEY3 resetting the timer.

Part C (Periodic): Precisely measured KEY1-driven intervals and demonstrated automatic 5-second cycle repetitions via LEDR7.

The project confirmed the system's precise timing capabilities and effective use of HAL and polling for real-time control.

9 Reflection on the Software Development

What I Learned in homework

--> Avalon Interval Timer
I discovered how to drive both the high-resolution “snapshot” timer (for microsecond precision via alt_timestamp_*() and direct register snapshots) and the millisecond “system” timer. Learning when to use alt_timestamp_start() versus manual IOWR_ALTERA_AVALON_TIMER_SNAP calls really clarified the difference between HAL abstractions and low-level register access.

--> Polling-Based
I got hands-on experience writing clear “press-hold-release” loops for the Stopwatch (Part A) and the Two-Press Timer (Part C), then reusing similar logic for the Watchdog (Part B). Organizing each mode’s polling in its own code block helped avoid cross-mode interference and kept the logic straightforward.

--> Timeout & Watchdog Logic
Implementing a 5-second inactivity detector using the same timer reinforced how to reuse hardware peripherals for multiple purposes. Resetting the “last activity” timestamp on KEY3 presses and comparing ticks in a loop drove home the basics of software watchdogs.

--> **Driving LEDs and Seven-Segment Displays**
Building and using a hex7seg[] lookup to map digits to seven-segment patterns, then writing a small display_decimal() helper, taught me how to combine binary indicators (LEDRs) with human-readable decimal output (HEX displays) in a single application.

--> Qsys/BSP Synchronization
I learned the hard way that any change in Qsys mandates a full BSP rebuild before the HAL APIs will work correctly. That dependency between hardware configuration and software stubs is crucial on the Nios II platform.

What Was Challenging to finish work

--> Synchronizing Qsys Settings & BSP
Every time I forgot to rebuild the BSP after toggling the timer’s “snapshot” or “always run” option, alt_timestamp_start() failed silently—leading to frustrating “timer not enabled” messages until I remembered the rebuild step.

--> Seven-Segment Indexing mismatch
My first mappings sent the least-significant digit to the wrong HEX physically on the board. Swapping the order in my HEX_BASE[] array fixed that once I rechecked the pin assignments.

--> Flooded JTAG-UART Console
Printing inside tight polling loops overwhelmed the UART buffer. Moving printf() calls to occur only after each complete measurement made the console logs readable and saved me from chasing phantom bugs.

Overall, Homework 3 drove home the flow on Nios II: from Qsys peripheral setup through BSP to clean, polling-based C code that blends hardware timing via LEDs, displays, and printf().

10 Conclusion

This Interval Timer Experiment successfully demonstrated a robust timing system on the DE1-SoC FPGA, built around a Nios II processor and its associated Hardware Abstraction Layer. All assignment objectives were precisely achieved: this included the accurate measurement of key-press durations in microseconds for the stopwatch functionality, the reliable detection of user inactivity after a five-second interval, and the establishment of automatic, periodic event repetitions following an initial two-press measurement. The project provided a comprehensive understanding of interval timer operation, low-level HAL API functions, and polling-based timing control, validating the system's capability for real-time applications.

11 Appendix

Top level module

```
module top (
   input  wire       CLOCK_50,    // 50 MHz clock
   input  wire [3:0]  KEY,        // Push buttons
   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;
```

```
    binduhw3 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),

        // 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
```

Main.c code

Part a - stopwatch.c

Code comments

```
// Start the high-resolution timer
alt_timestamp_start(); // This initializes the timer to capture time in microseconds

// Poll until KEY2 is released
while (KEY2 == 0) {
    // Continuously displays the elapsed time on the Nios II console and on seven-segment display
    // The timer is polled periodically for precise time capturing
}

// Stop the timer when KEY2 is released and display the final elapsed time
alt_timestamp_stop(); // This stops the timer and records the elapsed time
```

```c
#include <stdio.h>
#include <unistd.h>
#include "system.h"
#include "altera_avalon_pio_regs.h"
#include "sys/alt_timestamp.h"

// Active-low encoding for 7-segment display
int digit_encoding[10] = {
    0xC0, // 0
    0xF9, // 1
    0xA4, // 2
    0xB0, // 3
    0x99, // 4
    0x92, // 5
    0x82, // 6
    0xF8, // 7
    0x80, // 8
    0x90  // 9
};

// Function to show integer value (max 6 digits) on HEX0–HEX5
void display_on_hex(int value) {
    if (value > 999999) value = 999999;

    for (int i = 0; i < 6; i++) {
        int digit = value % 10;
        value /= 10;

        int encoded = digit_encoding[digit];

        switch (i) {
            case 0: IOWR_ALTERA_AVALON_PIO_DATA(HEX0_BASE, encoded); break;
            case 1: IOWR_ALTERA_AVALON_PIO_DATA(HEX1_BASE, encoded); break;
            case 2: IOWR_ALTERA_AVALON_PIO_DATA(HEX2_BASE, encoded); break;
            case 3: IOWR_ALTERA_AVALON_PIO_DATA(HEX3_BASE, encoded); break;
            case 4: IOWR_ALTERA_AVALON_PIO_DATA(HEX_4_BASE, encoded); break;
            case 5: IOWR_ALTERA_AVALON_PIO_DATA(HEX_5_BASE, encoded); break;
        }
    }
}

int main() {
    printf("== Stopwatch Mode: Hold KEY2 to Measure Time ==\n");

    if (alt_timestamp_start() < 0) {
        printf("Error: timestamp timer failed to start.\n");
        return 1;
    }


 int prev_keys = 0xF;
 alt_u64 start = 0, stop = 0, ticks = 0;
 int duration_us = 0;

 while (1) {
     int cur_keys = IORD_ALTERA_AVALON_PIO_DATA(KEYS_BASE);
     int key2_now = !(cur_keys & 0x04);
     int key2_prev = !(prev_keys & 0x04);
```

```
        if (key2_now && !key2_prev) {
            start = alt_timestamp();
            printf("KEY2 pressed.\n");
            printf("Start Condition: Timer started at tick = %lu\n", (unsigned long)start);
        }

        if (!key2_now && key2_prev) {
            stop = alt_timestamp();
            printf("KEY2 released.\n");
            printf("Stop Condition: Timer stopped at tick = %lu\n", (unsigned long)stop);

            ticks = (stop > start) ? (stop - start) : (start - stop);
            duration_us = (int)((double)ticks * 1e6 / alt_timestamp_freq());

            printf("Time Calculation: Duration = %d microseconds\n", duration_us);

            IOWR_ALTERA_AVALON_PIO_DATA(LEDS_BASE, duration_us & 0x3FF);
            display_on_hex(duration_us);
        }

        prev_keys = cur_keys;
        usleep(5000); // debounce
    }

    return 0;
}
```

Part b - watchdog.c

Code comments

```
// Polling the system timer to track inactivity time duration
if (elapsed_time > 5000000) { // 5 seconds time
   LEDR[9] = 1; // Timeout detected, LEDR9 is on
}

// Reset the timer when KEY3 is pressed
if (KEY3 == 0) {
   LEDR[9] = 0; // Turn off the timeout LED
}
```

Full code is written below

```c
#include "system.h"
#include "altera_avalon_pio_regs.h"
#include "sys/alt_timestamp.h"
#include <stdio.h>

// Define base addresses (adjust names based on your system.h)
#define KEY_BASE      KEYS_BASE
#define LED_BASE      LEDS_BASE
#define KEY3_MASK     0x08       // KEY3 is bit 3 (active low)
#define LEDR9_MASK    0x200      // LEDR9 is bit 9

int main() {
   printf("Timeout Detection Mode Started...\n");

   // Initialize timestamp timer
   if (alt_timestamp_start() < 0) {
      printf("Error starting timestamp timer.\n");
      return 1;
   }
// 5-second timeout in ticks

alt_u64 timeout_ticks = alt_timestamp_freq() * 5;
alt_u64 last_activity = alt_timestamp();
int led_state = 0;

while (1) {
   int key_val = ~IORD_ALTERA_AVALON_PIO_DATA(KEY_BASE) & KEY3_MASK;

   if (key_val) {
      // KEY3 pressed
      last_activity = alt_timestamp();
      led_state = 0;
      IOWR_ALTERA_AVALON_PIO_DATA(LED_BASE, 0x000);  // Turn off all LEDs
      printf("KEY3 pressed: Resetting timeout.\n");

      // Debounce: wait until KEY3 is released
      while (~IORD_ALTERA_AVALON_PIO_DATA(KEY_BASE) & KEY3_MASK);
   }

   // Check for timeout
   alt_u64 now = alt_timestamp();
   if ((now - last_activity) >= timeout_ticks && led_state == 0) {
      IOWR_ALTERA_AVALON_PIO_DATA(LED_BASE, LEDR9_MASK);  // Turn on LEDR9
      printf("Timeout detected! LEDR9 turned on.\n");
      led_state = 1;
   }
}

return 0;
}
```

Part c - Period.c

Code comments

```
// Store the time of the first KEY1 press
if (KEY1 == 0) {
    first_press_time = alt_timestamp(); // Record time when the first press occurs
}

// After the second press, calculate the duration
if (KEY1 == 0) {
    second_press_time = alt_timestamp(); // Record time of the second press
    duration = second_press_time - first_press_time; // Calculate the time interval between the presses
    display_duration(duration); // Display the measured duration on the seven-segment display
}
```

Full code written below

```
#include "system.h"
#include "altera_avalon_timer_regs.h"
#include "altera_avalon_pio_regs.h"
#include <stdio.h>
#include <unistd.h>
#include <stdbool.h>

#define TIMER_BASE      TIMER_0_BASE
#define LED_BASE        LEDS_BASE
#define KEYS_BASE_ADDR   KEYS_BASE
#define KEY1_MASK       0x02      // KEY1 is bit 1
#define LEDR7_MASK      0x80      // LEDR7 = bit 7

volatile unsigned int t1 = 0, t2 = 0, duration = 0;
bool first_press = true;
bool in_repeat_mode = false;
unsigned int start_repeat = 0;
int cycle_count = 0;

// Timer Read Function
unsigned int read_timer() {
    IOWR_ALTERA_AVALON_TIMER_SNAPL(TIMER_BASE, 0); // Latch timer snapshot
    unsigned int low  = IORD_ALTERA_AVALON_TIMER_SNAPL(TIMER_BASE);
    unsigned int high = IORD_ALTERA_AVALON_TIMER_SNAPH(TIMER_BASE);
    return (high << 16) | low;
```

```
}

int main() {
    // Turn off all LEDs
    IOWR_ALTERA_AVALON_PIO_DATA(LED_BASE, 0x00);

    // Configure timer in continuous mode (32-bit)
    IOWR_ALTERA_AVALON_TIMER_CONTROL(TIMER_BASE, 0x04); // Stop
    IOWR_ALTERA_AVALON_TIMER_PERIODL(TIMER_BASE, 0xFFFF);
    IOWR_ALTERA_AVALON_TIMER_PERIODH(TIMER_BASE, 0xFFFF);
    IOWR_ALTERA_AVALON_TIMER_CONTROL(TIMER_BASE, 0x07); // Start

    printf("Period Measurement and Repeat Mode Started...\n");

    while (1) {
        int keys = IORD_ALTERA_AVALON_PIO_DATA(KEYS_BASE_ADDR);

        // Check for KEY1 press (active-low)
        if (!(keys & KEY1_MASK)) {
            usleep(50000); // Debounce

            if (first_press) {
                t1 = read_timer();
                printf("First KEY1 press - Start Time: %u ticks\n", t1);
                first_press = false;
            } else {
                t2 = read_timer();
                if (t2 >= t1)
                    duration = t2 - t1;
                else
                    duration = (0xFFFFFFFF - t1) + t2;

                printf("Second KEY1 press - End Time: %u ticks\n", t2);
                printf("Measured Duration: %u ticks (approx %.3f seconds)\n", duration, duration / 1e6);

                start_repeat = read_timer();
                in_repeat_mode = true;
                first_press = true;
            }

            // Wait for KEY1 release
            while (!(IORD_ALTERA_AVALON_PIO_DATA(KEYS_BASE_ADDR) & KEY1_MASK));
        }

        // Auto-repeat logic
            if (in_repeat_mode) {
                unsigned int now = read_timer();
                unsigned int elapsed = (now >= start_repeat)
                                       ? (now - start_repeat)
                                       : ((0xFFFFFFFF - start_repeat) + now);

                if (elapsed >= 5000000) { // 5 seconds
                    cycle_count++;

                    printf("Cycle repeated %d times every 5 seconds.\n", cycle_count);

                    // Blink LEDR7
                    IOWR_ALTERA_AVALON_PIO_DATA(LED_BASE, LEDR7_MASK);
                    usleep(500000); // LED ON 0.5 sec
                    IOWR_ALTERA_AVALON_PIO_DATA(LED_BASE, 0x00);

                    start_repeat = read_timer(); // Reset timer
                }
            }
        }

        return 0;
    }
```

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:
#                binduhw3_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 top
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_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_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_global_assignment -name VERILOG_FILE binduhw3/synthesis/binduhw3.v
set_global_assignment -name QIP_FILE binduhw3/synthesis/binduhw3.qip
set_global_assignment -name VERILOG_FILE top.v
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_instance_assignment -name PARTITION_HIERARCHY root_partition -to | -section_id
Top
```