Laboratory 1

Getting Started with the DE10-Standard FPGA

Introduction

This laboratory introduces students to the initial steps of working with the DE10-Standard FPGA development board using the Quartus Prime software. The activity guides students through the entire process of creating a new project, writing and compiling HDL code, assigning physical pins, and programming the FPGA.

Students will implement a basic ALU (Arithmetic Logic Unit) design that performs simple arithmetic and logic operations. The Top module includes sequential logic that allows data to be loaded from physical peripherals (switches and buttons) and displays the results using the onboard LEDs.

This hands-on lab aims to familiarize students with the Quartus Prime development flow and the practical steps required to implement and test a digital circuit on an FPGA board.

Objectives

General objective:

To design, implement, and program a simple ALU circuit on the DE10-Standard FPGA using Quartus Prime, through the creation of a complete project flow including module design, compilation, pin assignment, and device programming.

Specific objectives:

1. To create and configure a new Quartus Prime project for the DE10-Standard board.
2. To implement a combinational ALU module and a sequential Top module using SystemVerilog.
3. To assign the appropriate physical pins on the FPGA for each input and output signal, based on the board's peripherals.
4. To compile the design and verify the absence of errors through simulation and synthesis reports.
5. To program the DE10-Standard FPGA and validate the operation of the ALU using physical switches and LEDs.

Materials

- Quartus Prime 24.1 Standard Edition.
- ModelSim – Intel FPGA Starter Edition 10.5b.
- DE10-Standard FPGA development board.

Procedure

New Project

1. Open the Quartus Prime software.

2. Go to the top menu and select: **File → New Project Wizard**, as shown in Figure 1.

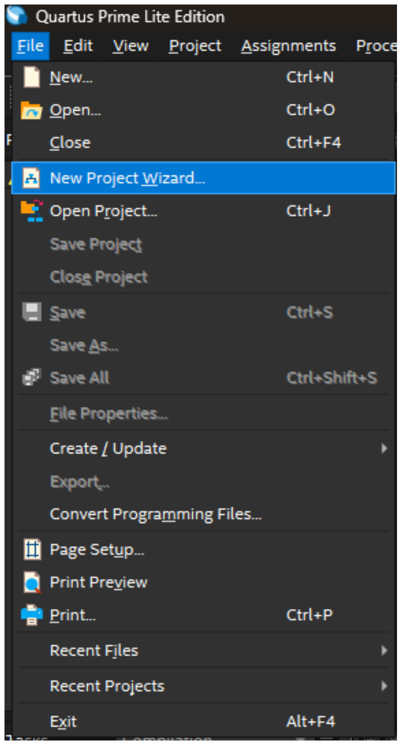


Figure 1. New Project Wizard.

3. In the New Project Wizard, click **Next**.

4. In the Directory, Name, Top-Level Entity, click the three dots next to "What is the working directory for this project?", then **browse and select** the folder **LabFPGA1** located at Desktop\LabFPGA_DE10St. In the second line, enter the project name: **LabFPGA1**, then click **Next**.

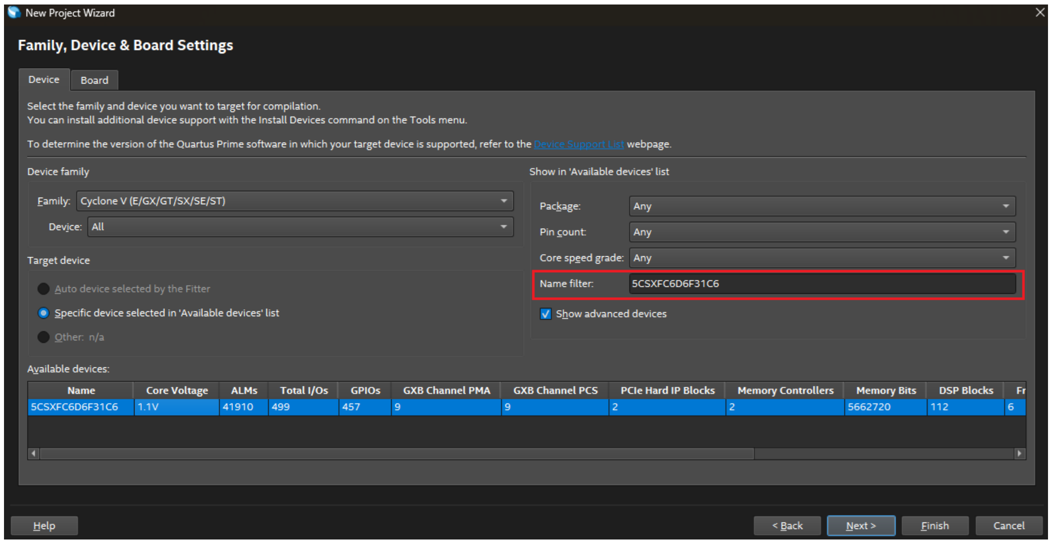
5. In Project Type, select the option **Empty project**, and click **Next**.

6. In **Add Files**, no files will be added at this stage, so just click **Next**.

7. In Family, Device & Board Settings, go to the right-side panel under "Show in 'Available devices' list", and click in the "Name filter" **textbox**.

8. Type the following device name: **5CSXFC6D6F31C6**, which corresponds to the FPGA to be used.

9. A single device should appear under "Available devices". Make sure to select it as shown in Figure 2. Then click **Next**.



New Project Wizard

Family, Device & Board Settings

Device | Board

Select the family and device you want to target for compilation.
You can install additional device support with the Install Devices command on the Tools menu.

To determine the version of the Quartus Prime software in which your target device is supported, refer to the Device Support List webpage.

Device family

Family: Cyclone V (E/GX/GT/SX/SE/ST)

Device: All

Show in 'Available devices' list

Package: Any

Pin count: Any

Core speed grade: Any

Name filter: 5CSXFC6D6F31C6

[✓] Show advanced devices

Target device

Auto device selected by the Fitter

● Specific device selected in 'Available devices' list

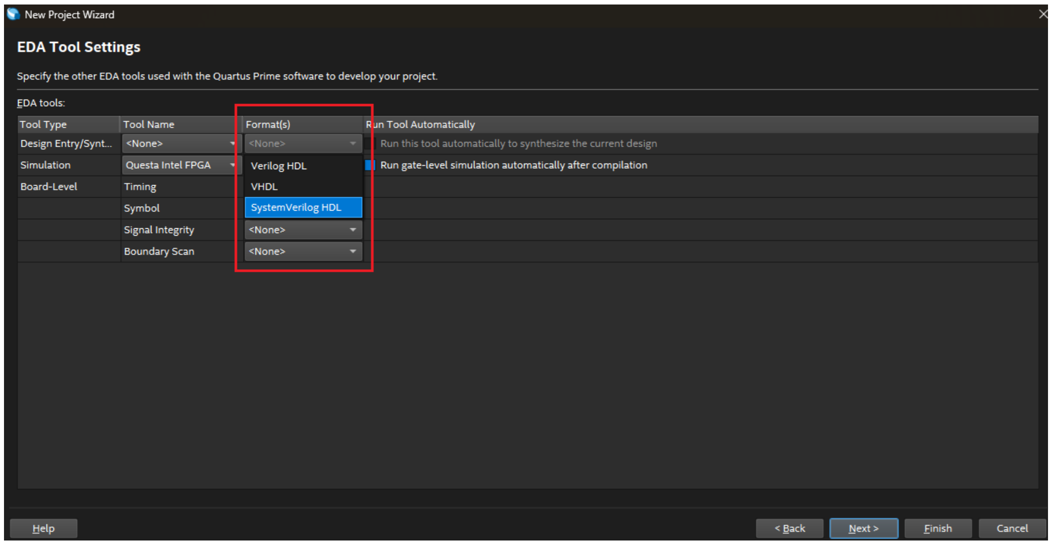
Other: n/a

Available devices:

Name	Core Voltage	ALMs	Total I/Os	GPIOs	GXB Channel PMA	GXB Channel PCS	PCIe Hard IP Blocks	Memory Controllers	Memory Bits	DSP Blocks	Fr
5CSXFC6D6F31C6	1.1V	41910	499	457	9	9	2	2	5662720	112	6

Figure 2. Device.

10. In the EDA Tool Settings, on the second row under the Format(s) column, select **SystemVerilog HDL** as shown in Figure 3. Click **Next** to continue.



New Project Wizard

EDA Tool Settings

Specify the other EDA tools used with the Quartus Prime software to develop your project.

EDA tools:

Tool Type	Tool Name	Format(s)	Run Tool Automatically
Design Entry/Synt...	<None>	<None>	Run this tool automatically to synthesize the current design
Simulation	Questa Intel FPGA	Verilog HDL / VHDL / SystemVerilog HDL	Run gate-level simulation automatically after compilation
Board-Level	Timing		
	Symbol		
	Signal Integrity	<None>	
	Boundary Scan	<None>	

Figure 3. EDA Tool Settings

11. In Summary, verify that the selected device matches the one from step 9 and that the simulation language is SystemVerilog. Finally, click **Finish**. If a pop-up window appears, select **Yes**.

Create module

12. In the top menu, select **New**, then choose **SystemVerilog HDL File** under Design Files, as shown in Figure 4. Click **OK** to finish.

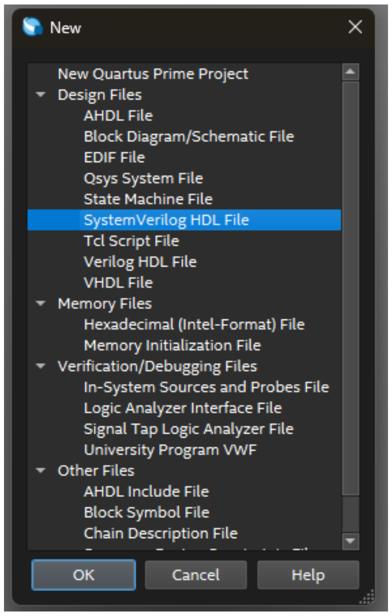
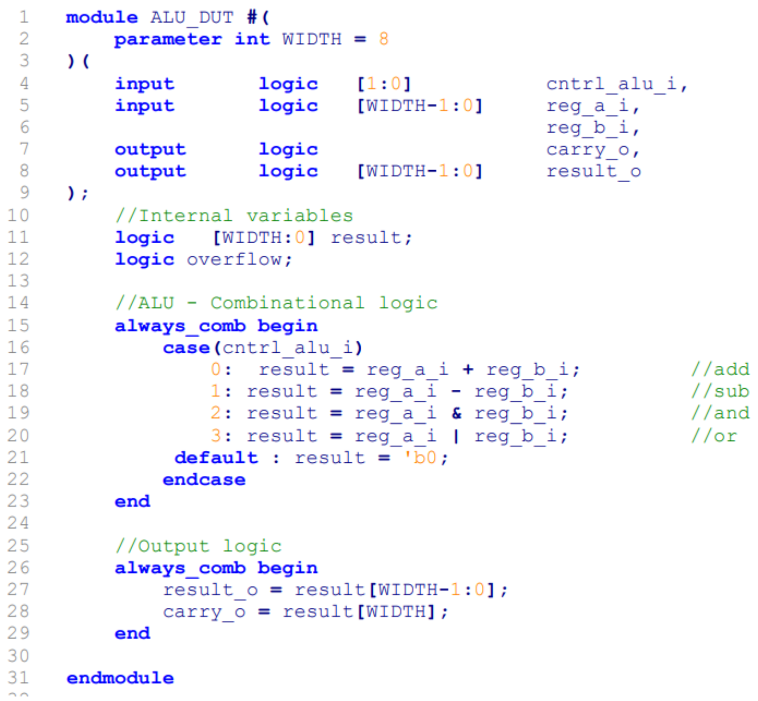


Figure 4. New file

13. In the <u>new file</u> created, **type the module** shown in Figure 5. This module corresponds to a small ALU implemented using only combinational logic (note that the module does not use a clock signal).

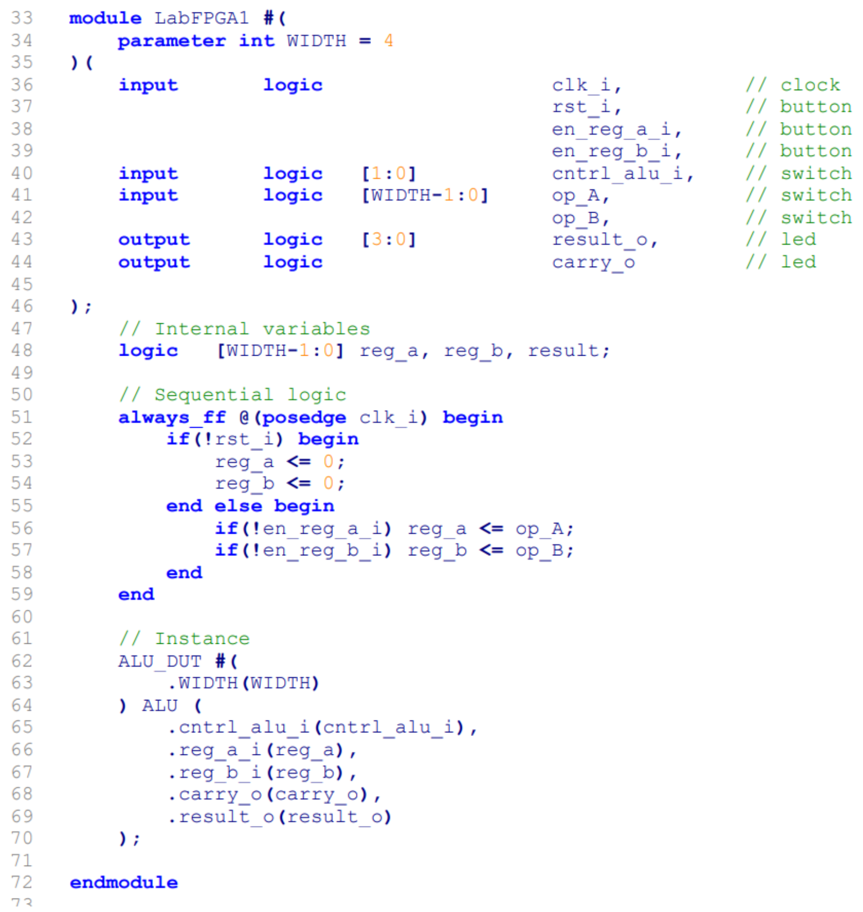


```
module ALU_DUT #(
    parameter int WIDTH = 8
)(
    input       logic    [1:0]          cntrl_alu_i,
    input       logic    [WIDTH-1:0]    reg_a_i,
                                        reg_b_i,
    output      logic                   carry_o,
    output      logic    [WIDTH-1:0]    result_o
);
    //Internal variables
    logic    [WIDTH:0] result;
    logic overflow;

    //ALU - Combinational logic
    always_comb begin
        case(cntrl_alu_i)
            0:  result = reg_a_i + reg_b_i;          //add
            1: result = reg_a_i - reg_b_i;           //sub
            2: result = reg_a_i & reg_b_i;           //and
            3: result = reg_a_i | reg_b_i;           //or
         default : result = 'b0;
        endcase
    end

    //Output logic
    always_comb begin
        result_o = result[WIDTH-1:0];
        carry_o = result[WIDTH];
    end

endmodule
```

Figure 5. Module ALU

14. **Type the module** shown in Figure 6. This is the **Top module** of the project. It contains sequential logic with two registers used to store the input data sent to the ALU, as well as the instantiation of the ALU module. In this module, the inputs are connected to the FPGA's physical peripherals.

    It is important that the **name of the Top Module matches the project name**, so the software recognizes it as the top-level entity.



```
module LabFPGA1 #(
    parameter int WIDTH = 4
)(
    input       logic                       clk_i,          // clock
                                            rst_i,          // button
                                            en_reg_a_i,     // button
                                            en_reg_b_i,     // button
    input       logic    [1:0]              cntrl_alu_i,    // switch
    input       logic    [WIDTH-1:0]        op_A,           // switch
                                            op_B,           // switch
    output      logic    [3:0]              result_o,       // led
    output      logic                       carry_o         // led

);
    // Internal variables
    logic   [WIDTH-1:0] reg_a, reg_b, result;

    // Sequential logic
    always_ff @(posedge clk_i) begin
        if(!rst_i) begin
            reg_a <= 0;
            reg_b <= 0;
        end else begin
            if(!en_reg_a_i) reg_a <= op_A;
            if(!en_reg_b_i) reg_b <= op_B;
        end
    end

    // Instance
    ALU_DUT #(
        .WIDTH(WIDTH)
    ) ALU (
        .cntrl_alu_i(cntrl_alu_i),
        .reg_a_i(reg_a),
        .reg_b_i(reg_b),
        .carry_o(carry_o),
        .result_o(result_o)
    );

endmodule
```

Figure 6. Top Module.

15. Before compiling the project, you must save the file. To do this, go to the top menu and select: **File → Save (or press Ctrl+S).** Save the file inside the project folder (LabFPGA1), and **make sure the file name matches the project name**.

Compile design

16. To compile the code, go to the top of the window and click on **Start Compilation**, as shown in Figure 7.

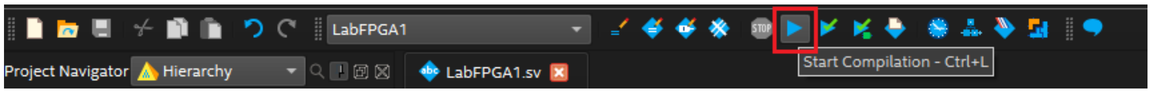


Figure 7. Start Compilation

17. The software will immediately execute various tasks automatically, such as Analysis & Synthesis, Fitter (Place & Route), etc. Wait until the compilation process is completed.

18. At the bottom of the screen, you will find the **terminal**, which displays the progress of each task. Check that no errors appear. If an error occurs, the terminal will indicate the approximate line where it happened — cross-check it with Figures 5 or 6 to identify and correct the issue.

Assign pins

19. Once the design has been compiled and synthesized successfully, you must assign the physical FPGA pins for the required peripherals. To do this, go to the top menu and select: **Assignments → Assignment Editor**, as shown in Figure 8.

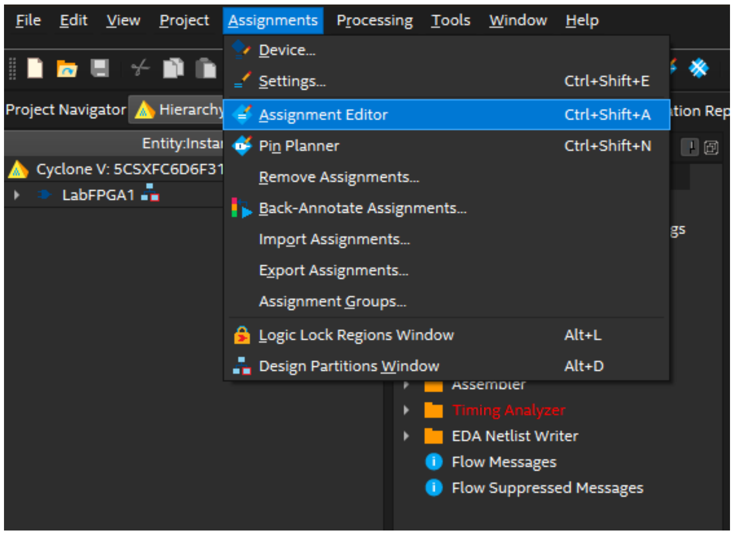


Figure 8. Assignment Editor

20. A table will appear (initially empty).

21. Click on **New** to add a new row, as shown in Figure 9.

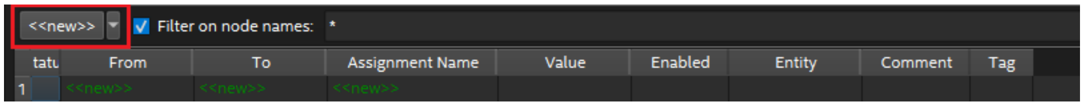


Figure 9. New Node

22. In the new row, **double-click** the cell under the "**To**" column. A small button labeled Node Finder will appear **inside** the cell, as shown in Figure 10. **Click on** it to open a new window.

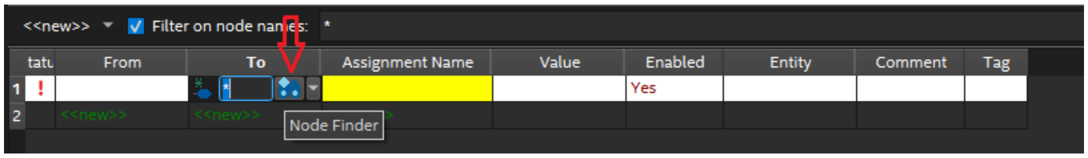
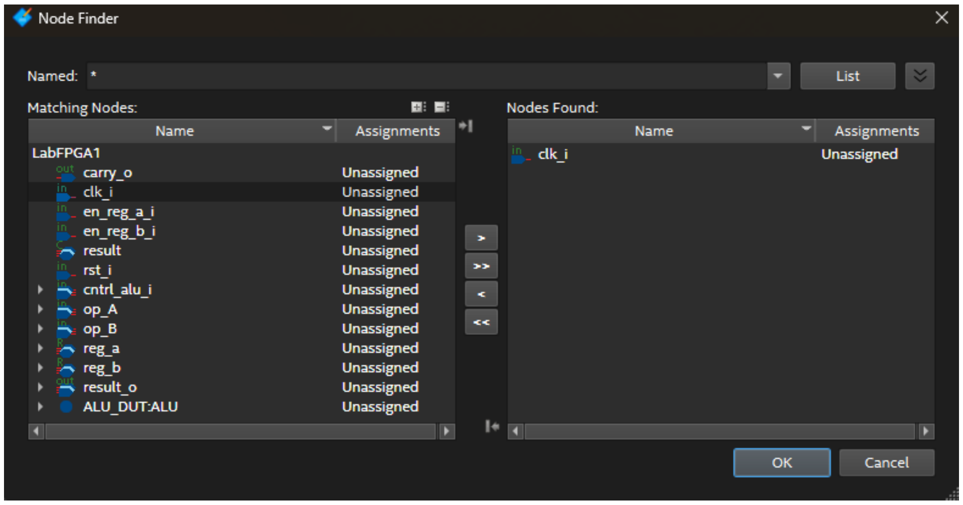


Figure 10. Access Node Finder

23. In the Node Finder window, **click List** to load all available design nodes. Select the input **clk_i**, move it to the selected nodes list, as shown in Figure 11, and **click OK**. The signal will now appear in the cell.



Matching Nodes:

Name	Assignments
LabFPGA1	
carry_o	Unassigned
clk_i	Unassigned
en_reg_a_i	Unassigned
en_reg_b_i	Unassigned
result	Unassigned
rst_i	Unassigned
cntrl_alu_i	Unassigned
op_A	Unassigned
op_B	Unassigned
reg_a	Unassigned
reg_b	Unassigned
result_o	Unassigned
ALU_DUT:ALU	Unassigned

Nodes Found:

Name	Assignments
clk_i	Unassigned

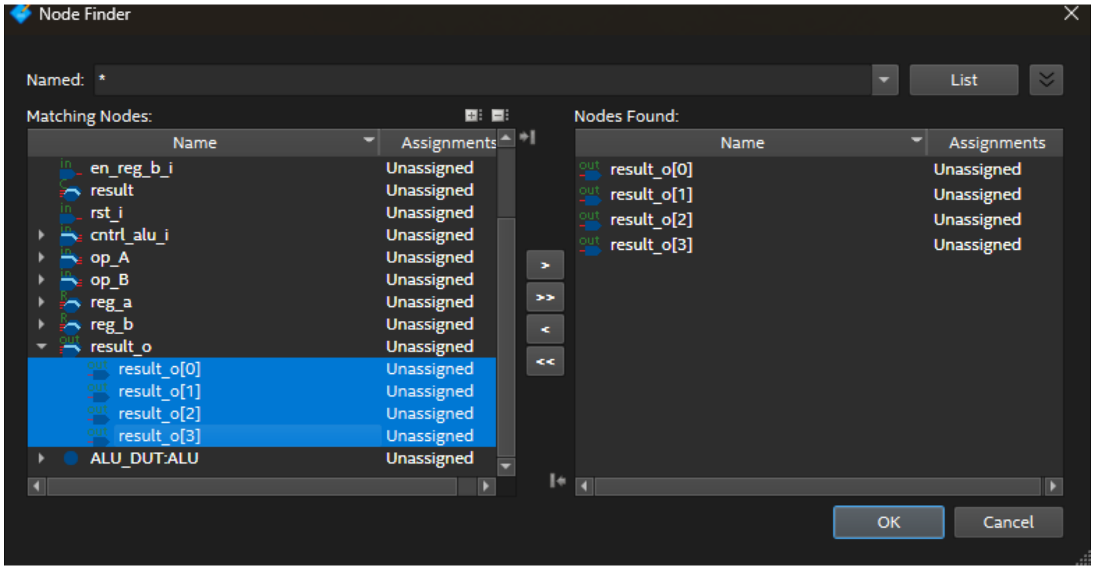
Figure 11. Node Finder

24. In the Assignment Name column, **double-click** the cell and select **Location** from the dropdown list.

25. In the Value column, enter the **physical pin location manually**. For the signal **clk_i**, type: **PIN_AF14**, which tells the software to use the 50 MHz onboard clock of the FPGA.

26. Repeat steps 21 through 25 for each input and output signal.

27. For **multi-bit** signals, each bit must be assigned to an individual physical pin. In the Node Finder window, you can expand the multi-bit variable and move each bit to the selected list. An example is shown in Figure 12.



Matching Nodes:

Name	Assignments
en_reg_b_i	Unassigned
result	Unassigned
rst_i	Unassigned
cntrl_alu_i	Unassigned
op_A	Unassigned
op_B	Unassigned
reg_a	Unassigned
reg_b	Unassigned
result_o	Unassigned
result_o[0]	Unassigned
result_o[1]	Unassigned
result_o[2]	Unassigned
result_o[3]	Unassigned
ALU_DUT:ALU	Unassigned

Nodes Found:

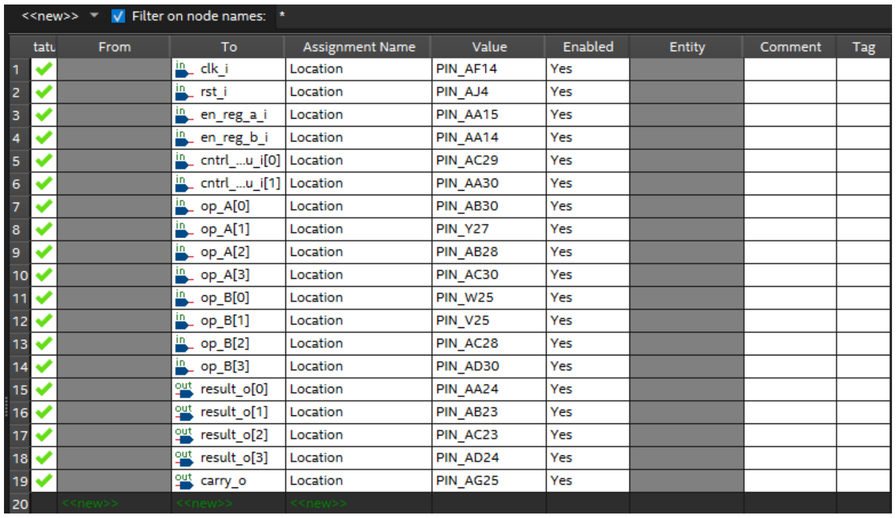
Name	Assignments
result_o[0]	Unassigned
result_o[1]	Unassigned
result_o[2]	Unassigned
result_o[3]	Unassigned

Figure 12. Assign multiple bits

28. Table 1 lists of signals with their corresponding FPGA pin assignments and connected peripherals. The expected result is shown in Figure13.

Table 1.Pin Assignment

Signal	Type	Pin	Peripheral
clk_i	Input	PIN_AF14	Clock (50 MHz)
rst_i	Input	PIN_AJ4	Key 0
en_reg_a_i	Input	PIN_AA15	Key 2
en_reg_b_i	Input	PIN_AA14	Key 3
cntrl_alu_i[0]	Input	PIN_AC29	Switch 8
cntrl_alu_i[1]	Input	PIN_AA30	Switch 9
op_A[0]	Input	PIN_AB30	Switch 0
op_A[1]	Input	PIN_Y27	Switch 1
op_A[2]	Input	PIN_AB28	Switch 2
op_A[3]	Input	PIN_AC30	Switch 3
op_B[0]	Input	PIN_W25	Switch 4
op_B[1]	Input	PIN_V25	Switch 5
op_B[2]	Input	PIN_AC28	Switch 6
op_B[3]	Input	PIN_AD30	Switch 7
result_o[0]	Output	PIN_AA24	LED 0
result_o[1]	Output	PIN_AB23	LED 1
result_o[2]	Output	PIN_AC23	LED 2
result_o[3]	Output	PIN_AD24	LED 3
carry_o	Output	PIN_AG25	LED 4


<<new>> ▾ ☑ Filter on node names: *

	tatu	From	To	Assignment Name	Value	Enabled	Entity	Comment	Tag
1	✓		clk_i	Location	PIN_AF14	Yes			
2	✓		rst_i	Location	PIN_AJ4	Yes			
3	✓		en_reg_a_i	Location	PIN_AA15	Yes			
4	✓		en_reg_b_i	Location	PIN_AA14	Yes			
5	✓		cntrl_...u_i[0]	Location	PIN_AC29	Yes			
6	✓		cntrl_...u_i[1]	Location	PIN_AA30	Yes			
7	✓		op_A[0]	Location	PIN_AB30	Yes			
8	✓		op_A[1]	Location	PIN_Y27	Yes			
9	✓		op_A[2]	Location	PIN_AB28	Yes			
10	✓		op_A[3]	Location	PIN_AC30	Yes			
11	✓		op_B[0]	Location	PIN_W25	Yes			
12	✓		op_B[1]	Location	PIN_V25	Yes			
13	✓		op_B[2]	Location	PIN_AC28	Yes			
14	✓		op_B[3]	Location	PIN_AD30	Yes			
15	✓		result_o[0]	Location	PIN_AA24	Yes			
16	✓		result_o[1]	Location	PIN_AB23	Yes			
17	✓		result_o[2]	Location	PIN_AC23	Yes			
18	✓		result_o[3]	Location	PIN_AD24	Yes			
19	✓		carry_o	Location	PIN_AG25	Yes			
20		<<new>>	<<new>>	<<new>>					

Figure 13. Pin Assignment Result

29. You must **recompile** the design as shown in **Step 16**. A pop-up window will appear asking to save the changes made in the assignments — **click Yes** to confirm.

Programming FPGA

30. Finally, the only step left is to load the design onto the FPGA. To do this, connect the **DE10-Standard** board to the power supply and the USB-Blaster cable to a USB port on your computer, then turn on the FPGA using the **red power button**.
31. In the top menu, select **Tools → Programmer**, as shown in Figure 14

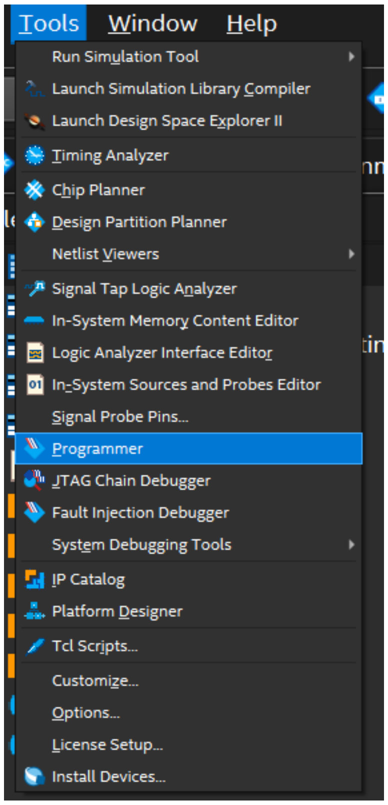
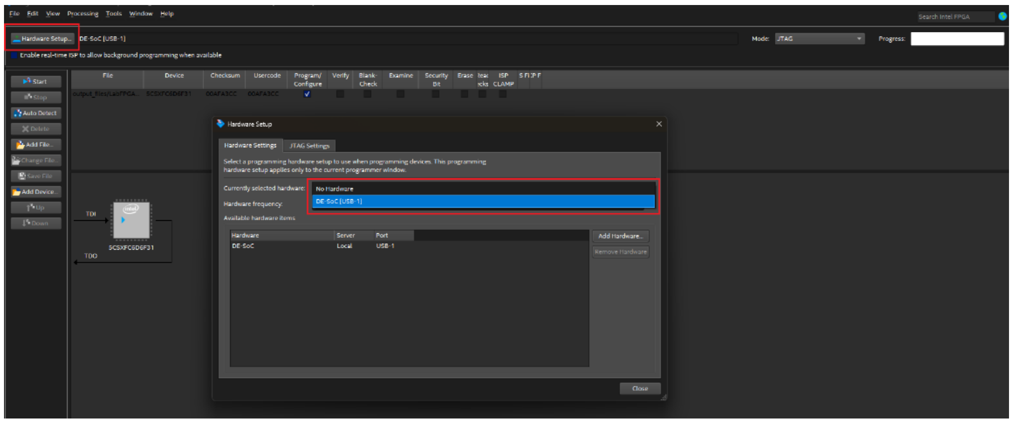


Figure 14. Access to Programmer

32. Once the Programmer window opens, **click** on **Hardware Setup** in the upper-left corner. A new window will appear — under <u>Currently selected hardware</u>, choose the FPGA connected to your computer, as shown in Figure 15. Then click Close.



Hardware Setup... DE-SoC [USB-1]   Mode: JTAG   Progress:

Enable real-time ISP to allow background programming when available

File	Device	Checksum	Usercode	Program/ Configure	Verify	Blank-Check	Examine	Security Bit	Erase
output_files/LabFPGA...	5CSXFC6D6F31	00AFA3CC	00AFA3CC	✓					

Hardware Setup

Hardware Settings | JTAG Settings

Select a programming hardware setup to use when programming devices. This programming hardware setup applies only to the current programmer window.

Currently selected hardware: No Hardware / DE-SoC [USB-1]

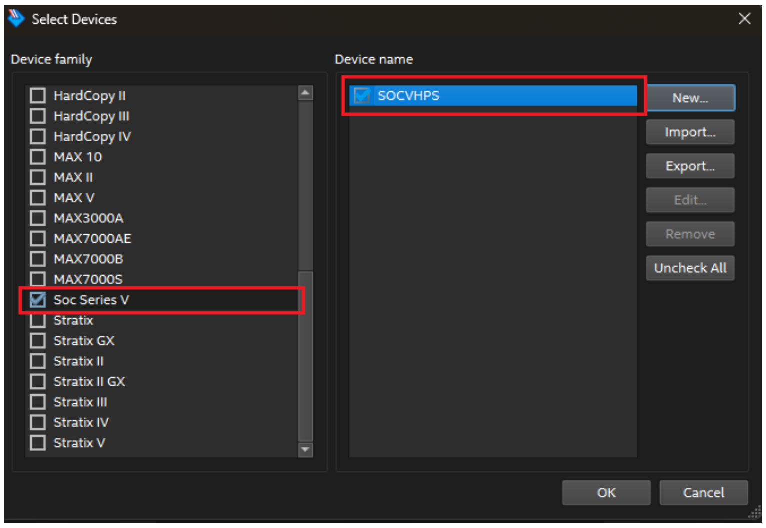
Hardware frequency:

Available hardware items

Hardware	Server	Port
DE-SoC	Local	USB-1

Figure 15. Select Hardware

33. On the left side of the *Programmer* window, click on ***Add Device***. A new window with two columns will appear. In the first column, look for ***SoC Series V*** and in the second column, select ***SOCVHPS***, as shown in Figure 16. Finally, click *OK*.



Select Devices

Device family

☐ HardCopy II
☐ HardCopy III
☐ HardCopy IV
☐ MAX 10
☐ MAX II
☐ MAX V
☐ MAX3000A
☐ MAX7000AE
☐ MAX7000B
☐ MAX7000S
☑ Soc Series V
☐ Stratix
☐ Stratix GX
☐ Stratix II
☐ Stratix II GX
☐ Stratix III
☐ Stratix IV
☐ Stratix V

Device name

☑ SOCVHPS

Figure 16. Add SOCVHPS

34. The new device will appear to the right of the FPGA device in the chain. **Click and drag SOCVHPS** to the **left of the FPGA** in the chain, so TDI connects to SOCVHPS, as shown in Figure 17.

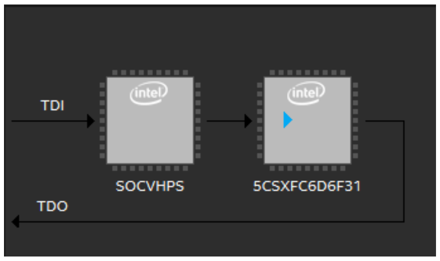
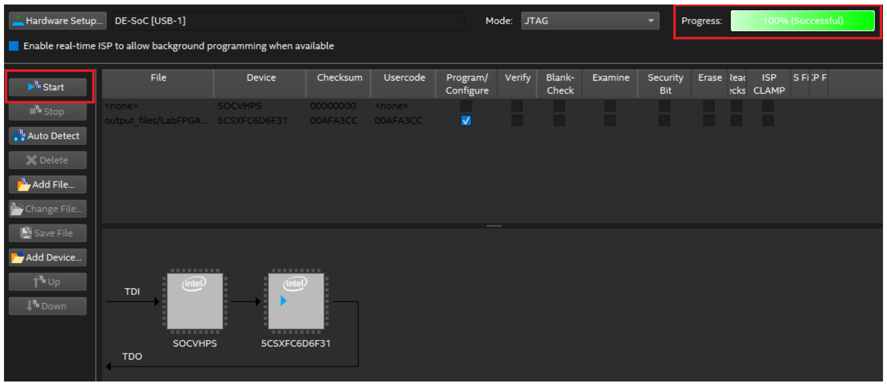


Figure 17. SOCVHPS

35. Finally, click **Start** to load the design onto the FPGA. Make sure the progress bar reaches 100% and displays Successful, as shown in Figure 18.



Hardware Setup... DE-SoC [USB-1] Mode: JTAG Progress: 100% (Successful)

☑ Enable real-time ISP to allow background programming when available

File	Device	Checksum	Usercode	Program/ Configure	Verify	Blank-Check	Examine	Security Bit	Erase	Read ecks	ISP CLAMP
<none>	SOCVHPS	00000000	<none>	☐	☐	☐	☐	☐	☐	☐	☐
output_files/LabFPGA...	5CSXFC6D6F31	00AFA3CC	00AFA3CC	☑	☐	☐	☐	☐	☐	☐	☐

Figure 18. Start programmer

Peripherals

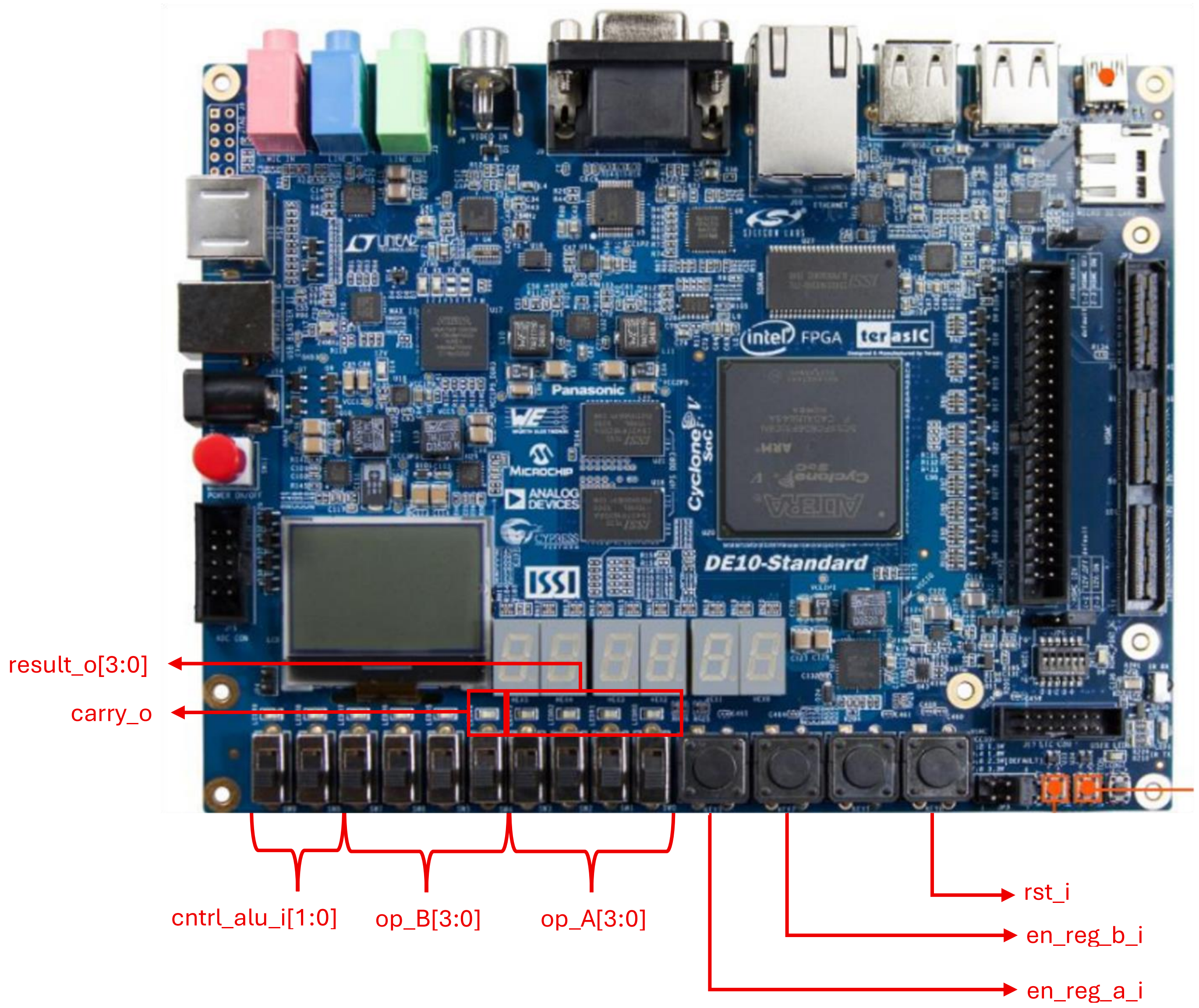


Table 2. Description of peripherals.

Signal	**Type**	**Function Description**
clk_i	Input	System clock. All synchronous logic is triggered on the rising edge.
rst_i	Input	Resets the internal registers to their known default values.
en_reg_a_i	Input	Captures the values from the op_A switches and stores them into register A.
en_reg_b_i	Input	Captures the values from the op_B switches and stores them into register B.
cntrl_alu_i	Input	Selects which ALU operation to execute.
op_A	Input	Sets the operand A values via physical switches.
op_B	Input	Sets the operand B values via physical switches.
result_o	Output	Displays the output bits of the ALU result.
carry_o	Output	Indicates if there was a carry-out from the ALU operation.