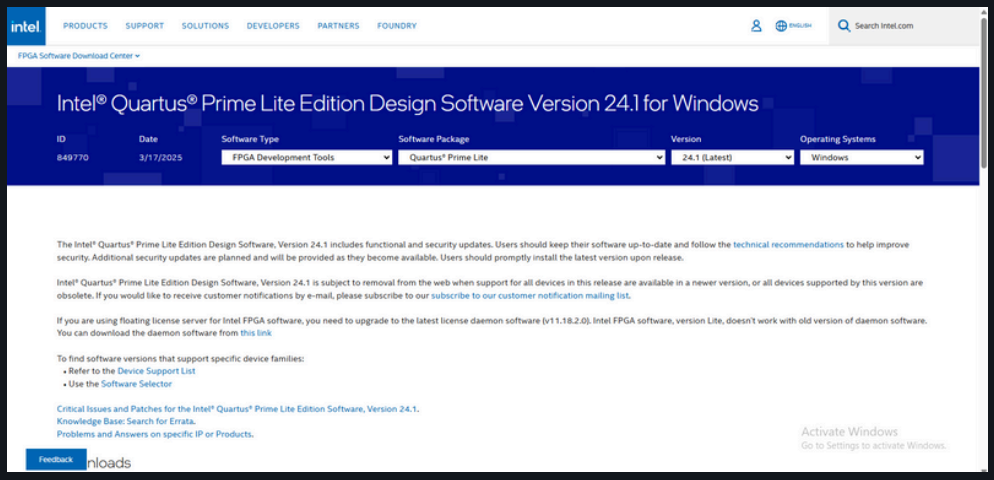
FPGA Setup and Programming using Quartus

Edition License Needed Supported Devices Notes

Edition	Device Support	Description	License
Lite	❌ No	Simple boards (e.g., MAX 10)	Free to use
Standard	✅ Yes	Simple + mid-range devices	Requires paid license
Pro	✅ Yes	High-end, complex devices	For large-scale, pro work

Downloaded Lite because it's a non-licensed software and is sufficient for low-end FPGAs (here MAX 10 and Cyclone V)

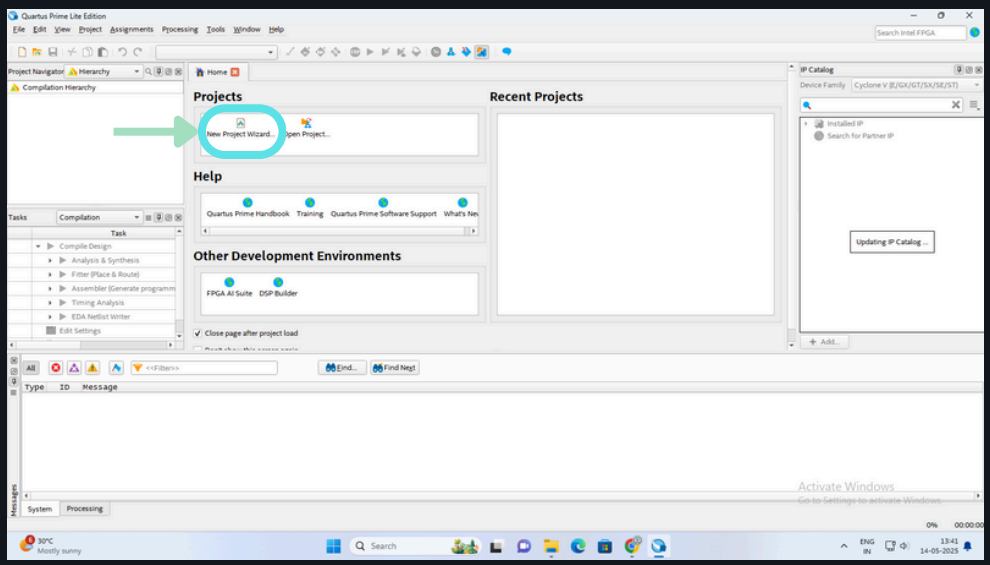


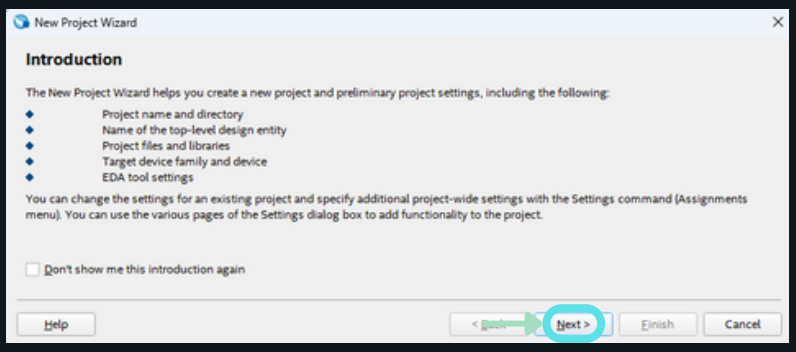
For Windows:
- If you downloaded a .zip file, simply right-click on it and choose "Extract All..." to unzip the contents.
- If it's an .exe installer, just double-click to start the installation process.

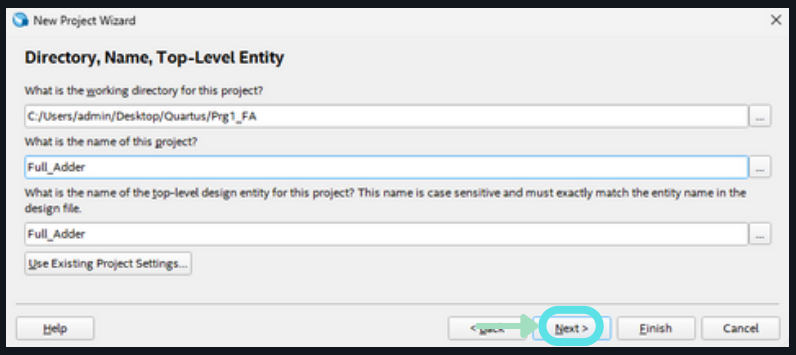
Creating a New Project in Quartus

Launch Intel Quartus Prime Lite.

Go to File → New Project Wizard.



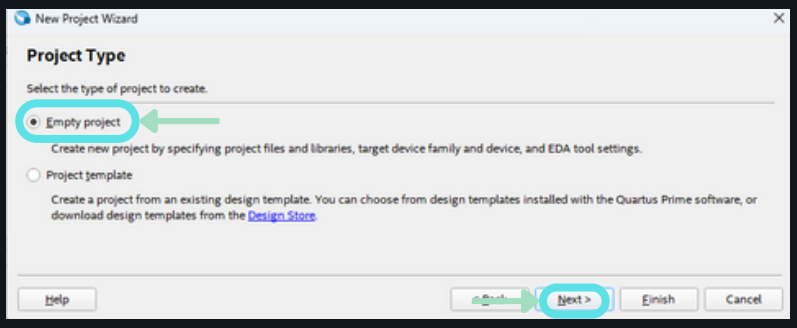




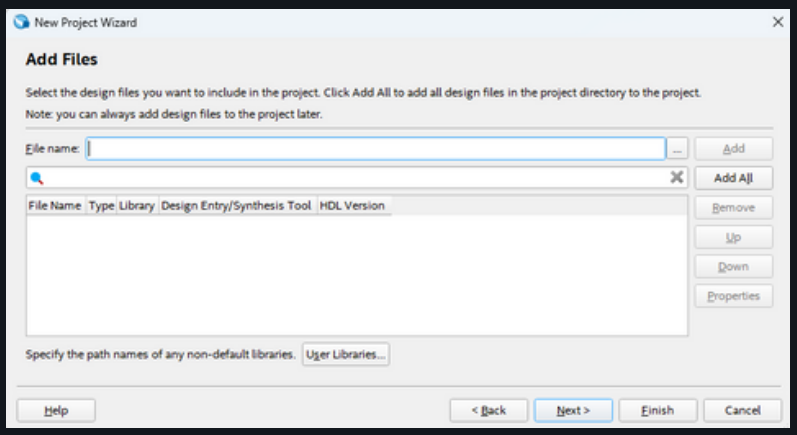
Click Next.

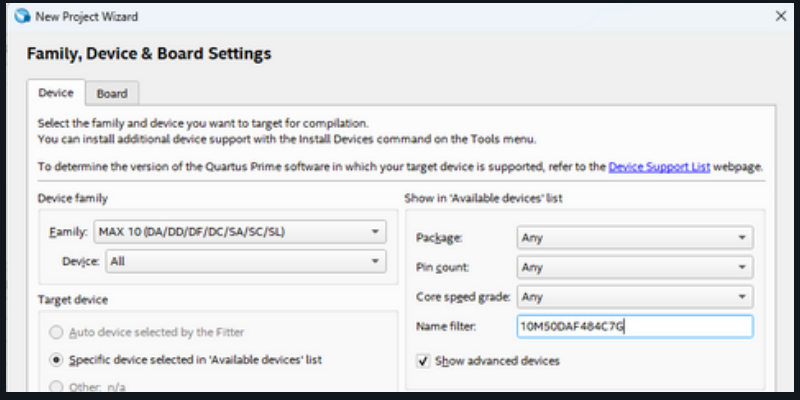
- Fill These Important Fields:
- Project Name: Choose a name (e.g., led_blink).
- Project Directory: Choose folder location.
- Top-Level Entity Name: Must match your Verilog module name (e.g., led_blink).
  - Doesn't require it to be the same or different to the Project Name
- Click Next.

Select Empty Project, then click Next.



Skip file adding for now → click Next.

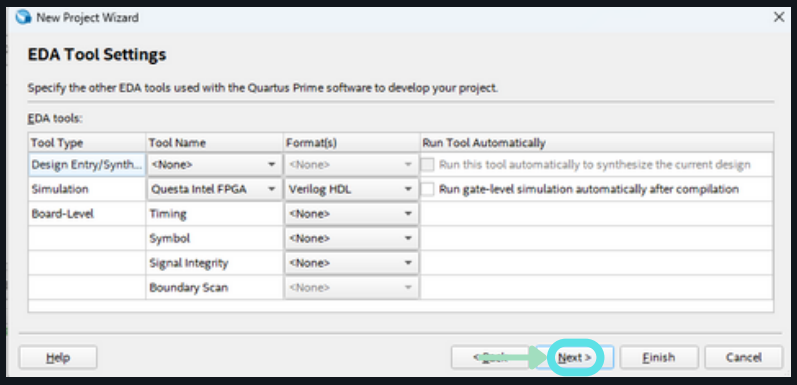




Choose:

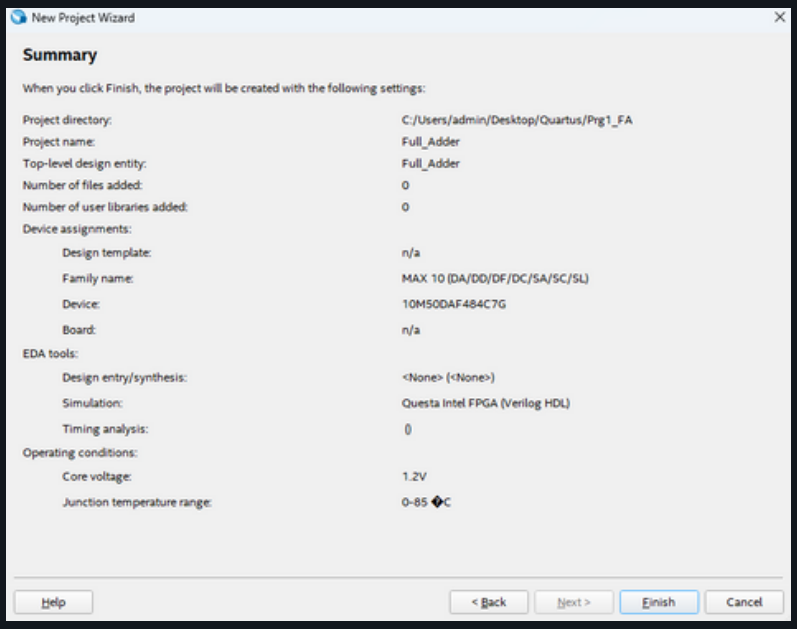
- Family: MAX 10
- Name Filter: 10M50DAF484C7G

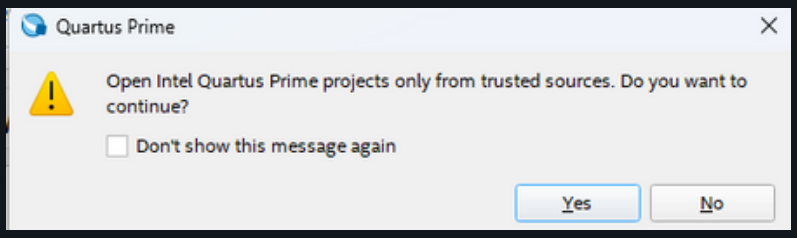
Select device and click Next.



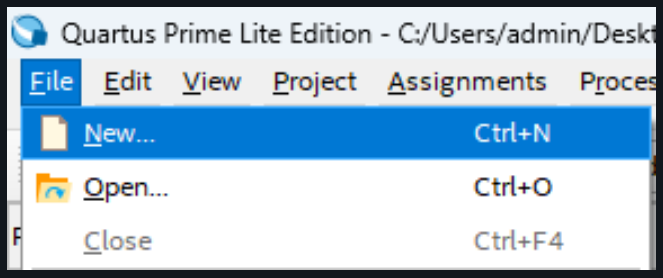
Leave everything as default.
Click Next.

Review the summary.
Click Finish.

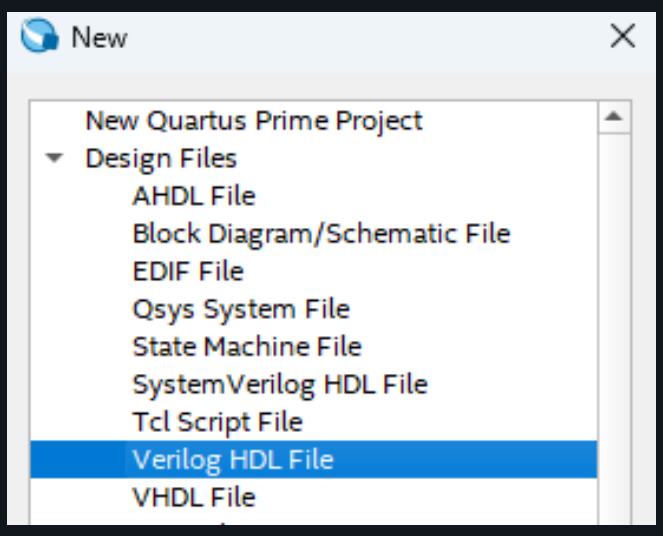




Pop-up: Click Yes



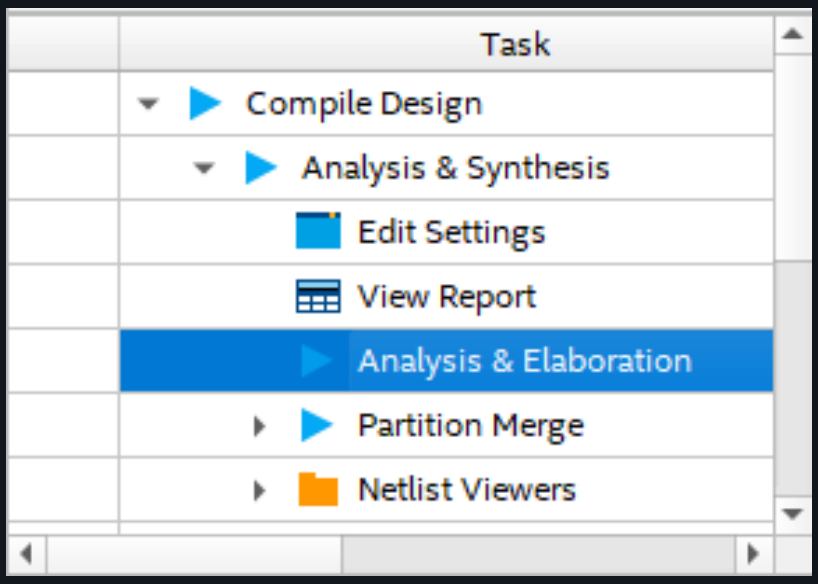
Create Verilog HDL File
Go to File → New → Verilog HDL File



Write your Verilog code inside.

Save it with the same name as your top-level entity.

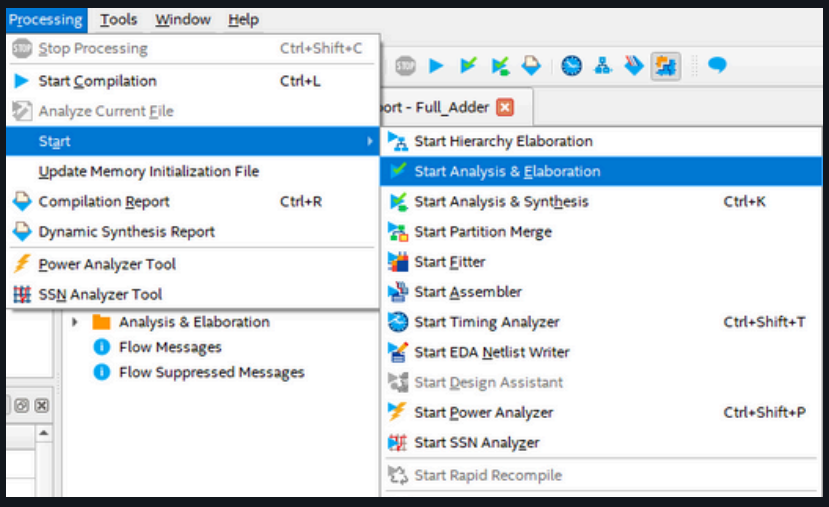
```
module Full_Adder

(input A,B, Cin , output S,Cout);

assign S=A ^ B ^ Cin ,
       Cout=(A & B)|( B & Cin)|(Cin & A);

endmodule
```

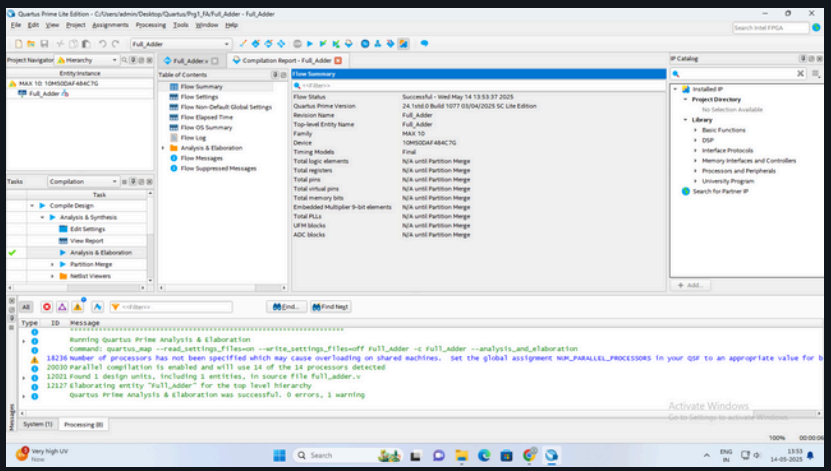


Analysis & Elaboration

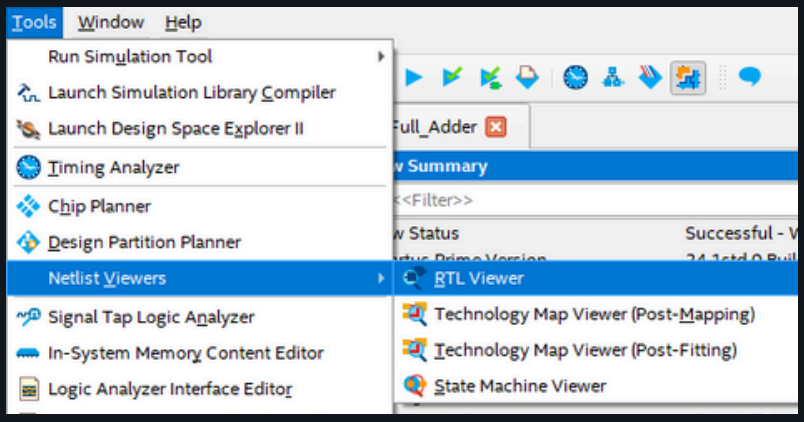
Right-click → Start or Double Click

Or go to Processing → Start → Start Analysis & Elaboration



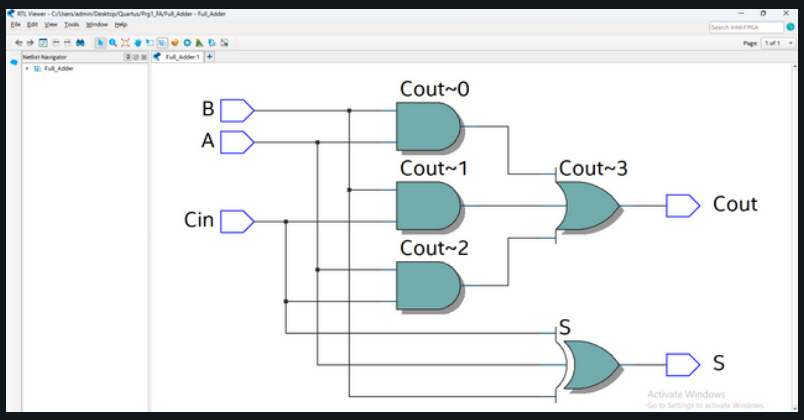


Summary will show → You may review or skip.



Go to Tools → Netlist Viewers → RTL Viewer

You'll see a logic-level block diagram of your design.



Reference your FPGA board's datasheet to set:
Input/Output pins (e.g., LEDs, switches)
Voltage standards

Find the Datasheet:

- Search for MAX 10 10M50DAF484C7G datasheet on the official Intel FPGA website or authorized distributors.

Identify I/O Pins:

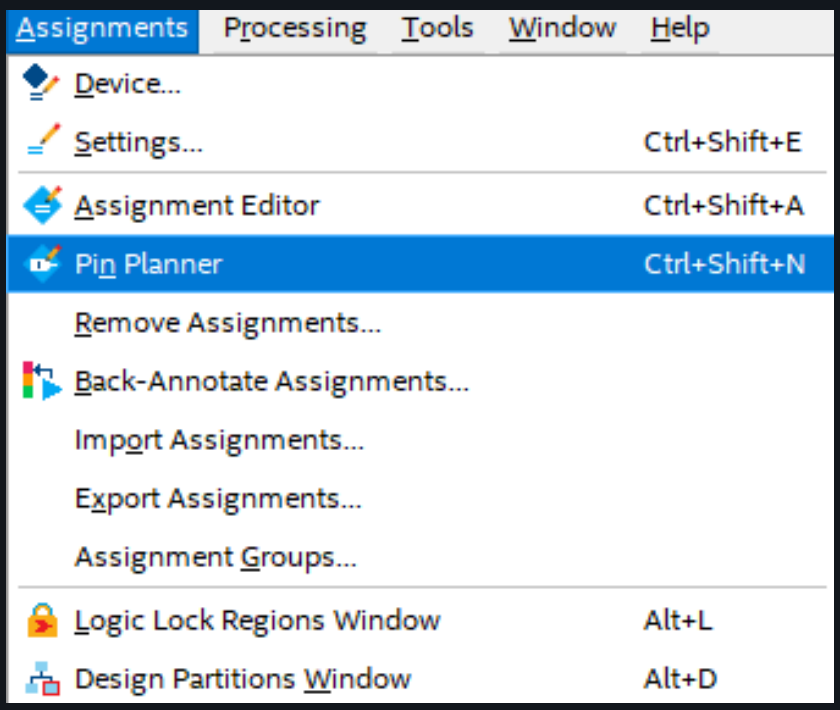
- In the datasheet, go to the Pinout or Pin Assignments section to find I/O pin configurations (e.g., LEDs, switches).

Check Voltage Standards:

- Locate the Voltage Standards or I/O Voltage Characteristics section to determine supported voltage levels (e.g., LVCMOS, LVTTL, LVDS).

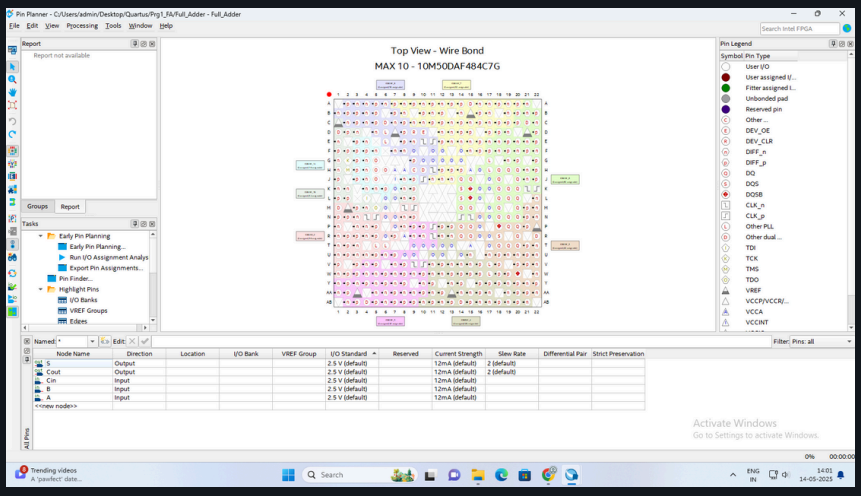
Configure in Design:

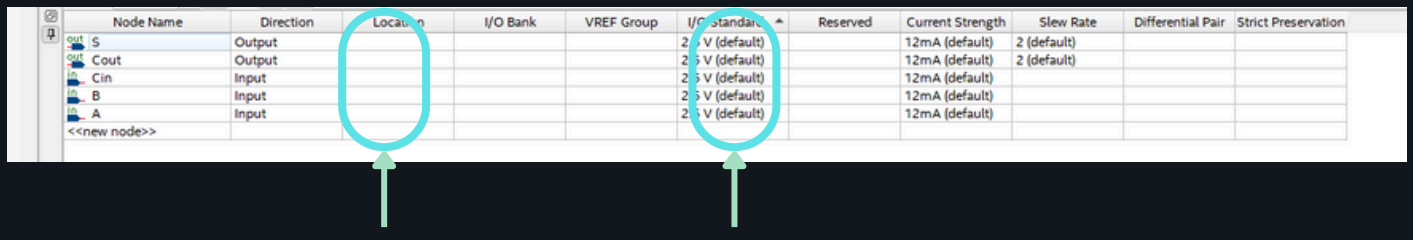
- Use the pinout and voltage info to set I/O pins and voltage standards in your FPGA design software (e.g., Intel Quartus).



Go to Assignments
→ Pin Planner

Assign Pins
(Pin Planner)





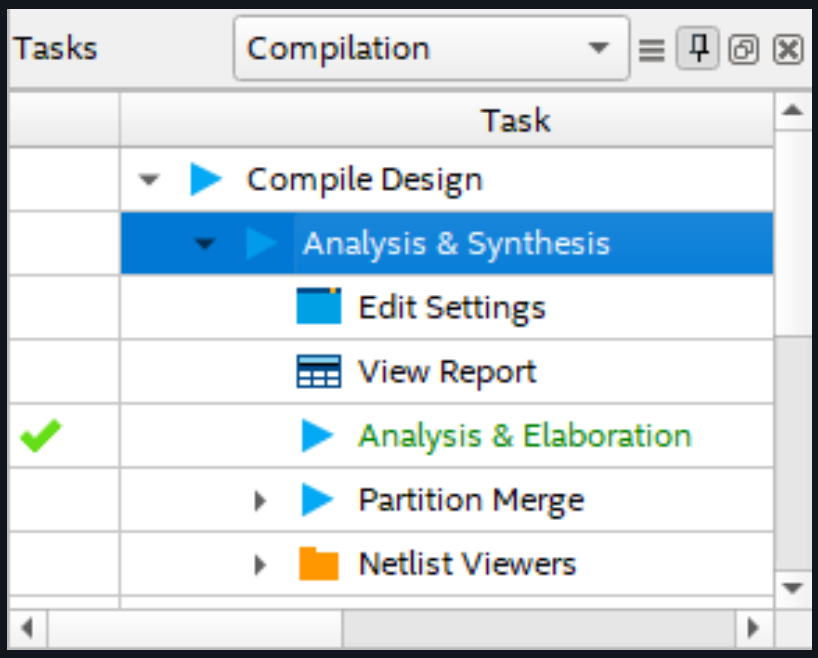
Reference the FPGA board's datasheet to configure I/O pins (e.g., LEDs, switches) and set appropriate voltage standards.

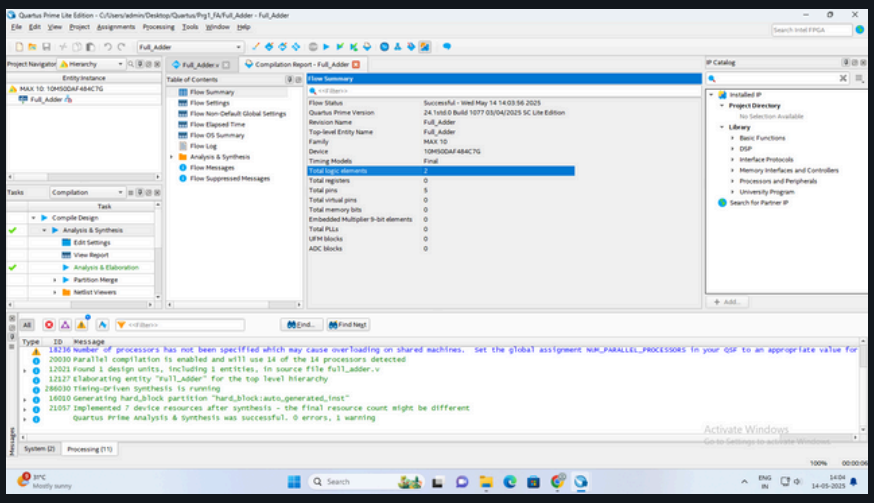
Analysis & Synthesis

Right-click → Start
or Double Click

Or

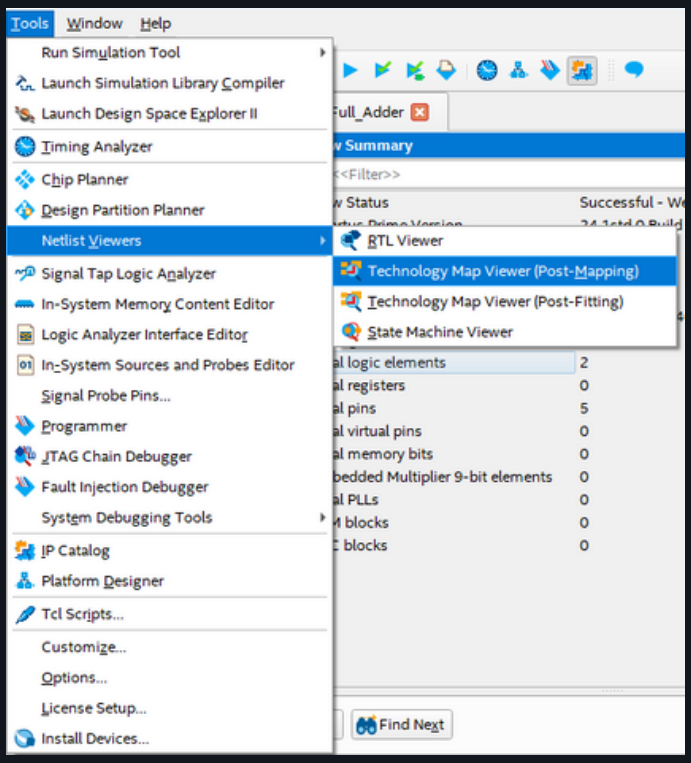
go to
Processing
→ Start
→ Start Analysis & Synthesis

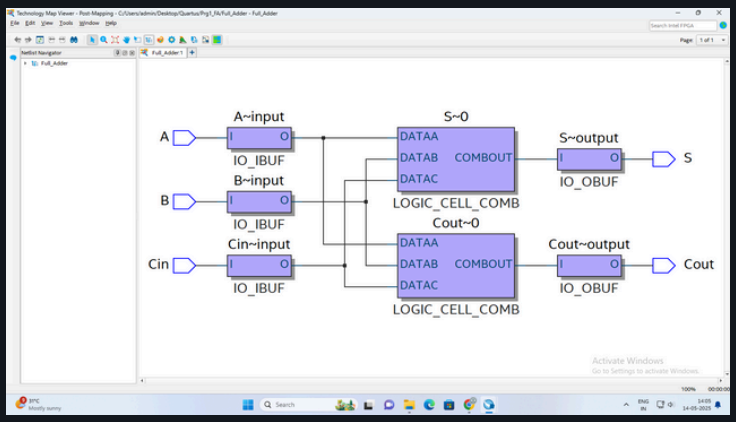




Summary will show →
You may review or skip.

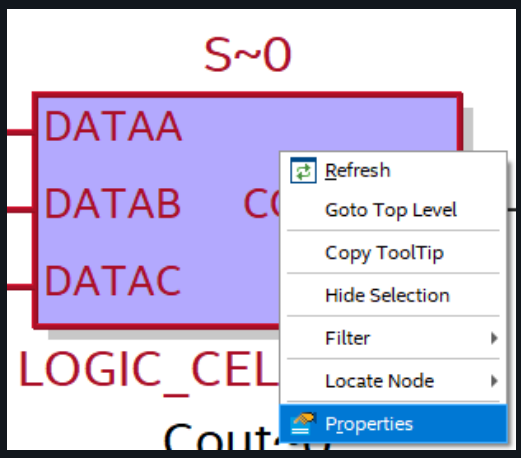
Go to Tools
→ Netlist Viewers
→ Technology Map Viewer
(post-mapping)

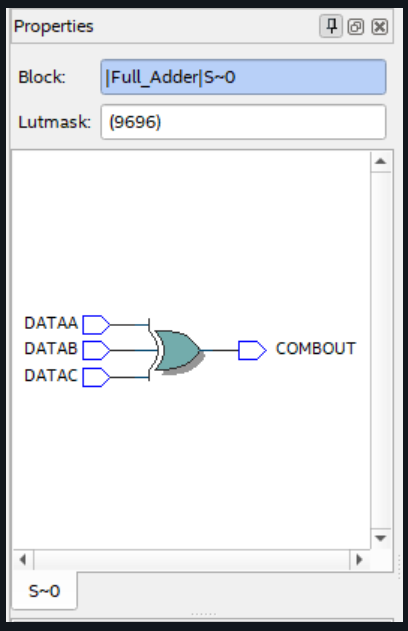


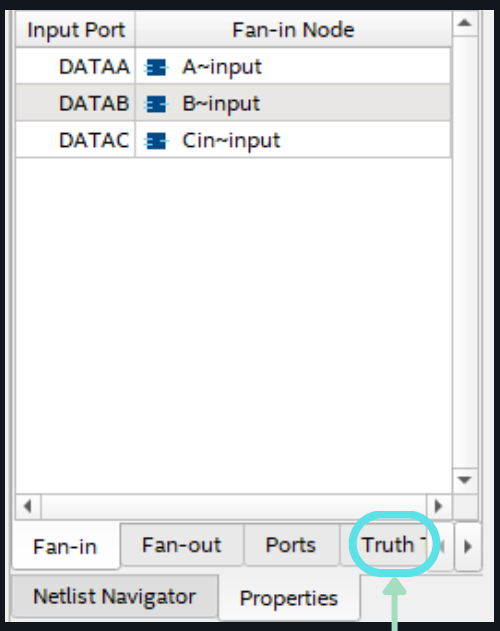


You'll see the post-mapped logic block diagram in the FPGA design software, which shows the arrangement and connections of logic blocks after mapping the design to the FPGA's resources.

Right Click on boxes
→ Click Properties



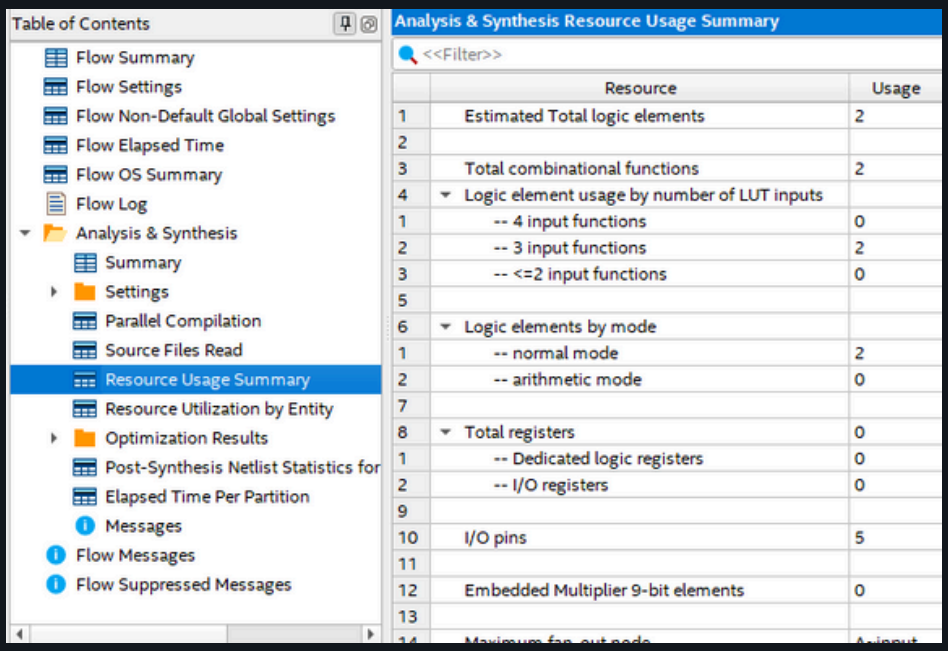




Inspect the logic and truth table to verify the functionality,

or skip the process if you trust the design is correct and sufficient verification has been done.

Click to open the Truth Table or Look-Up Table (LUT)



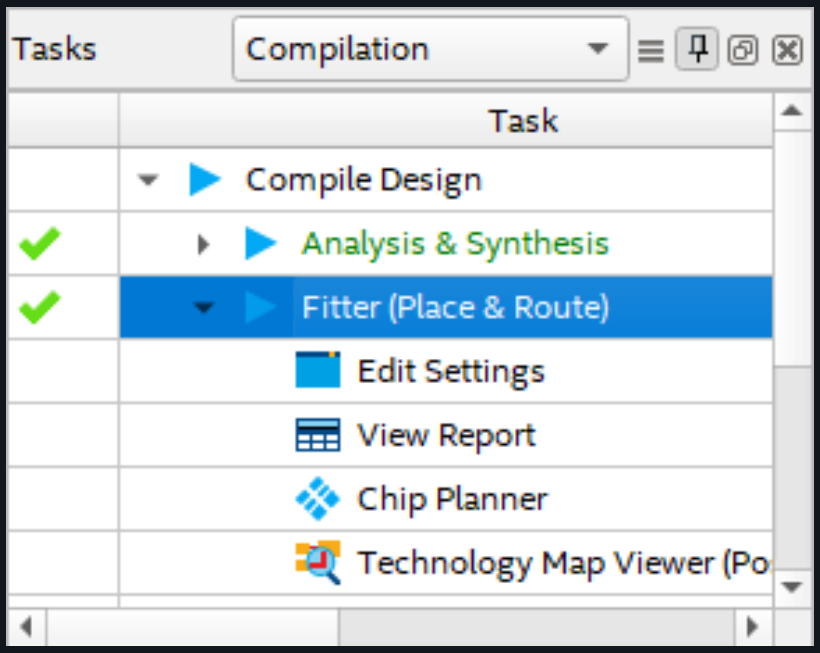
Go to Table of Contents
→ Resource Usage Summary for logic utilization.

Fitter

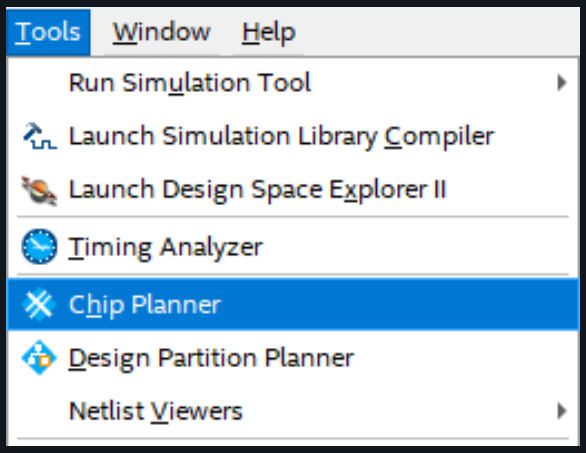
Right-click → Start
or Double Click

Or

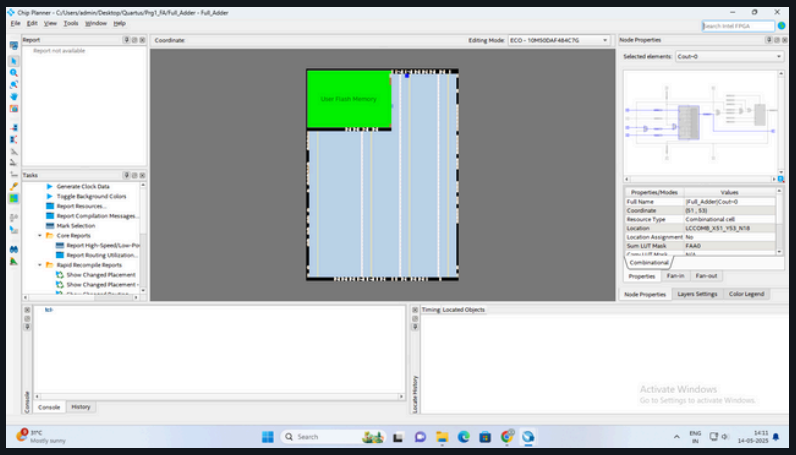
go to
Processing
→ Start
→ Start Analysis & Synthesis



Summary will show → You may review or skip.

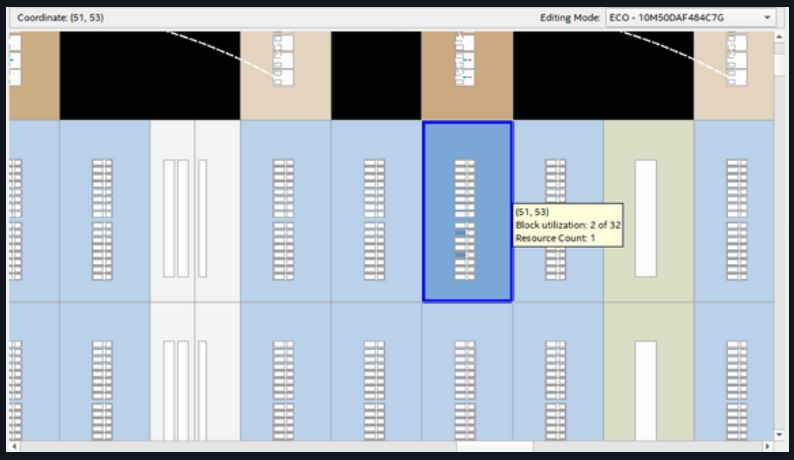


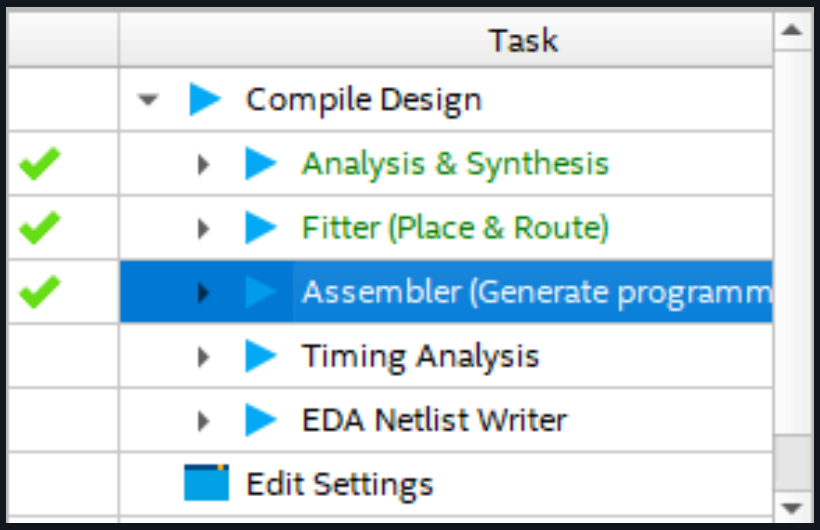
Go to Tools → Chip Planner



You can see the logic mapped on the FPGA.

To zoom in or out,
hold Ctrl and scroll with your mouse.





Assembler

Right-click → Start
or Double Click

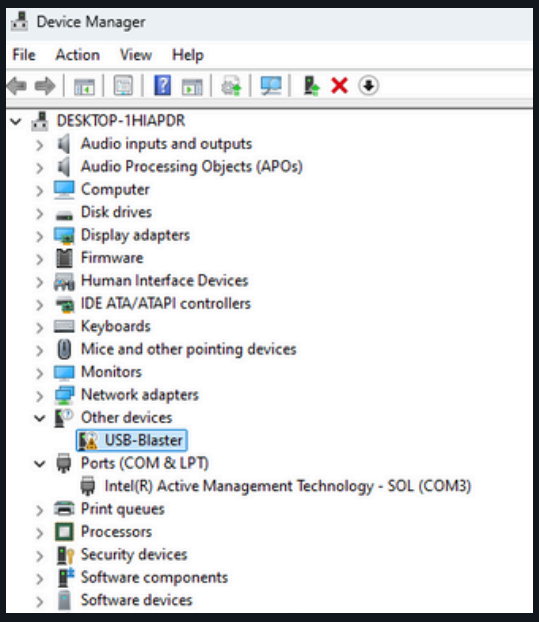
Or

go to
Processing
→ Start
→ Start Assembler

> This generates the .sof file for programming the board.

Summary will show → You may review or skip.

Connect your FPGA to the PC using a suitable programming cable (e.g., USB-Blaster for Intel FPGAs).
Ensure the FPGA is powered and the connection is established properly for programming.



Setup USB Blaster Driver
(First Time Only)

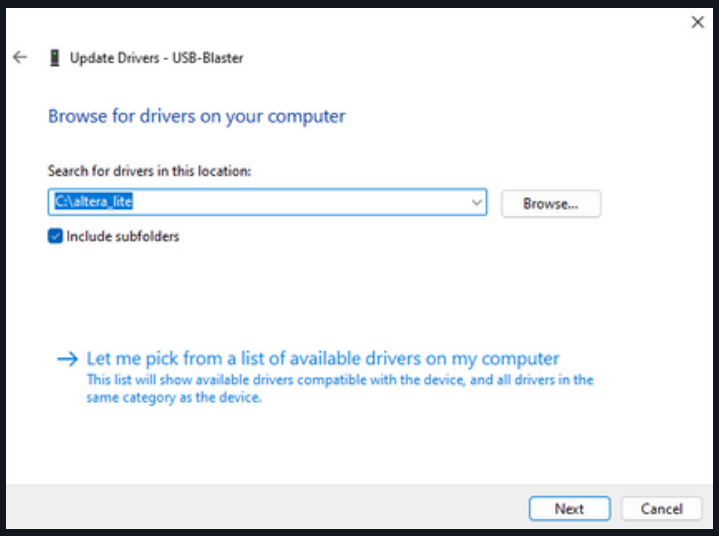
Open Device Manager
Look for Other Devices
→ USB - Blaster
Right-click → Update Driver

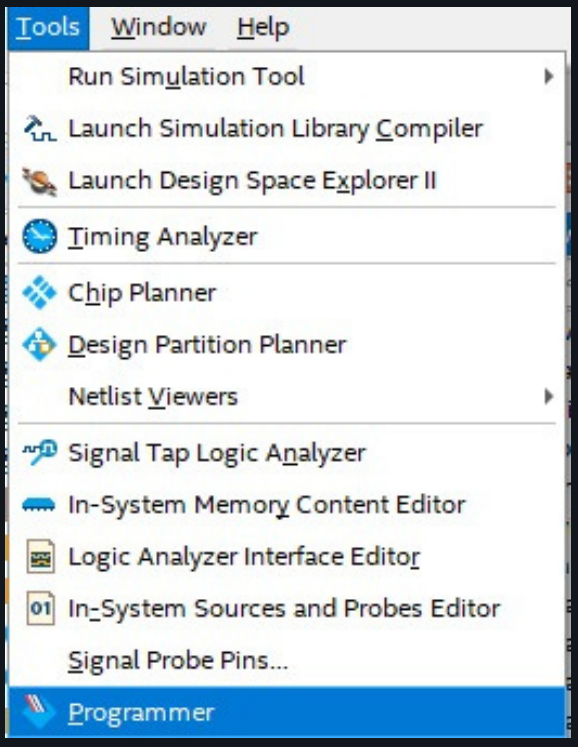
Browse to:

C:\intelFPGA_lite...\drivers\usb-blaster (where Quartus is installed)

Choose altera_lite

Click Next and
Install driver successfully.

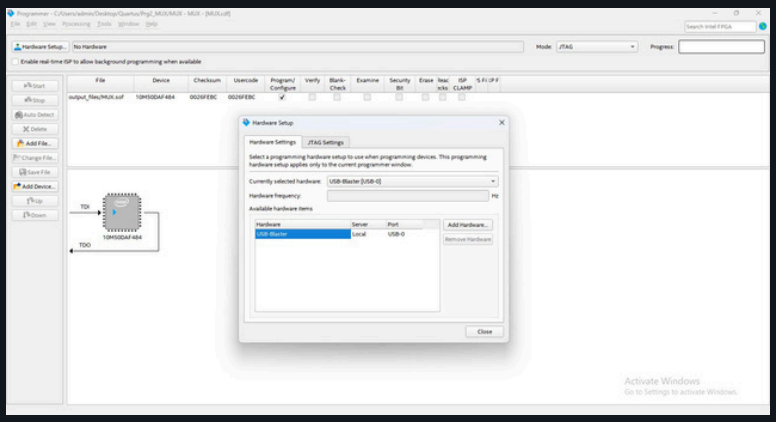




Go to Tools → Programmer

Hardware Setup → Select USB-Blaster

Click Start to dump program to your board



The .sof file is auto-detected and added

Verify the output (e.g., LED blinking)

This is a Full Adder module. It takes in three inputs: A, B, and Cin (carry-in), and produces two outputs: S (sum) and Cout (carry-out).

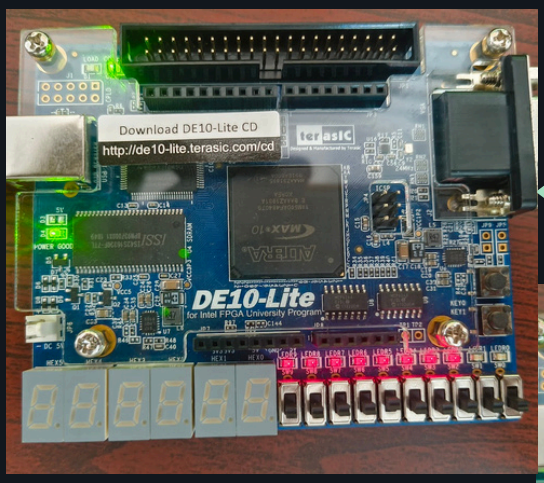
```
// Full Adder Module
// This module implements a 1-bit Full Adder, which takes in two input bits (A and B)
// and a carry-in (Cin), producing a sum (S) and a carry-out (Cout).

module Full_Adder (
    input A,      // Input A (1-bit)
    input B,      // Input B (1-bit)
    input Cin,    // Carry-in (1-bit)
    output S,     // Sum output (1-bit)
    output Cout   // Carry-out output (1-bit)
);

// Sum is the result of the XOR operation between A, B, and Cin.
assign S = A ^ B ^ Cin;

// Carry-out is generated by checking if any two of the inputs are 1.
assign Cout = (A & B) | (B & Cin) | (Cin & A);

endmodule
```

Inputs			Outputs	
A	B	C_{in}	S	C_{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

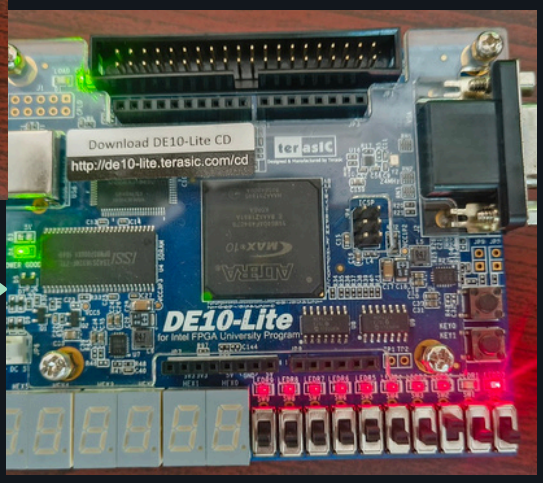
Use a truth table to verify the output in FPGA. In the FPGA setup,

The connections are as follows:

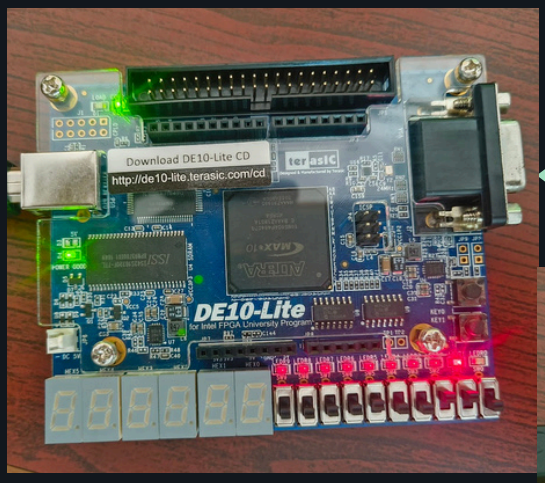
- The last two LEDs represent SUM and COUT, respectively.
- The last three switches represent A, B, and Cin inputs.



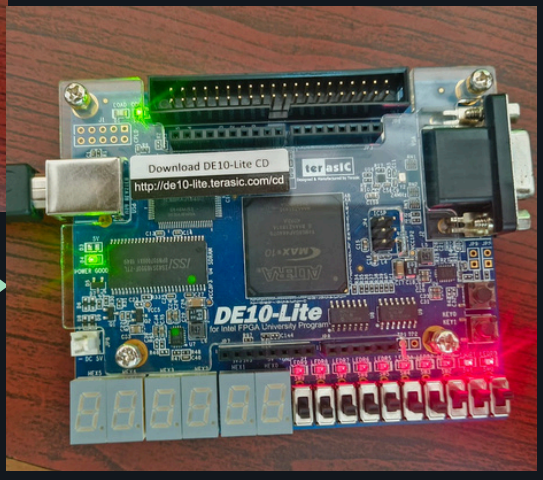
For the input combination
A = 0, B = 0, and Cin = 0,
the outputs are
COUT = 0 and SUM = 0.



For the input combination
A = 1, B = 0, and Cin = 0,
the outputs are
COUT = 0 and SUM = 1.



For the input combination
A = 1, B = 1, and Cin = 0,
the outputs are
COUT = 1 and SUM = 0.

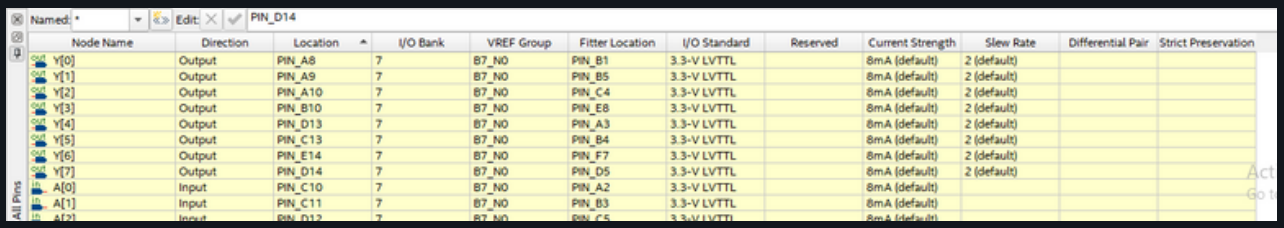


For the input combination
A = 1, B = 1, and Cin = 1,
the outputs are
COUT = 1 and SUM = 1.

This Verilog module represents a 2-to-1 Multiplexer (MUX), which selects one of two inputs based on the value of the selector signal.

```
// Multiplexer (MUX) Module
// This module implements a 2-to-1 multiplexer, which selects between two inputs (A and B)
// based on the value of the selector signal (S). If S is 1, the output Y is B;
// if S is 0, the output Y is A.

module MUX (
    input A,      // Input A (1-bit)
    input B,      // Input B (1-bit)
    input S,      // Selector signal (1-bit)
    output Y      // Output Y (1-bit)
);

// The output Y is determined based on the selector signal S.
// If S is 1, Y = B; if S is 0, Y = A.
assign Y = S ? B : A;

endmodule
```

Reference the FPGA board's datasheet to configure I/O pins (e.g., LEDs, switches) and set appropriate voltage standards.



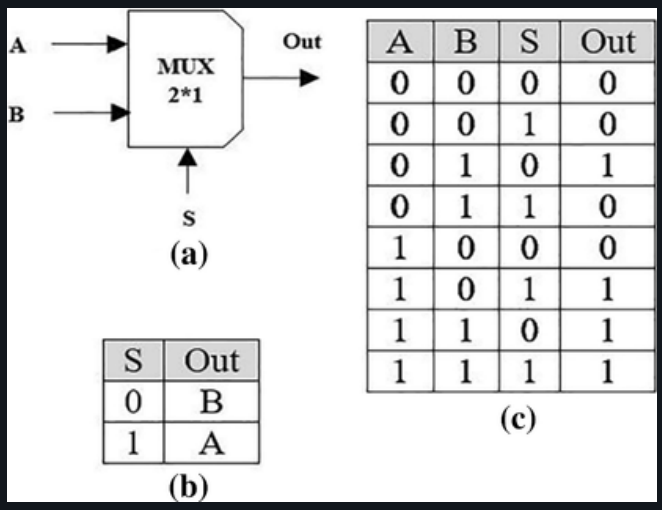
Node Name	Direction	Location	I/O Bank	VREF Group	Fitter Location	I/O Standard	Reserved	Current Strength	Slew Rate	Differential Pair	Strict Preservation
Y[0]	Output	PIN_A8	7	B7_N0	PIN_B1	3.3-V LVTTL		8mA (default)	2 (default)		
Y[1]	Output	PIN_A9	7	B7_N0	PIN_B5	3.3-V LVTTL		8mA (default)	2 (default)		
Y[2]	Output	PIN_A10	7	B7_N0	PIN_C4	3.3-V LVTTL		8mA (default)	2 (default)		
Y[3]	Output	PIN_B10	7	B7_N0	PIN_E8	3.3-V LVTTL		8mA (default)	2 (default)		
Y[4]	Output	PIN_D13	7	B7_N0	PIN_A3	3.3-V LVTTL		8mA (default)	2 (default)		
Y[5]	Output	PIN_C13	7	B7_N0	PIN_B4	3.3-V LVTTL		8mA (default)	2 (default)		
Y[6]	Output	PIN_E14	7	B7_N0	PIN_F7	3.3-V LVTTL		8mA (default)	2 (default)		
Y[7]	Output	PIN_D14	7	B7_N0	PIN_D5	3.3-V LVTTL		8mA (default)	2 (default)		
A[0]	Input	PIN_C10	7	B7_N0	PIN_A2	3.3-V LVTTL		8mA (default)			
A[1]	Input	PIN_C11	7	B7_N0	PIN_B3	3.3-V LVTTL		8mA (default)			

A 2-to-1 multiplexer is a digital switch that selects one of two inputs based on a control signal (selector S). It has:

Two Inputs:
A: The first input (1-bit).
B: The second input (1-bit).

One Selector (Control) Signal:
S: A single control signal (1-bit) that determines which input is passed to the output.

Y: The output (1-bit), which will be either A or B based on the value of S.

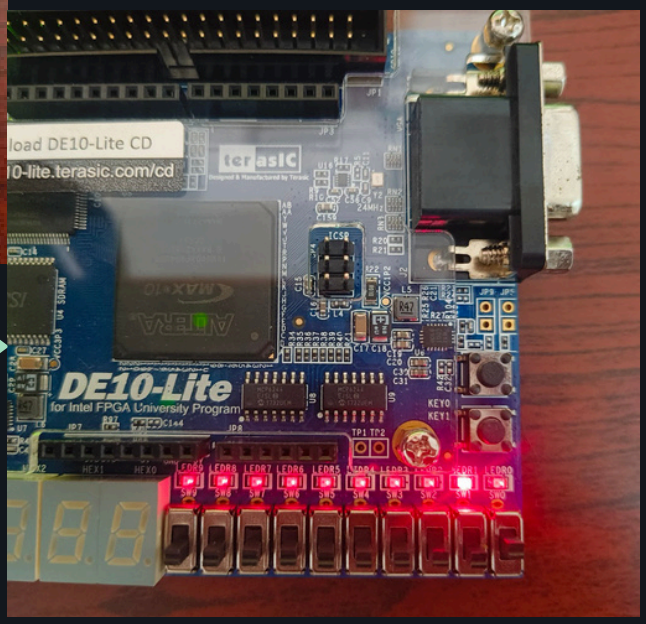


How `S` Affects `A` and `B`:

- When `S = 0`, the output `Y` is directly connected to `A` regardless of the value of `B`.
  - If A = 0, then `Y = 0`.
  - If A = 1, then `Y = 1`.
- When `S = 1`, the output `Y` is directly connected to `B` regardless of the value of `A`.
  - If B = 0, then `Y = 0`.
  - If B = 1, then `Y = 1`.

The second LED from the last shows the output of the 2-to-1 multiplexer (Y).
The last three switches control the inputs: A , B , S (selector)

If S = 1, A = 0, and B = 0, Y = 0
because when S = 1,
the output is determined by B,
and since B = 0,
the output Y will be 0.
The second LED will display 0.



If S = 0, A = 0, and B = 1, Y = 0
because when S = 0,
the output is determined by A,
and since A = 0,
the output Y will be 0.
The second LED will display 0.

This Verilog module represents a 3-to-8 Decoder, which takes in 3 input bits and decodes them into one of 8 output lines. Each output corresponds to one unique combination of the input bits.

```
// 3-to-8 Line Decoder Module
// This module implements a 3-to-8 decoder that converts a 3-bit input (A) to an 8-bit
// output (Y). The output Y will have exactly one bit set to '1' based on the value
// of the 3-bit input A, while all other bits in Y will be '0'. If the input A
// is within the range [000, 111], the corresponding bit in Y will be set. If A
// is invalid (which isn't possible here due to the 3-bit limit), the output defaults to all zeros.

module Decoder (
    input [2:0] A,    // 3-bit input (A) representing the binary value to decode
    output [7:0] Y    // 8-bit output (Y) where only one bit will be '1' based on A
);

// A ternary operator is used to check each possible value of A (from 3'b000 to 3'b111).
// The corresponding output Y will have only one bit set to '1' based on A's value.
// For example, if A = 3'b000, then Y = 8'b00000001, if A = 3'b001, then Y = 8'b00000010, and so on.
assign Y = (A == 3'b000) ? 8'b00000001 :  // If A is 000, Y is 00000001
           (A == 3'b001) ? 8'b00000010 :  // If A is 001, Y is 00000010
           (A == 3'b010) ? 8'b00000100 :  // If A is 010, Y is 00000100
           (A == 3'b011) ? 8'b00001000 :  // If A is 011, Y is 00001000
           (A == 3'b100) ? 8'b00010000 :  // If A is 100, Y is 00010000
           (A == 3'b101) ? 8'b00100000 :  // If A is 101, Y is 00100000
           (A == 3'b110) ? 8'b01000000 :  // If A is 110, Y is 01000000
           (A == 3'b111) ? 8'b10000000 :  // If A is 111, Y is 10000000
                           8'b00000000;  // Default case: if A is invalid, Y is all zeros (not possible in this case)

endmodule
```

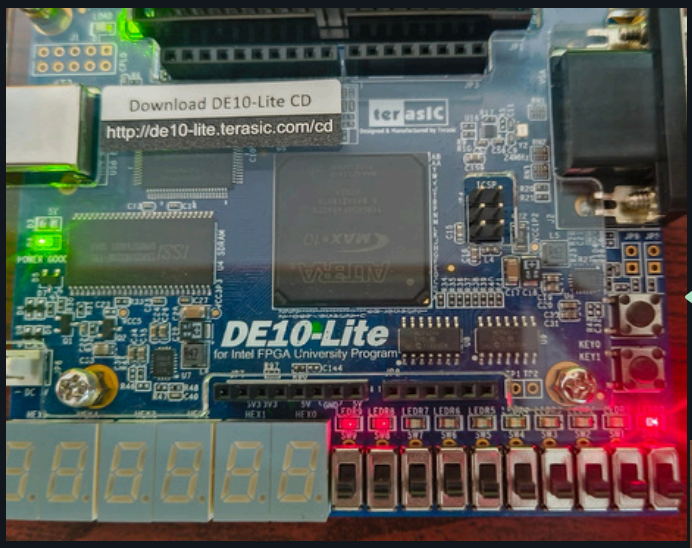
Truth Table
:) Decoder

X (Input)	Y (Input)	Z (Input)	D7	D6	D5	D4	D3	D2	D1	D0
0	0	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	1	0	0	0	0	0	0	1	0	0
0	1	1	0	0	0	0	1	0	0	0
1	0	0	0	0	0	1	0	0	0	0
1	0	1	0	0	1	0	0	0	0	0
1	1	0	0	1	0	0	0	0	0	0
1	1	1	1	0	0	0	0	0	0	0

The last 8 LEDs represent D7 to D0, corresponding to the output of the 3-to-8 decoder. The last 3 switches are X, Y, and Z, which represent the 3-bit input to the decoder (A).

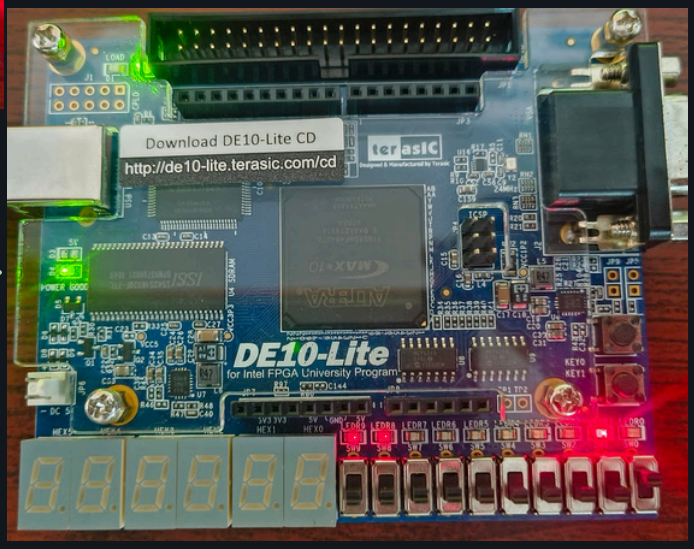
A = X, Y, and Z (3-bit input)
Y = D7 to D0 (8-bit output)

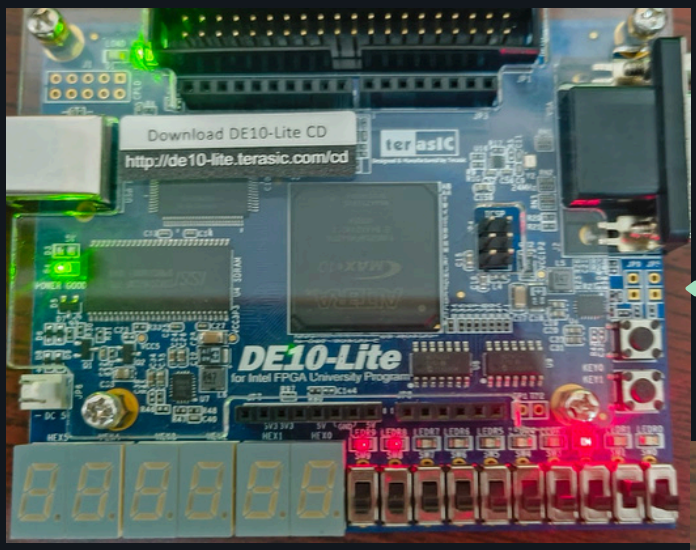
A = XYZ (3-bit input)
Y = D7 D6 D5 D4 D3 D2 D1 D0 (8-bit output)



For the input combination A = 000,
the output is Y = 00000001.

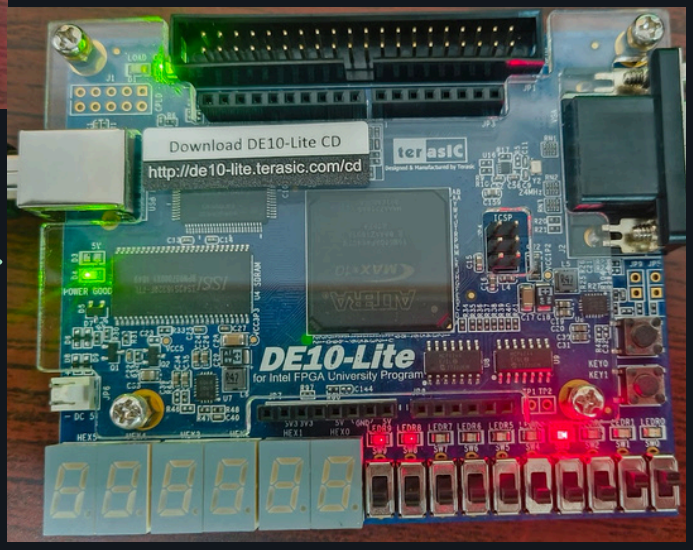
For the input combination A = 001,
the output is Y = 00000010.

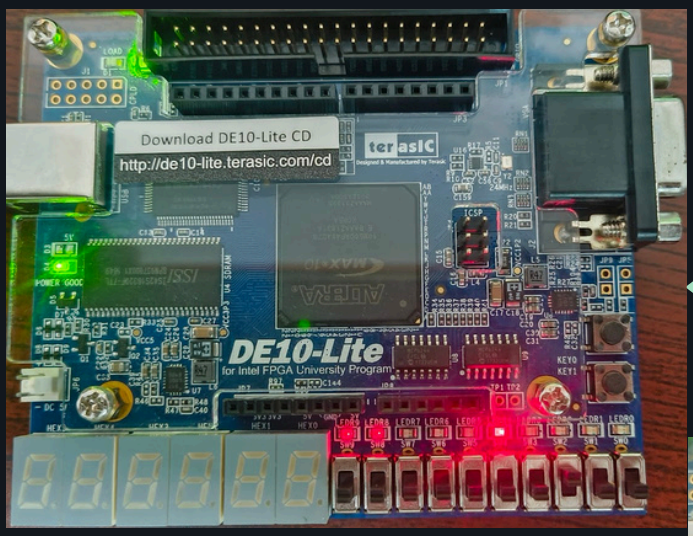




For the input combination A = 010,
the output is Y = 00000100.

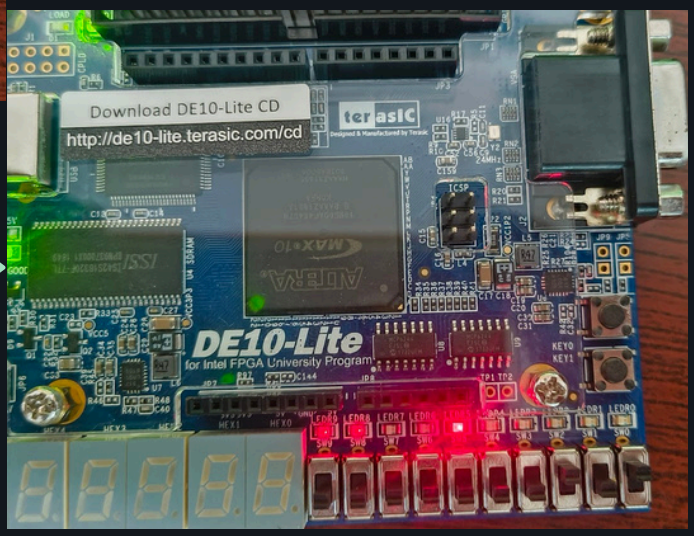
For the input combination A = 011,
the output is Y = 00001000.

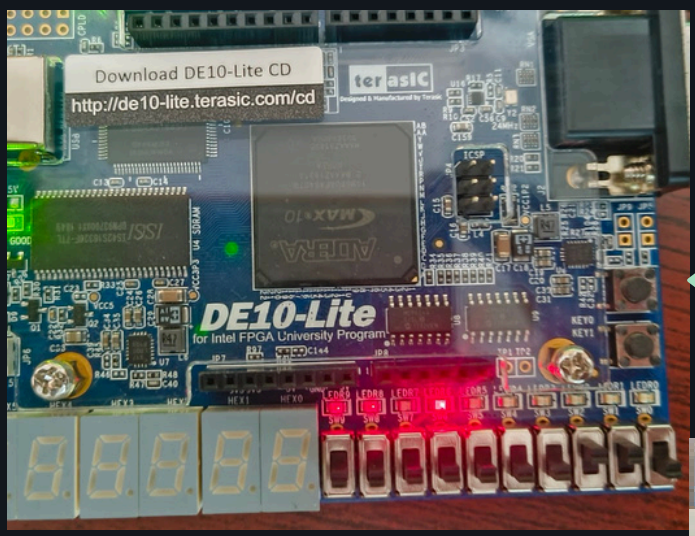




For the input combination A = 100,
the output is Y = 00010000.

For the input combination A = 101,
the output is Y = 00100000.





For the input combination A = 110,
the output is Y = 01000000.

For the input combination A = 111,
the output is Y = 10000000.

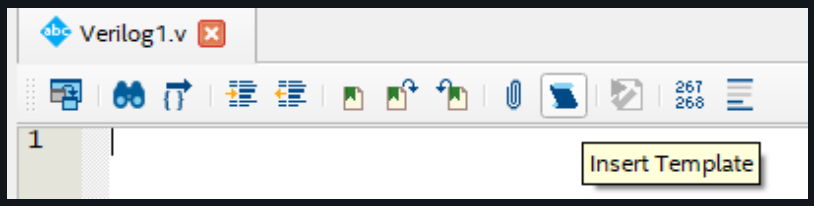
How to Use Templates & IPs in Quartus to Generate Base Code

Note: In both the Insert Template and IP methods, make sure that the Verilog file name matches the module name exactly. This is required for Quartus to compile and analyze the design correctly.

Method 1: Insert Template (for quick base code)

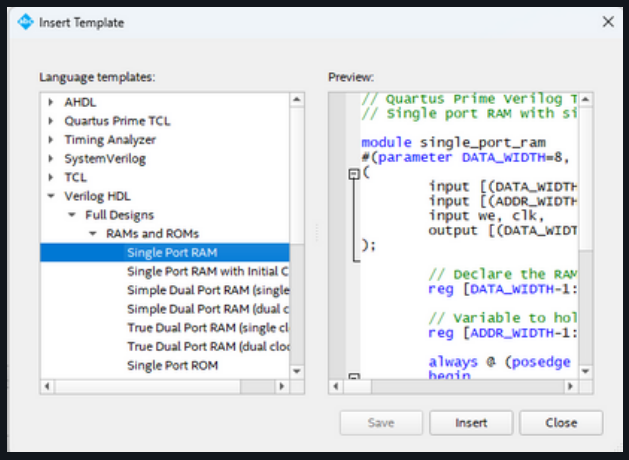
Create a new project and open a new Verilog HDL file as usual.

With the Verilog file open, click Insert Template (usually found in the toolbar or right-click menu).



In the dialog:

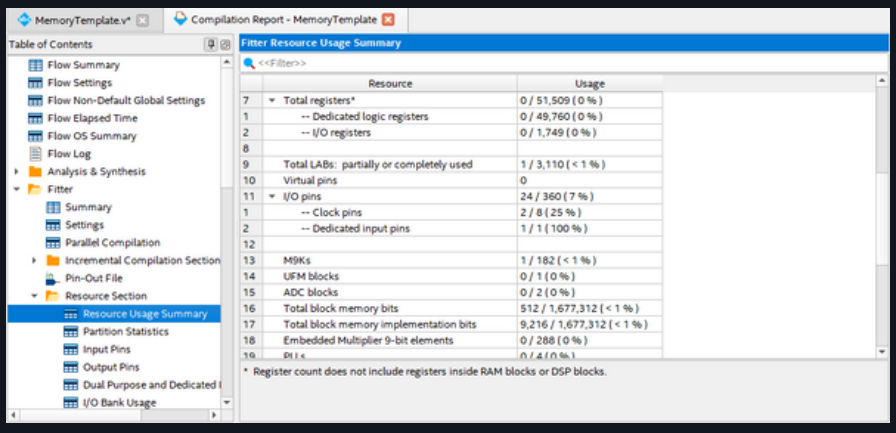
- Select the language (e.g., Verilog HDL).
- Navigate to the desired template:
- For example: Verilog HDL → Full Designs → RAMs and ROMs → Single Port RAM



Click Insert to add the code into your Verilog file.

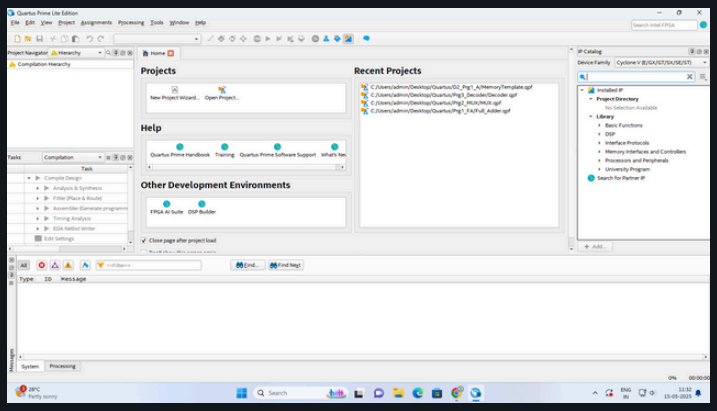
Modify or integrate it as needed, then follow the usual flow: analysis, synthesis, pin assignment, etc.

Check Tools → Netlist Viewers → Resource Usage Summary to review elements like M9K blocks used for memory.



Method 2: Use IP Catalog (for more customizable modules)

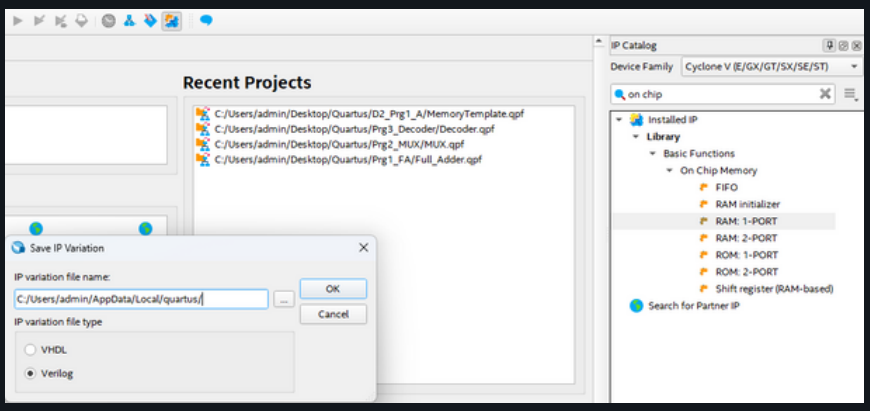
Create a new project (no need to add a Verilog file initially).



On the right panel, open the IP Catalog.
Use the search bar to find your desired component.

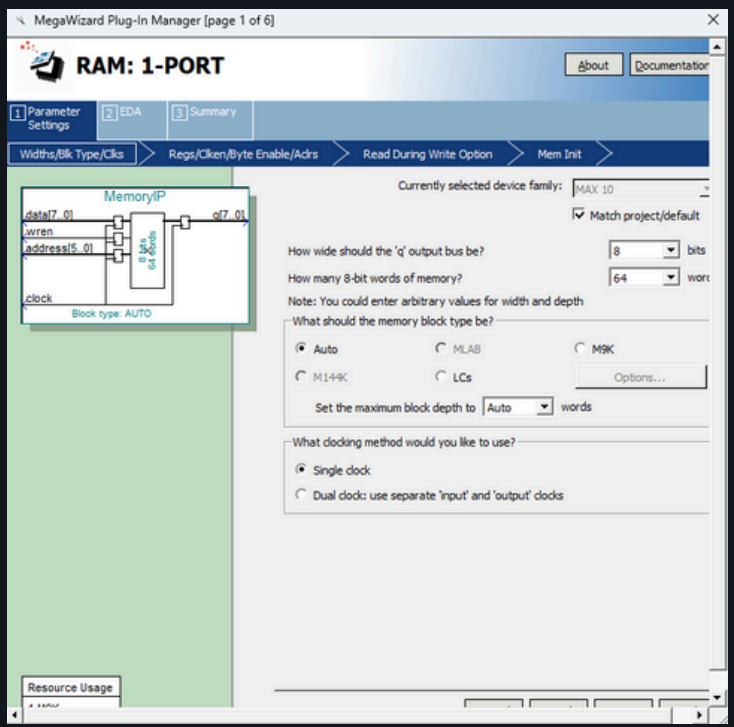
- For example, search for: On-Chip Memory (RAM 1-PORT)

Click the result to open the configuration wizard.

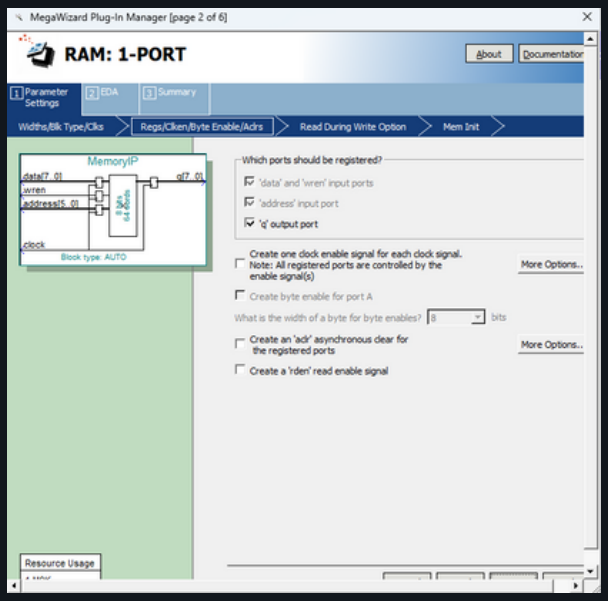


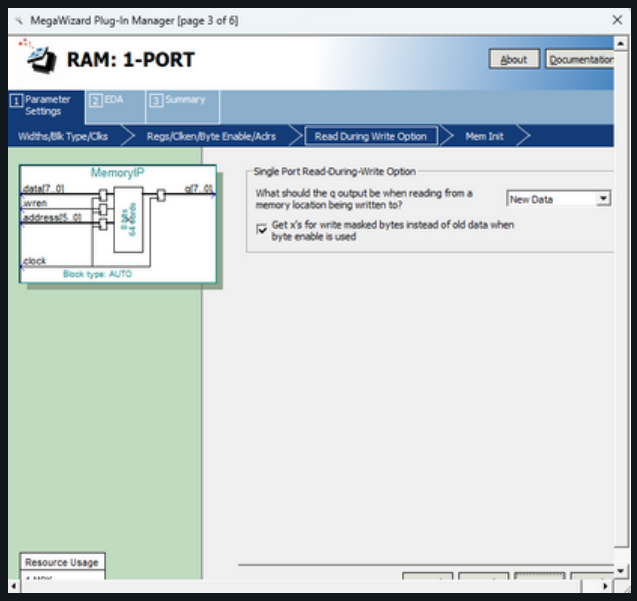
Set:

- Word size (number of bits per word – defines data width)
- Number of words (memory depth – how many entries)



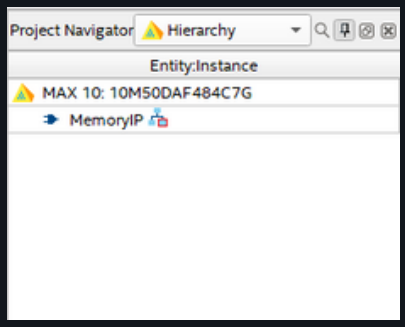
Continue through the configuration steps and click Finish.

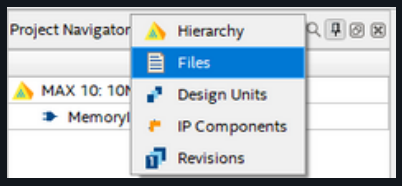


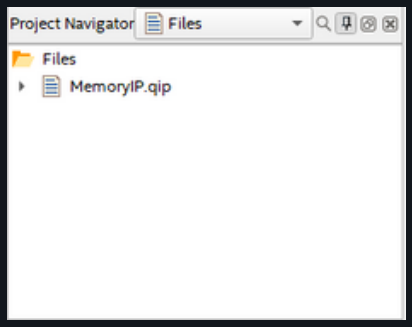


Quartus will generate the necessary files (e.g., .v, .qip, etc.).

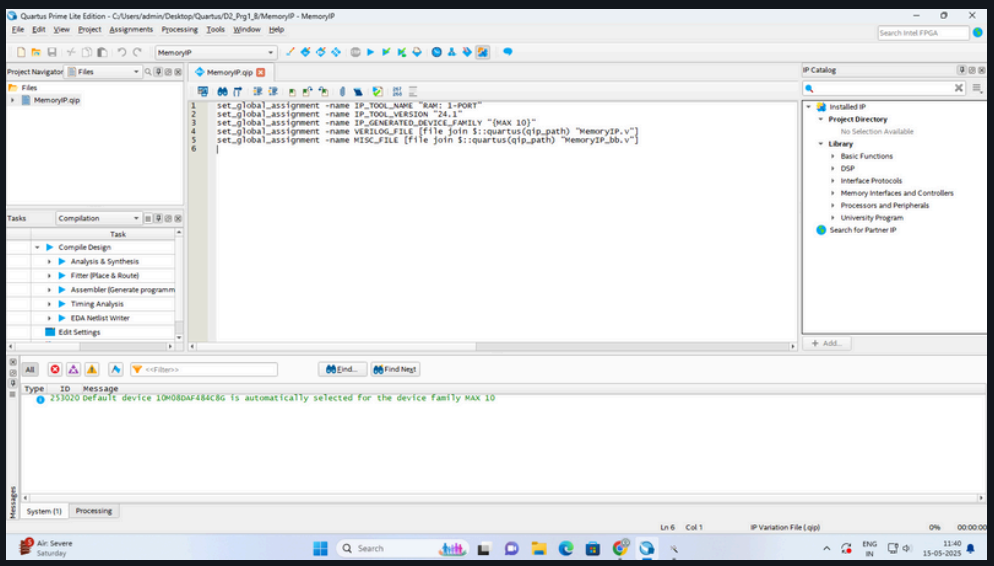
In the Project Navigator, change the view from Hierarchy to Files to see all generated components.



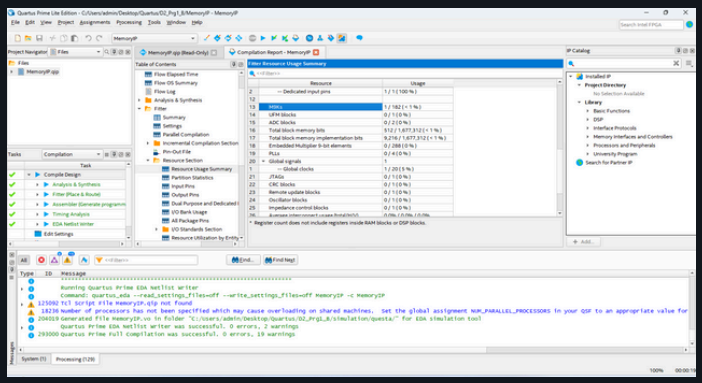




Compile and simulate or program as usual.



Check Tools → Netlist Viewers → Resource Usage Summary to review elements like M9K blocks used for memory.

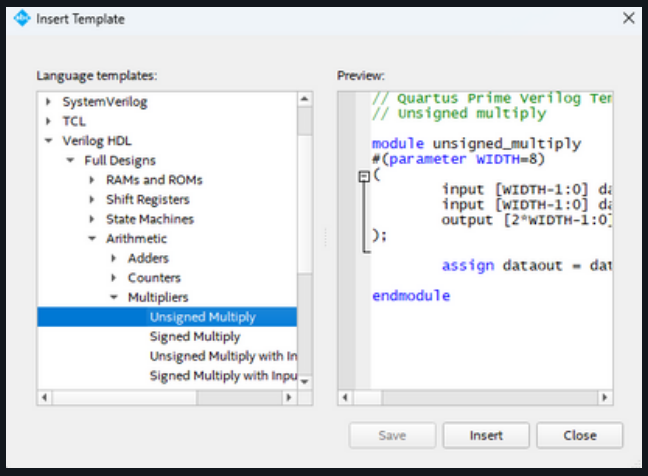


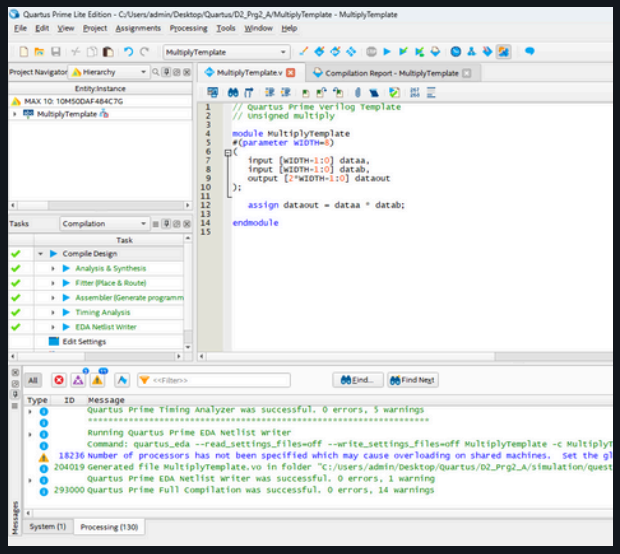
Multiply Code – Using Template and IP in Quartus

Method 1: Insert Template (for quick base code)

Open a new project and Verilog file.

- Click Insert Template → Verilog HDL → Full Designs → Arithmetic → Unsigned Multiply → Insert





🔍 Explanation:

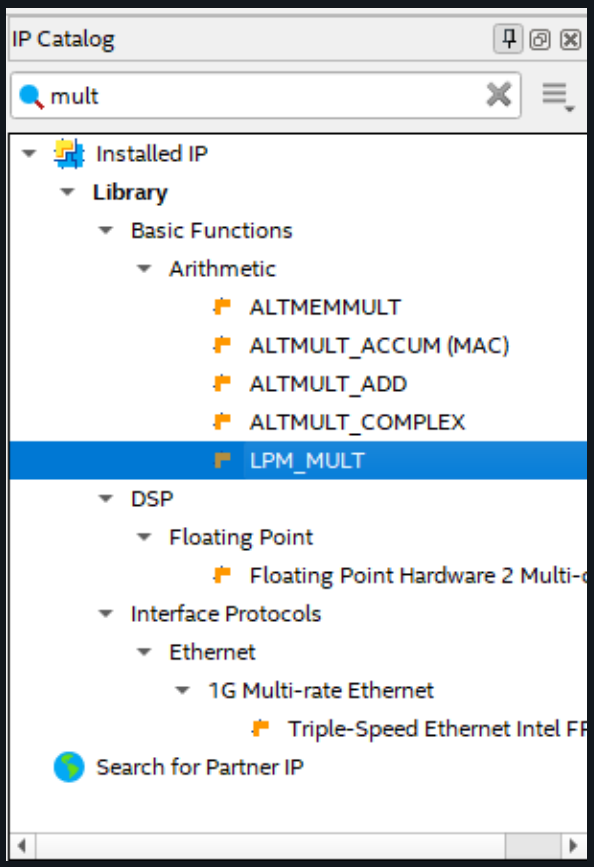
- WIDTH sets the bit-width of the two inputs.
- dataa, datab are the input numbers to be multiplied.
- dataout is the output, with a width of 2 * WIDTH to avoid overflow.

🛠️ To modify output:

- Change WIDTH to match your desired input size:
- Example: For 4-bit inputs, set WIDTH=4, output becomes 8 bits.
- This affects both input size and the output range.

➡ Compile, run Analysis & Synthesis, and verify with RTL/Tech Viewer as usual.

Method 2: Use IP Catalog (for more customizable modules)



Start a new or existing project.

- In the IP Catalog, search for mult or browse:
- Basic Functions → Arithmetic → LPM_MULT

In the configuration wizard:

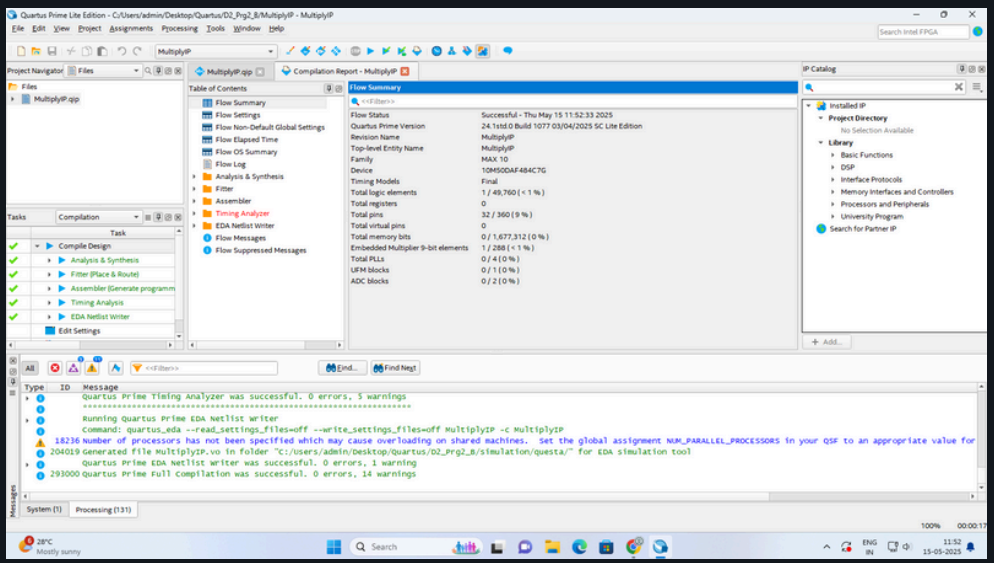
- Set dataa width and datab width
- Choose unsigned or signed multiplication
- Choose if you want pipelined or combinational operation

Click Finish to generate the multiplier IP.

In Project Navigator, switch to Files view to access the generated .v and .qip files.

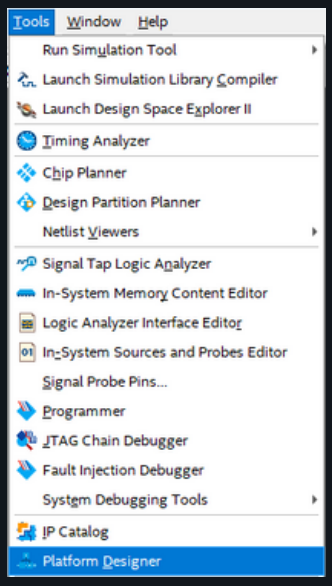
Instantiate the module in your top-level design.

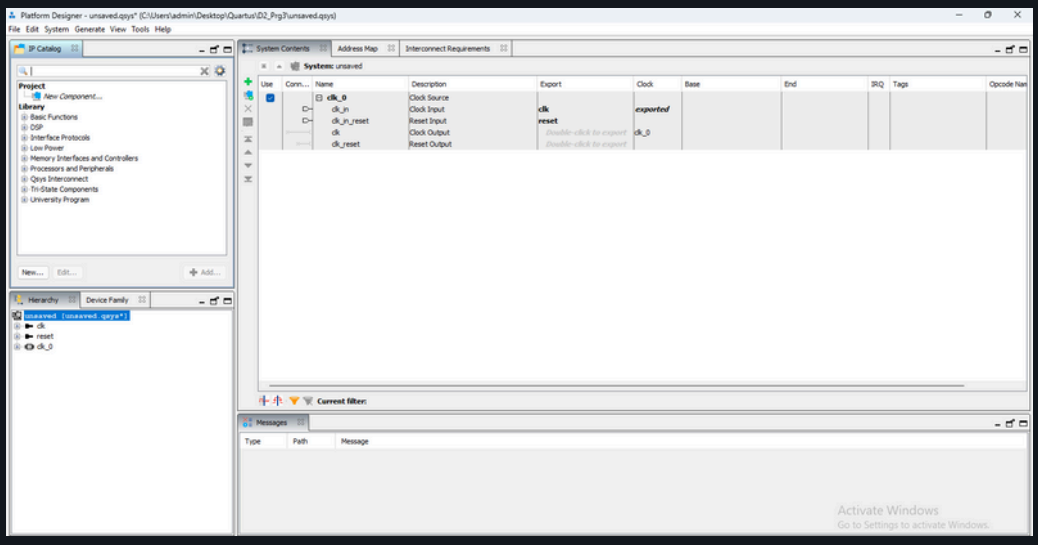
Execute the flow as usual: analysis → synthesis → fitting → programming



Steps to Use Platform Designer in Quartus

Open Platform Designer: Go to Tools → Platform Designer.

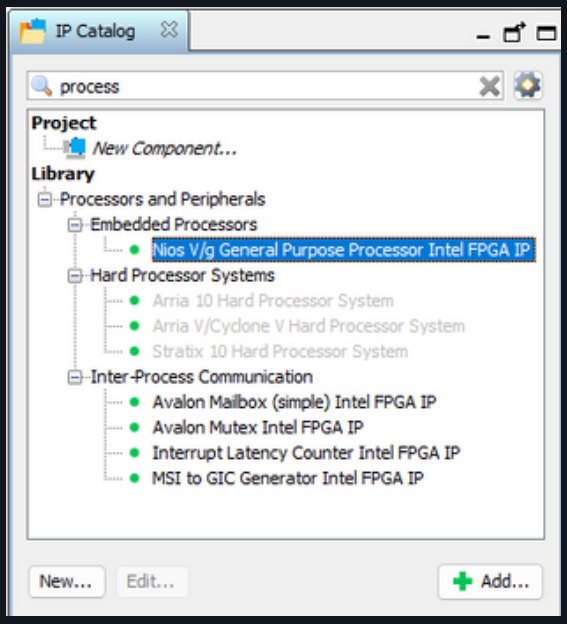


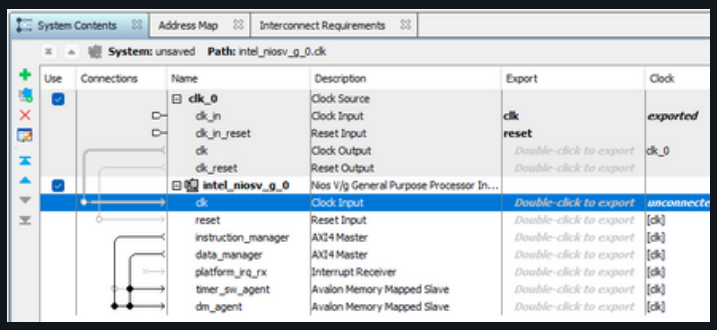


Add Nios II Processor:

- On the right side, in the IP Catalog, search for:
- Nios II Processor (Intel)
- Double-click to add it to the block diagram.

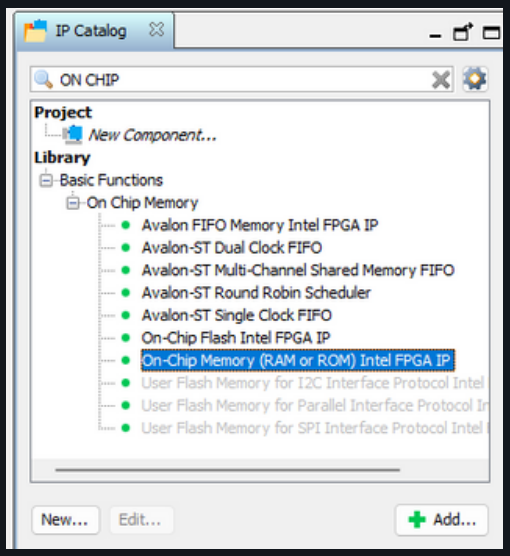
⚠ Some connection errors may show up initially—these can be resolved in later steps, so continue.





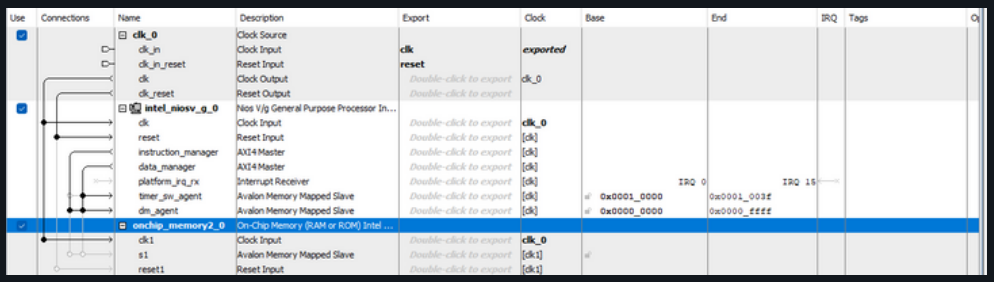
Connect Processor Clock and Reset:

- Connect clock and reset of the processor to the system clock and reset.
- Do this by clicking the connection dots near the signal lines.

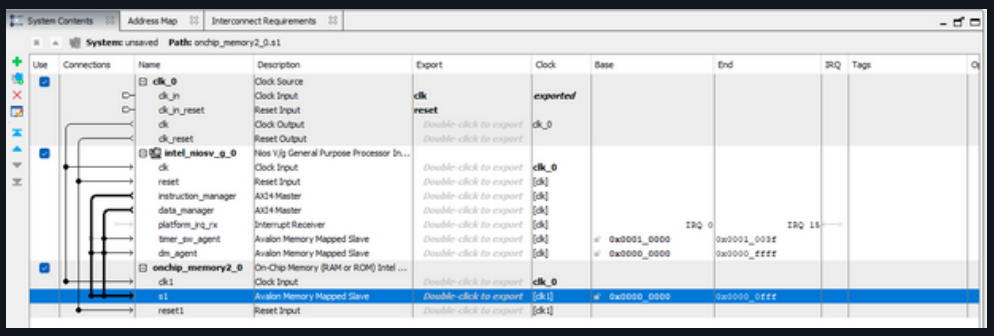


Add On-Chip Memory:

- Search for: On-Chip Memory (RAM or ROM Intel)
- Add it to the diagram.

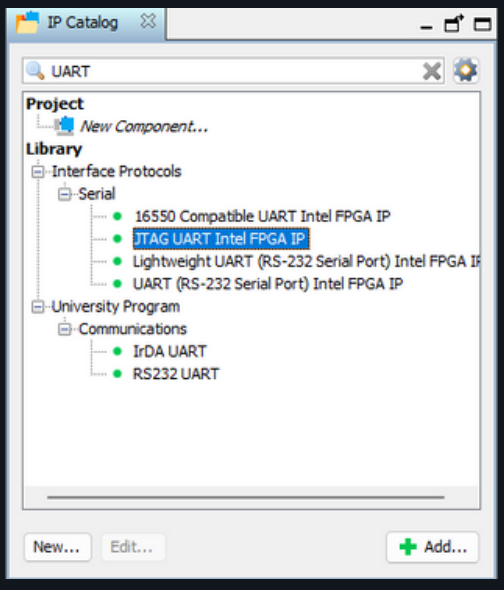


Connect clk and reset to the system signals.
Connect the processor master interface to the OCM slave.



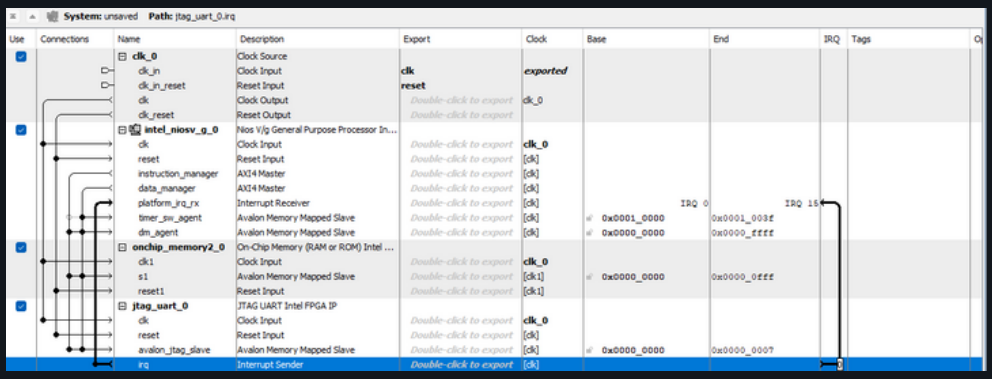
Add JTAG UART:

- Search: UART
- Add it JTAG UART (Intel)

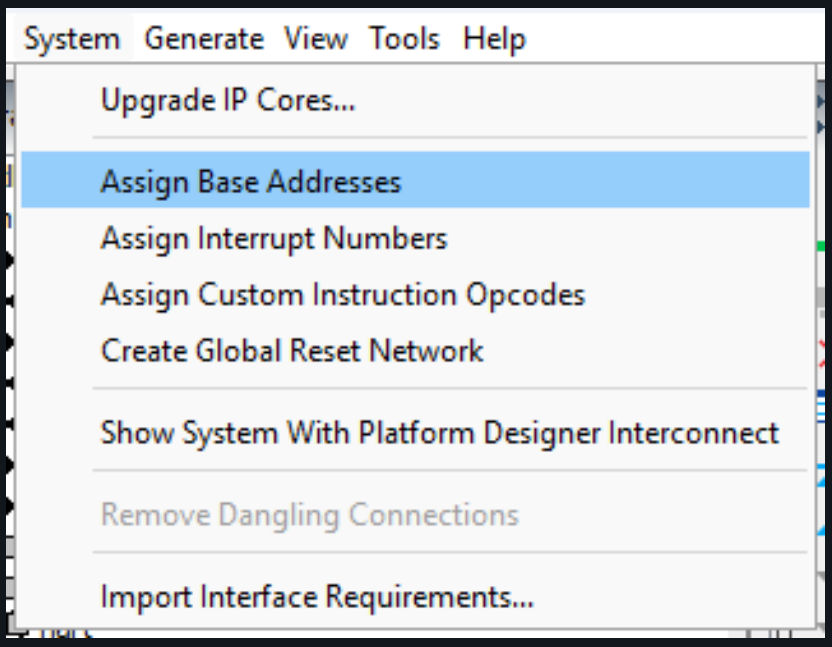


Connect:

- clk and reset to system.
- slave and interrupt lines to the processor.



Assign Base Addresses:
Go to System → Assign Base Addresses.
This ensures no two components share the same memory address.

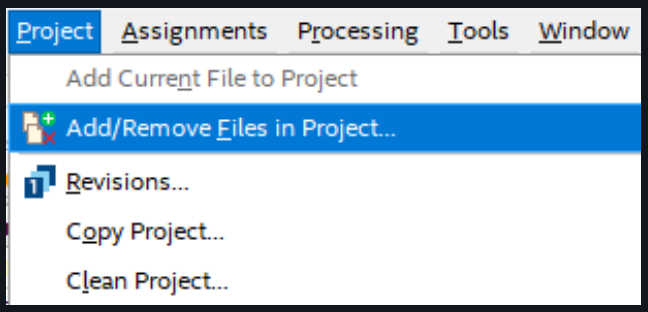


Map Processor Reset/Exception Vectors:

- Double-click the Nios II processor block.
- Set the reset and exception address to point to the On-Chip Memory.

Generate HDL: Click Generate → Generate HDL.
This saves your configuration into HDL files so you don't have to recreate the design manually.

Note: When you save your design in Platform Designer, it is stored as a .qsys file, which stands for Quartus System Integration File. This file contains your full hardware system configuration (including processors, memory, and peripherals) and can be regenerated or modified later without rebuilding everything from scratch.

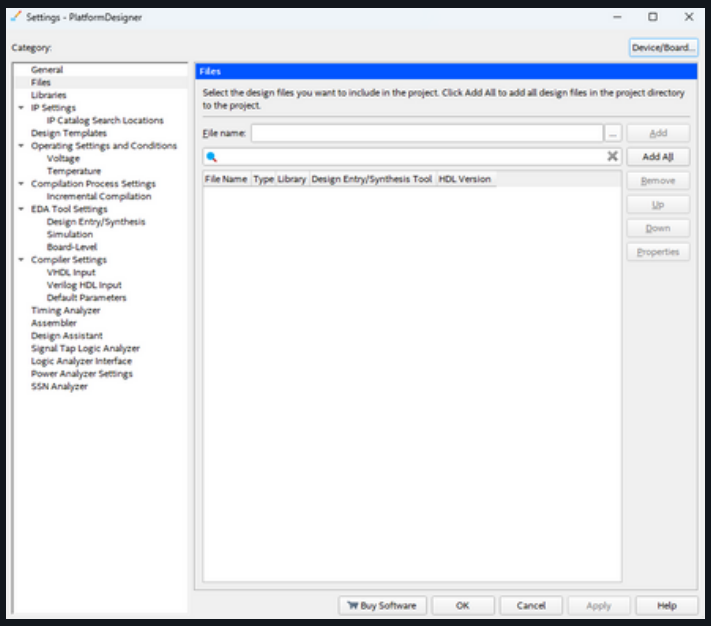


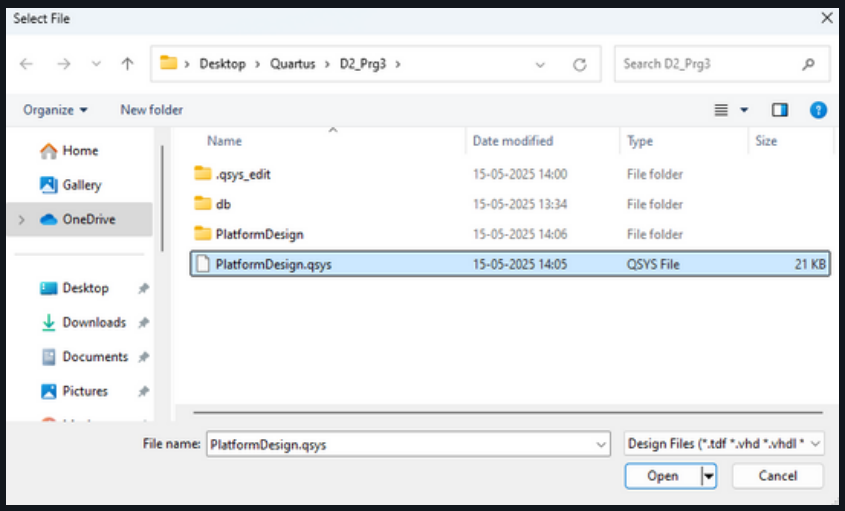
Add Generated Files to Project:

- Go to Project → Add/Remove Files in Project.
- Click the three dots next to the file path search bar.
- Locate the generated HDL file and add it.

To Add Generated Files to Project:

- Go to Project → Add/Remove Files in Project.
- Click the three dots next to the file path search bar.
- Locate the generated HDL file and add it.



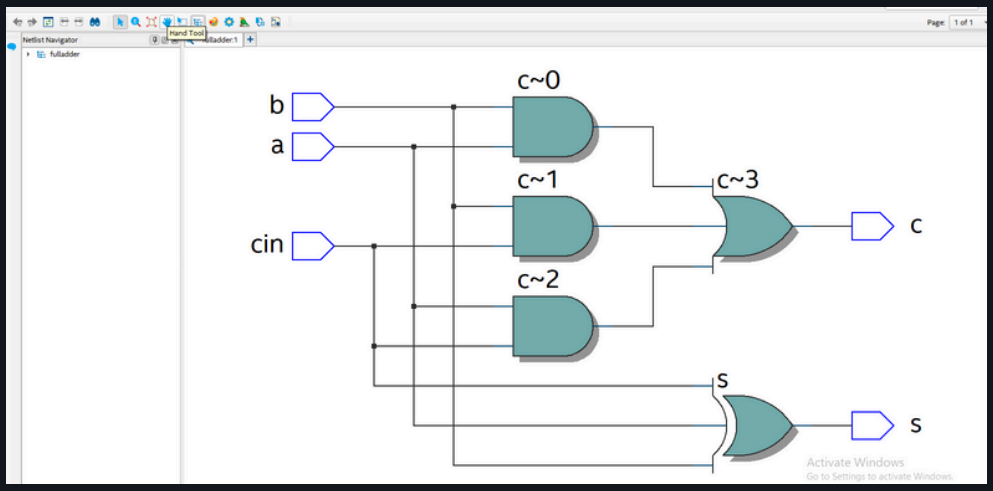


- Locate the generated HDL file and add it.

Extra Section: Visual Demonstrations & RTL Snapshots

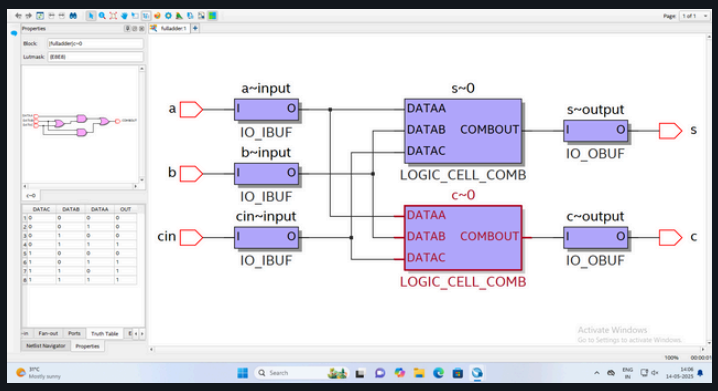
Additional visual proofs and detailed hardware mappings captured during experimentation

- Register and Basic Logic Components



RTL view of a 4-bit register with parallel load capability combined with a 3-bit ripple carry adder.

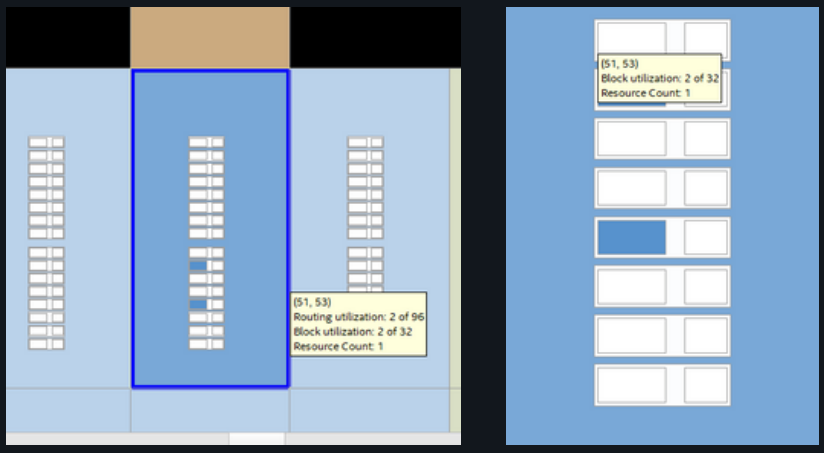
- Full Adder: RTL & Physical Implementation



RTL Viewer output showing logic layout of a full adder module in Quartus.

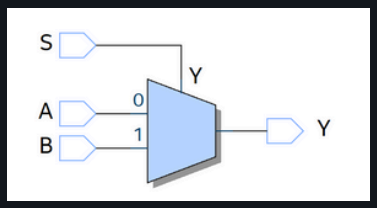
Snapshot from FPGA device viewer, showing how the full adder is mapped physically on chip.

Image:



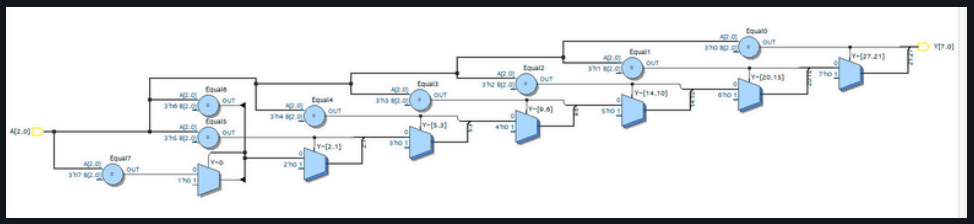
These zoomed-in visuals demonstrate how multiple logic blocks (LUTs) are utilized for a simple full adder, emphasizing real hardware usage.

- Multiplexers and Decoders

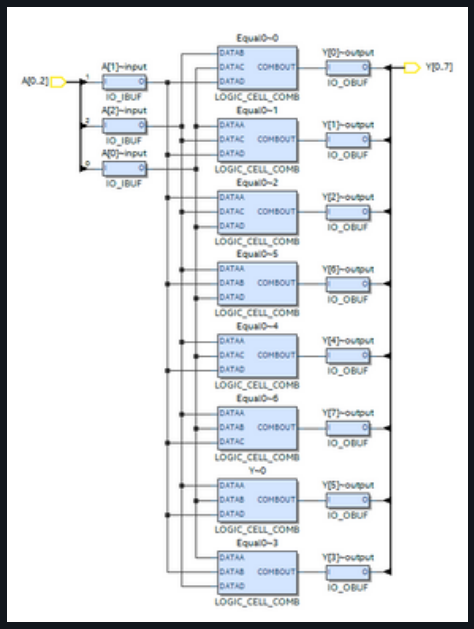


Basic RTL view of a 2-to-1 multiplexer showing data flow.

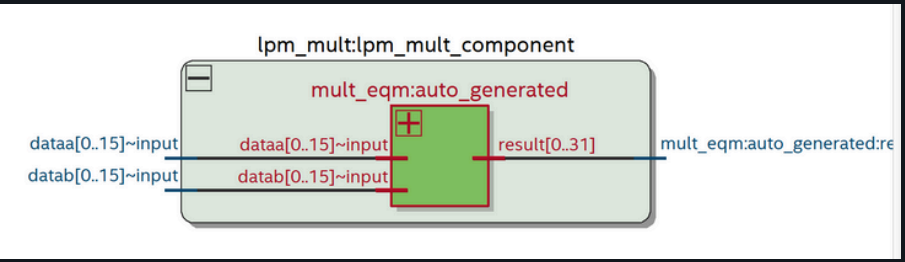
RTL representation of a 3-to-8 decoder logic block.

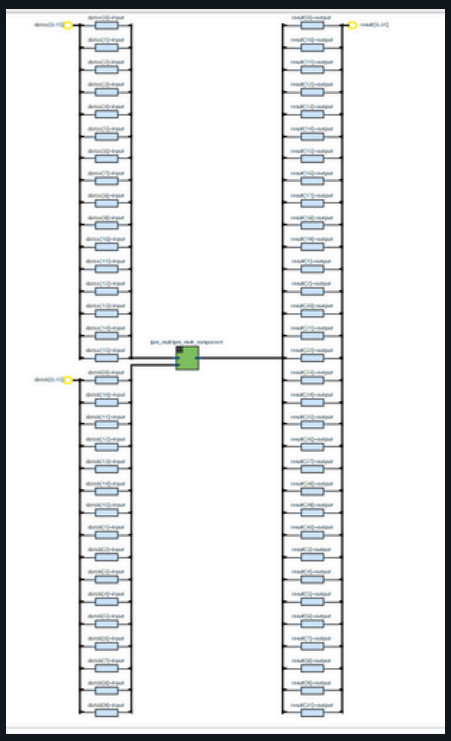


Technology Map Viewer output showing how the decoder logic maps to FPGA resources.



Technology Map view of a multiplier generated using the LPM_MULT IP in Quartus.





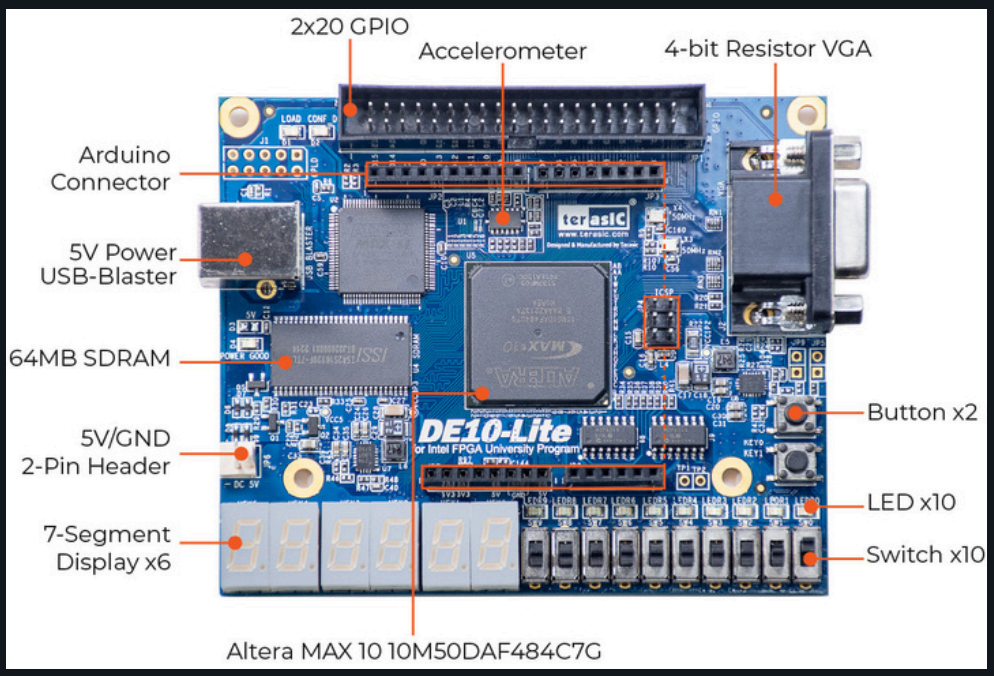
Detailed internal view of a 16-bit multiplier implementation, showcasing the use of M9K blocks and logic elements.

Box showing family and filter name

Family: MAX 10

Name filter: 10M50DAF484C7G

Top view of the DE10-Lite FPGA board. Major components like the MAX 10 FPGA, push buttons, switches, LEDs, USB-Blaster, and 7-segment displays are clearly labeled for reference.



Rear view of the DE10-Lite board showing power circuitry, traces, and supporting components. Useful for identifying board layout and debugging physical connections.