

# Hardware Design Lab

## Lab 3: Finite State Machines

**Instructor: Dr. Haiyu Mao**

**TA:**

**Zihao Pu**

**Ali Alsarraf**

**Stephen Johannesson**

26 FEB 2026

Gratefully acknowledge:  
Prof. Onur Mutlu (ETH)  
Dr. Yair Linn (TRIUMF Canada)

# Table of Contents

---

- ❑ Part 1: Finite State Machines Recap
- ❑ Part 2: Lab Task – Running an FSM on basys3
- ❑ Part 3: Coursework Task 3 Introduction

# Learning Outcomes

---

- ⑩ Understand the theory behind Finite State Machines (FSMs)
- ⑩ Distinguish between Moore and Mealy machine architectures
- ⑩ Read and understand a pre-written SystemVerilog FSM design
- ⑩ Learn how the 1011 sequence detector works and why it uses overlap logic
- ⑩ Run the design on the Basys 3 board and observe real hardware behaviour
- ⑩ Use the 7-segment display and LEDs as debugging tools

# Part 1

---

FSM RECAP

# Finite State Machines

---

- ❑ What is a **Finite State Machine** (FSM)?
  - A system with a **fixed set of possible states**, that moves between them **based on inputs**.
  - Next state is based on current **state + input**.
  
- ❑ Your **CPU is an FSM**. So is a traffic light. Any system that reacts **differently depending on its history** can be modelled this way.
  
- ❑ An FSM diagram shows:
  - Every situation the system can be in, and every path between them.
  - circles are states, arrows are transitions.
  
- ❑ **State diagrams let you reason about behaviour before writing a single line of code.**

# Finite State Machines (FSMs) Consist of

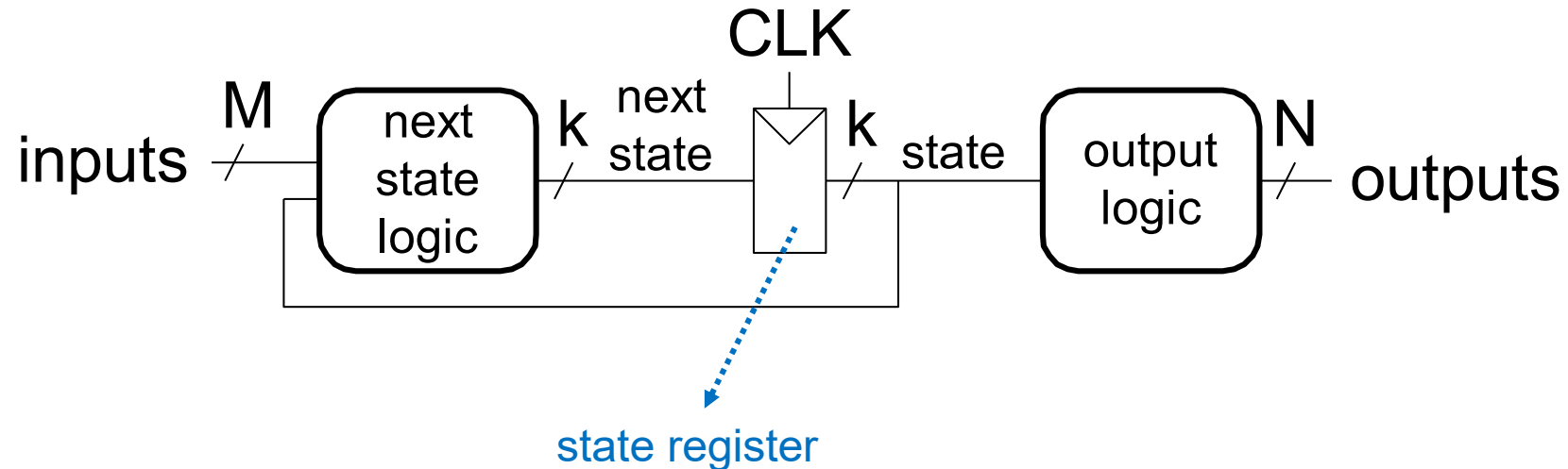
---

## □ Five elements:

1. **States** - every situation the system can be in
2. **Inputs** - external signals that drive transitions
3. **Outputs** - what the system communicates to the outside world
4. **Transition Rule** - for every state and input, where do you go next?
5. **Output Rule** - for every state, what do you **output**?

# Finite State Machines (FSMs)

- Each FSM consists of three separate parts:
  - next state logic
  - state register
  - output logic

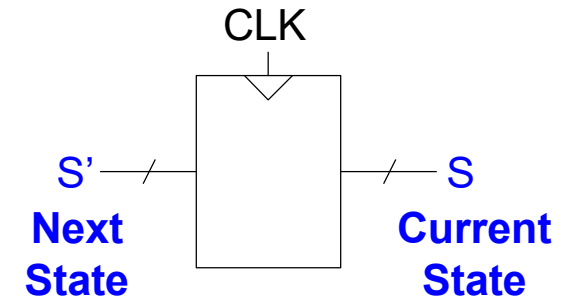


At the beginning of the clock cycle, the next state is latched into the state register

# Finite State Machines (FSMs) Consist of

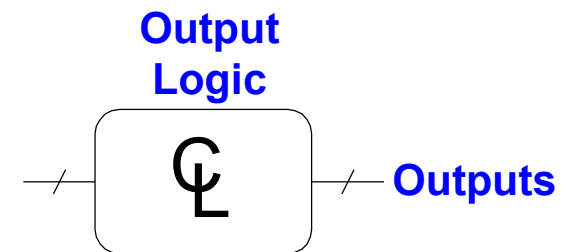
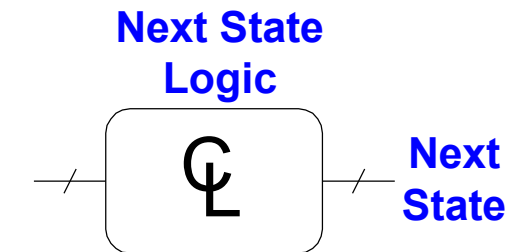
## □ Sequential Circuits

- Output depends on current input and history (state)
  - Has memory, stored in flipflops
  - Clocked, changes only on clock edge
- State register(s)
    - Store the current state and
    - Load the next state at the clock edge



## □ Combinational Circuits

- Output only depends on current input
  - No clock, no memory
- Next state logic
    - Determines what the next state will be
  - Output logic
    - Generates the outputs



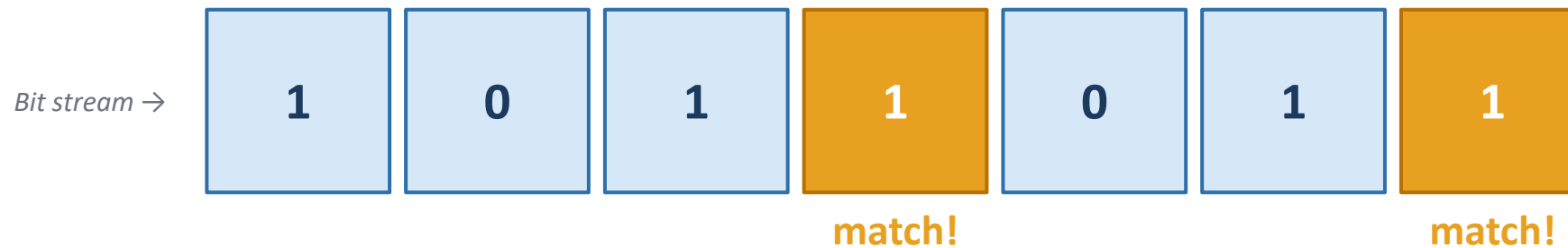
# Part 2

---

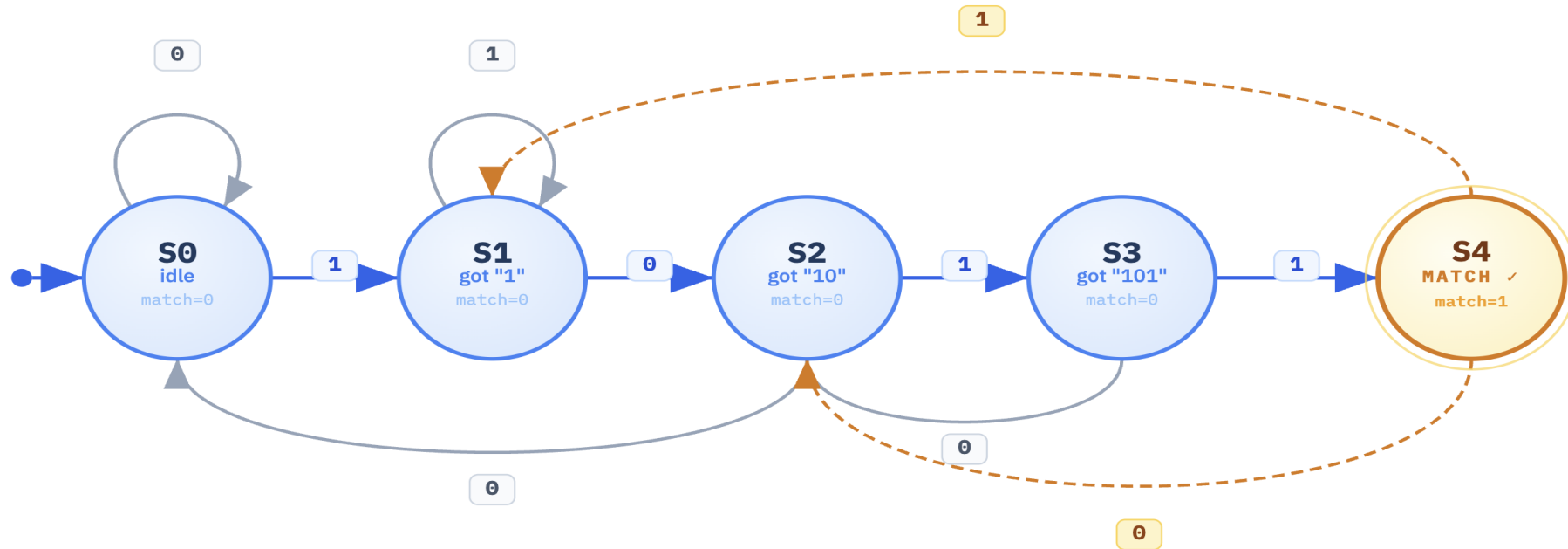
LAB 3: LAB TASK – RUNNING AN FSM ON BASYS3

# Background: Sequence Detector

- **Goal: detect the bit pattern 1-0-1-1 in a serial bit stream**
- One input bit (din) is presented at a time
- When the last four bits received spell '1011', the output match = 1
- We use overlap detection. After matching 1011, we don't start from scratch:
  - The tail of the sequence may be the start of the next one
  - Example: 1 0 1 1 0 1 1 — a second match is detected at the end



# Background: State Diagram



State	din = 0	din = 1	match output
S0 (idle)	S0	S1	0
S1 (got '1')	S2	S1	0
S2 (got '10')	S0	S3	0
S3 (got '101')	S2	S4	0
<b>S4 (MATCH)</b>	S2	S1	<b>1 ✓</b>

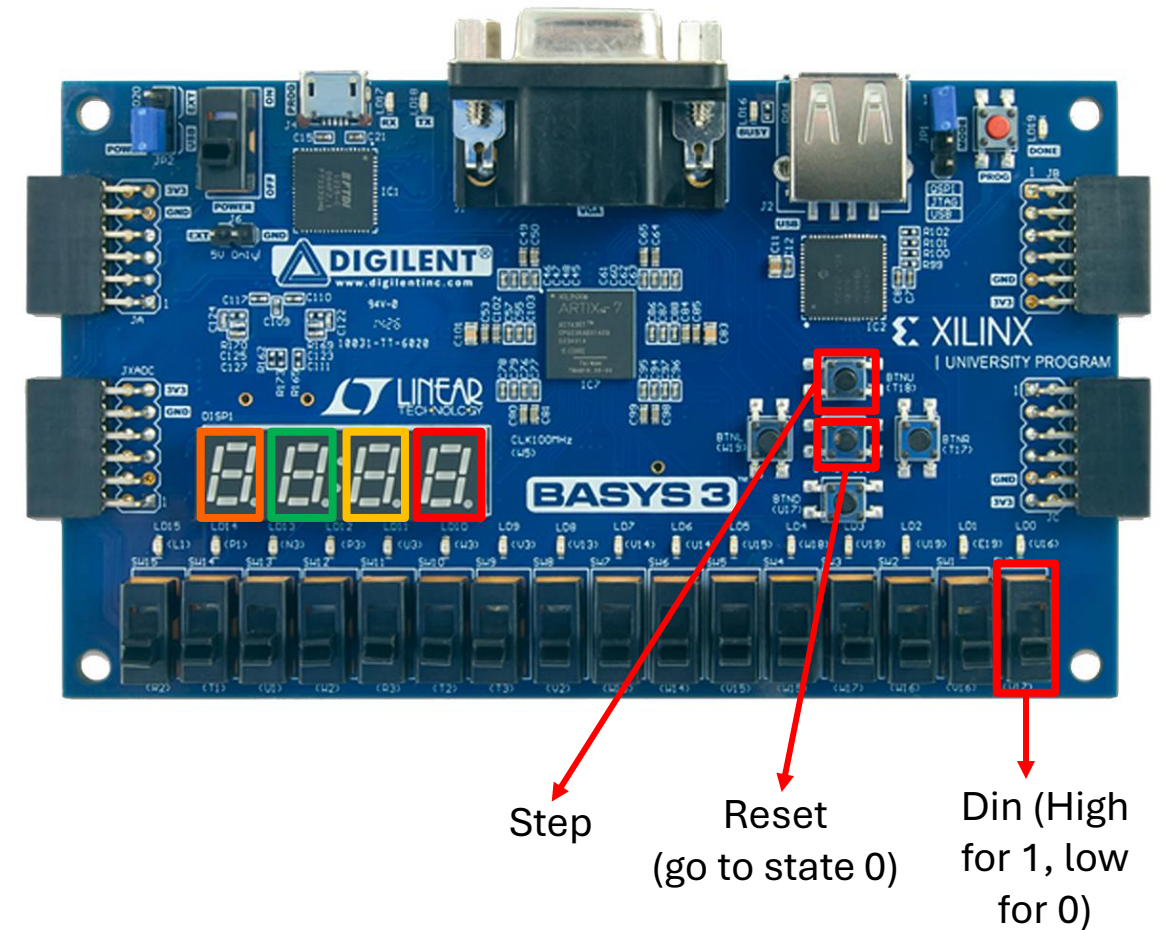
# Background: FSM Example

- The bits are left-aligned as they build up, so bit #1 is the MSB. 1 = 0x1, 10 in binary = 0x2, 101 = 0x5, 1011 = 0xB.
- On the Basys 3, the **history shift register** shifts in each new bit from the right.
- The match output only goes high **after** the 4th bit is clocked in and the FSM reaches S4 — it is a Moore output so it reflects the current state, not a combinational response to the input.

Step	din	Bits so far	Hex	State after input	match
<u>✓ Correct sequence: 1 → 0 → 1 → 1 (pattern 1011 detected)</u>					
1	1	1	0x1	S1 — got "1"	0
2	0	10	0x2	S2 — got "10"	0
3	1	101	0x5	S3 — got "101"	0
4	1	1011	0xB	S4 — MATCH ✓	1
<u>✗ Incorrect sequence: 1 → 1 → 0 → 0 (no match — returns to idle)</u>					
1	1	1	0x1	S1 — got "1"	0
2	1	11	0x3	S1 — still "1" (loop)	0
3	0	110	0x6	S2 — got "10"	0
4	0	1100	0xC	S0 — <u>idle</u> (no match)	0

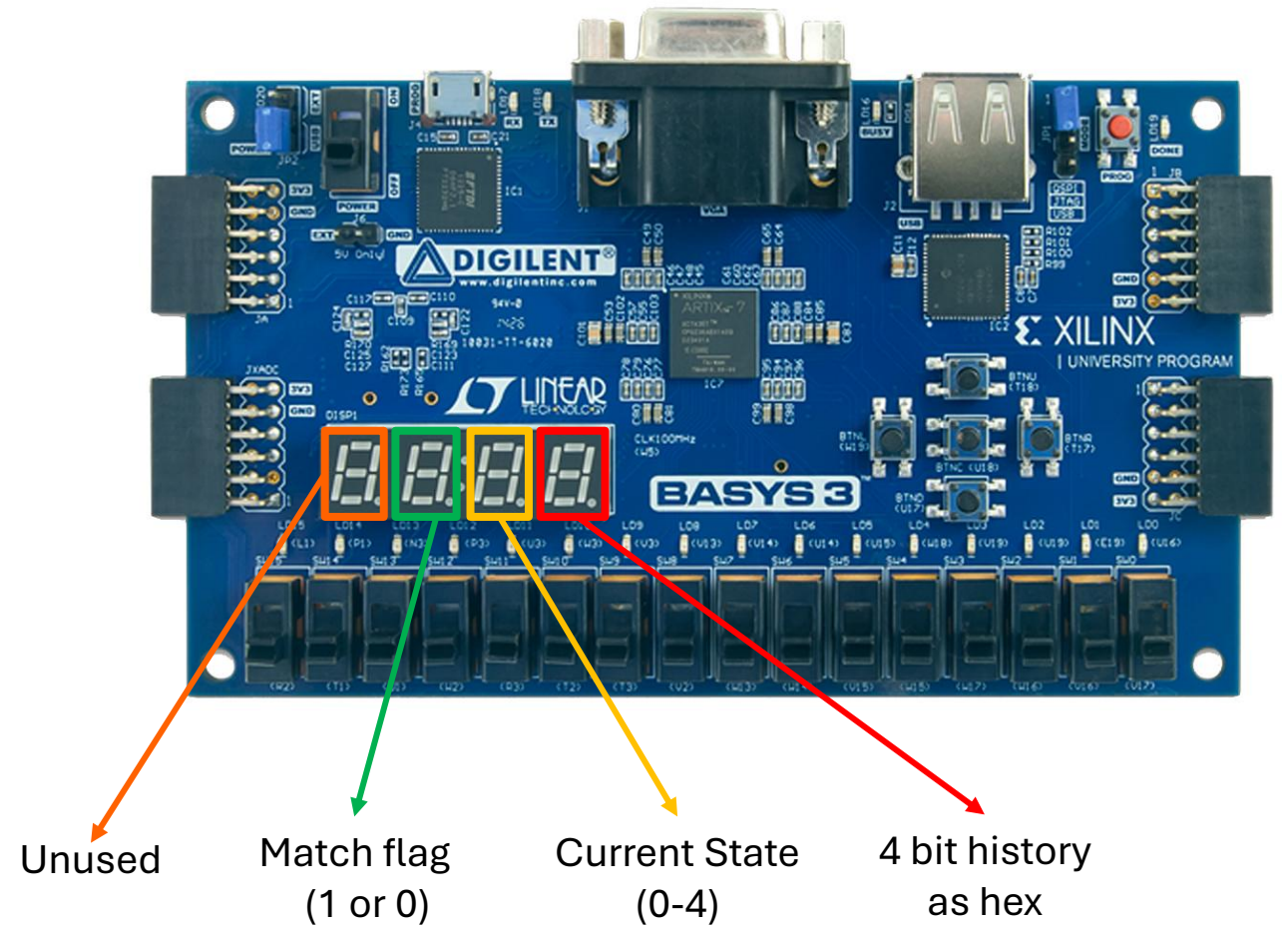
# FPGA Inputs

Signal	FPGA Pin	Type	Description
Clk	W5	Clock	100 MHz
SW[0]	V17	Switch	Din - the serial input bit to feed into the FSM
BTNC	U18	Button	Reset
BTNU	T18	Button	Step - advances FSM by one bit



# FPGA Outputs

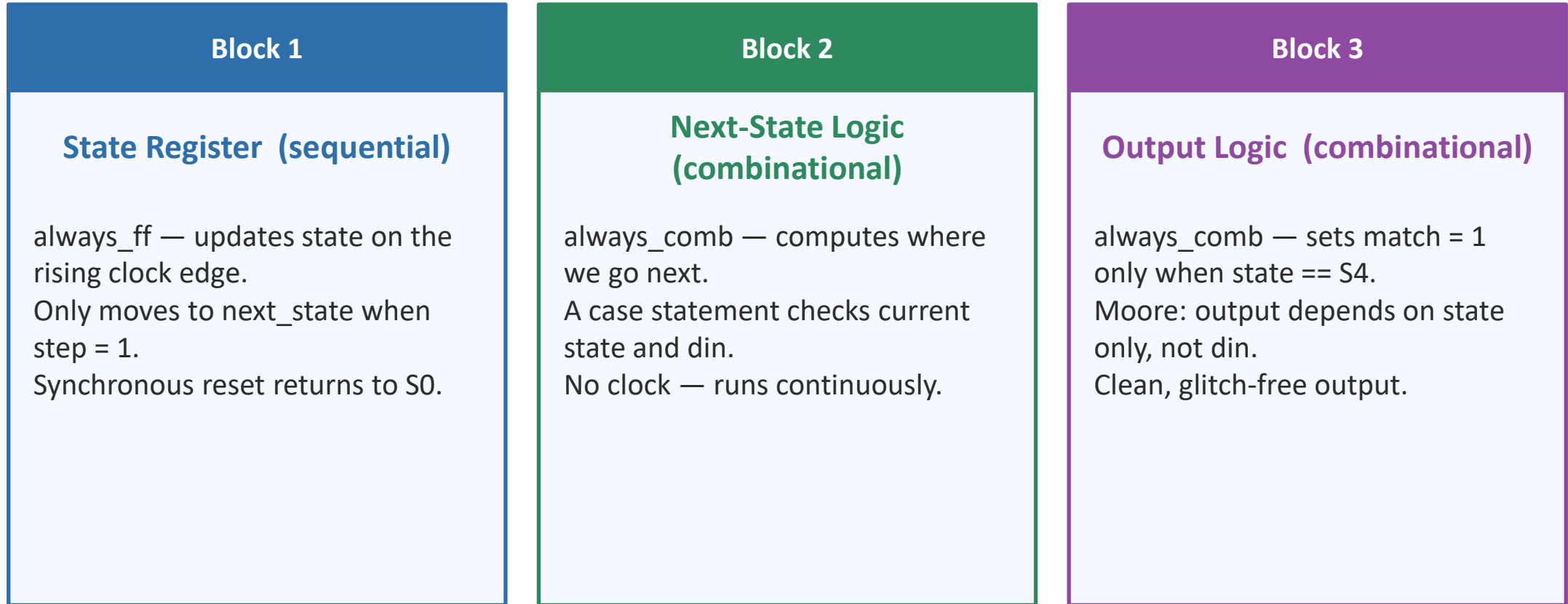
Signal	FPGA Pin	Type	Description
LED[0]	U16	LED	match — lights up when FSM is in S4
LED[1]	E19	LED	State bit 0 (LSB of current state number)
LED[2]	U19	LED	State bit 1
LED[3]	V19	LED	State bit 2 (MSB — only used if state > 3)
SEG[6:0]	W7-W6-U8-V8-U5-V5-U7	7-seg	Segment lines for all 4 digits



# Project Files

File	What it does
<code>seq_detect_1011_moore.sv</code>	The FSM itself — 5 states, next-state logic, Moore output
<code>debounce_onepulse.sv</code>	Button debouncer — prevents one press from being read many times
<code>seg7display.v</code>	7-segment display driver — multiplexes 4 digits at 100 MHz
<code>top.sv</code>	Top-level wiring — connects FSM, debouncer, display, LEDs
<code>seq_detect_tb.sv</code>	Testbench — simulates the FSM with a known bit stream (simulation source)
<code>lab3_basys3.xdc</code>	Constraints file — pins and voltages for the Basys 3 board

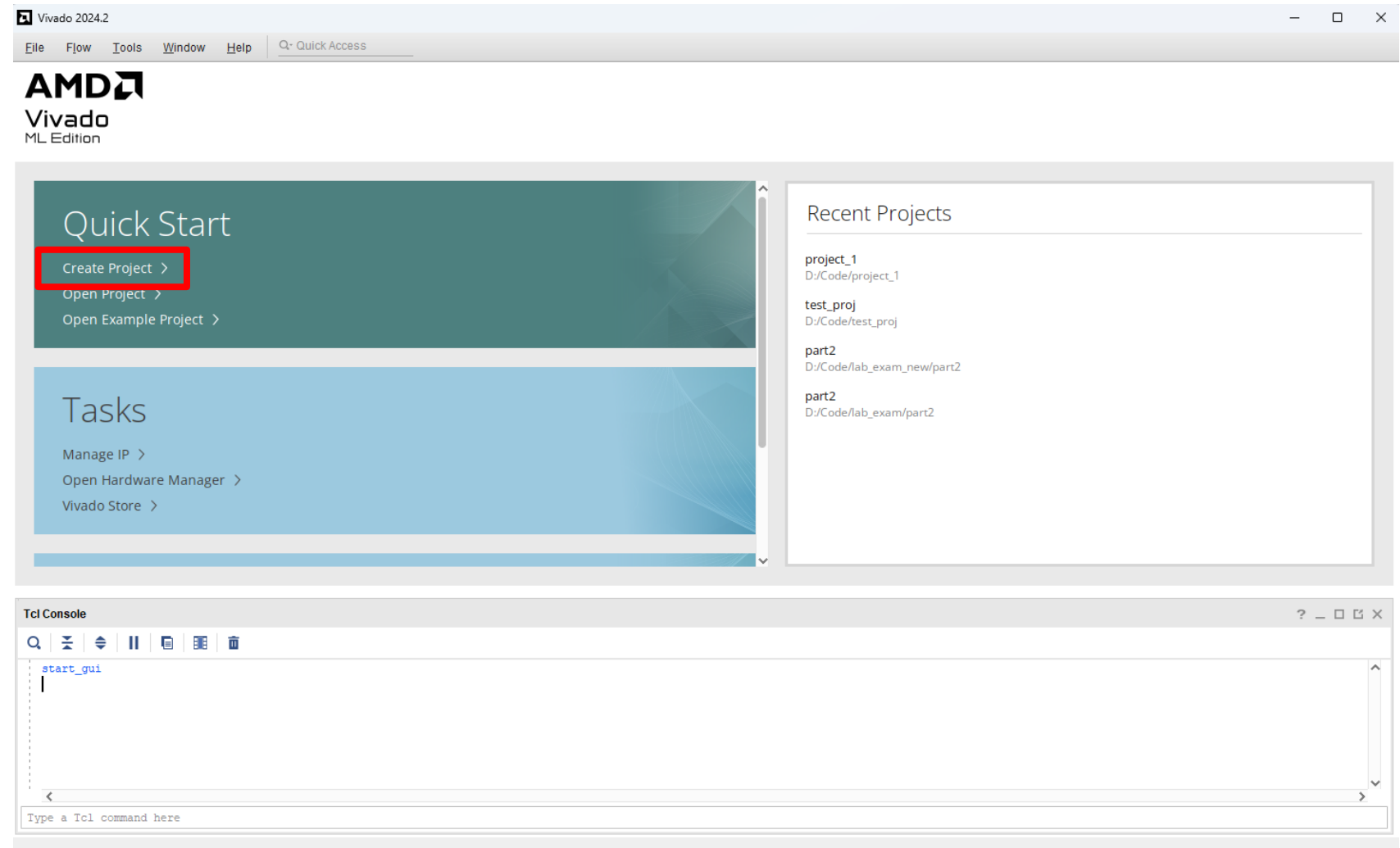
# FSM Template



*This three-block pattern is a universal template — you will use it for every clocked FSM you ever design.*

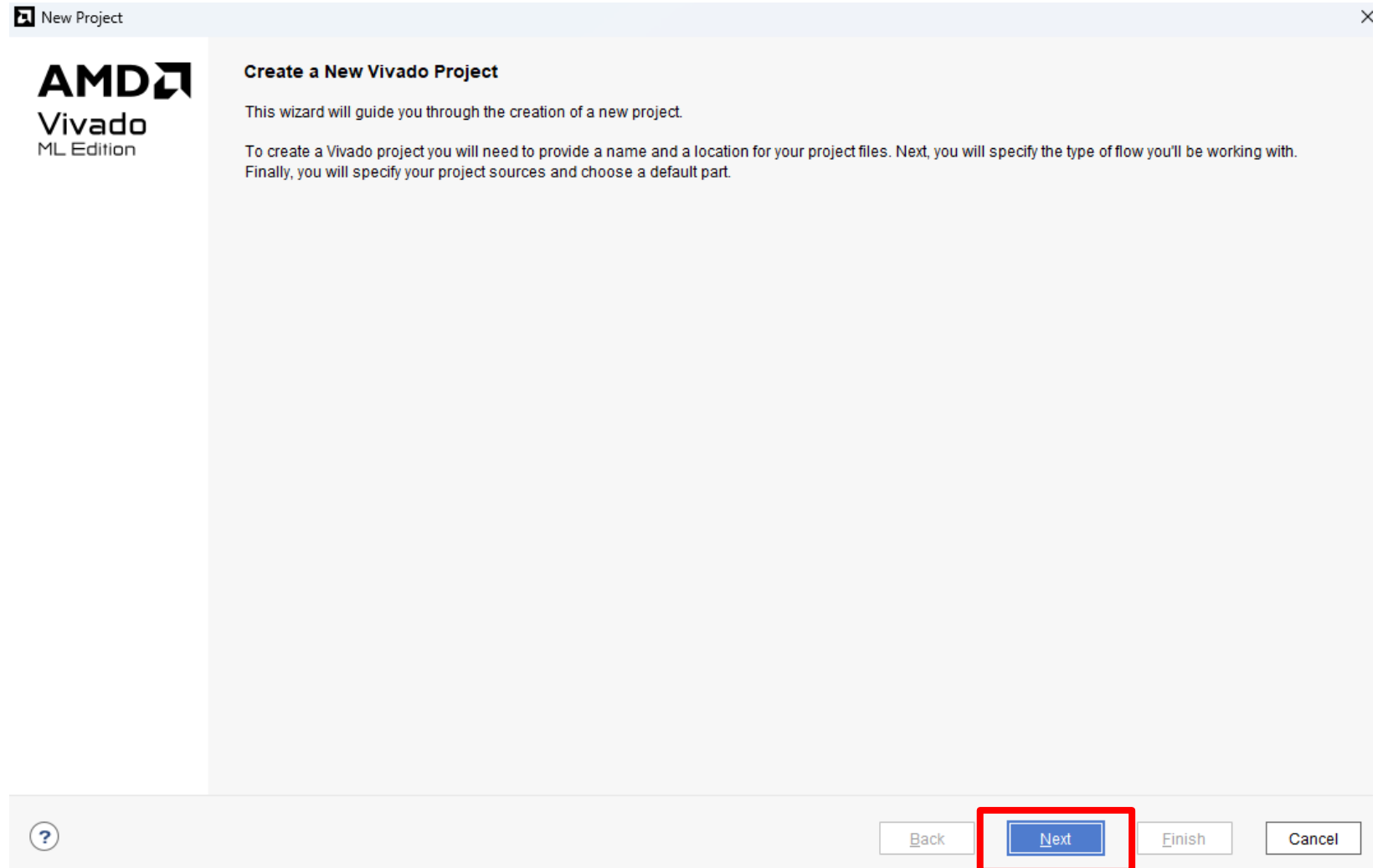
# Simulation – Start with VIVADO

- Start Vivado, click “Create Project”



# Simulation – Start with VIVADO

- ❑ In the New Project wizard, click “next”



# Simulation – Start with VIVADO

- ❑ In the New Project wizard, choose RTL project. Check “Do not specify sources at this time”.

The screenshot shows the 'New Project' dialog box in Vivado. The 'Project Type' section is active, with the following options:

- RTL Project**  
You will be able to add sources, create block designs in IP Integrator, generate IP, run RTL analysis, synthesis, implementation, design planning and analysis.
  - Do not specify sources at this time
  - Project is an extensible Vitis platform
- Post-synthesis Project**  
You will be able to add sources, view device resources, run design analysis, planning and implementation.
  - Do not specify sources at this time
- I/O Planning Project**  
Do not specify design sources. You will be able to view part/package resources.
- Imported Project**  
Create a Vivado project from a Synplify Project File.
- Example Project**  
Create a new Vivado project from a predefined template.

At the bottom of the dialog, there are four buttons: 'Back', 'Next', 'Finish', and 'Cancel'. The 'Next' button is highlighted with a red box.

# Simulation – Start with Vivado

- ❑ In the New Project wizard, choose **xc7a35tcpg236-1**. This is the FPGA device used on your BASYS3.
- ❑ Click next

**New Project**

**Default Part**  
Choose a default AMD part or board for your project.

Parts | Boards

[Reset All Filters](#)

Category: All Package: All Temperature: All  
Family: All Speed: All Static power: All

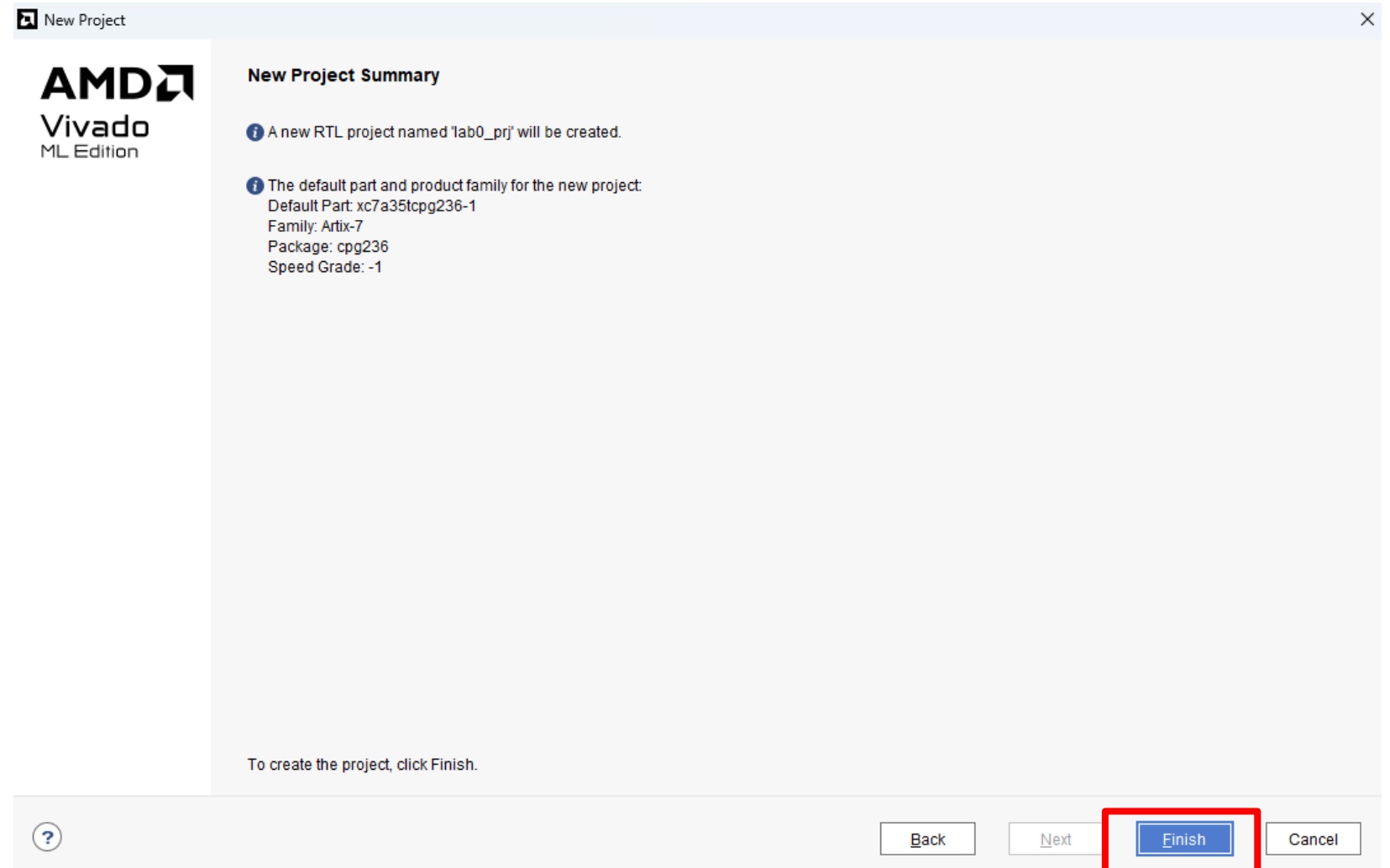
Search: Q-

Part	I/O Pin Count	Available IOBs	LUT Elements	FlipFlops	Block RAMs	Ultra RAMs	DSPs	BUFGs	Gb Transceivers	GTPE:
xc7a35tcpg236-3	236	106	20800	41600	50	0	90	32	2	2
xc7a35tcpg236-2	236	106	20800	41600	50	0	90	32	2	2
xc7a35tcpg236-2L	236	106	20800	41600	50	0	90	32	2	2
<b>xc7a35tcpg236-1</b>	<b>236</b>	<b>106</b>	<b>20800</b>	<b>41600</b>	<b>50</b>	<b>0</b>	<b>90</b>	<b>32</b>	<b>2</b>	<b>2</b>
xc7a35tcsfg324-3	324	210	20800	41600	50	0	90	32	0	0
xc7a35tcsfg324-2	324	210	20800	41600	50	0	90	32	0	0
xc7a35tcsfg324-2L	324	210	20800	41600	50	0	90	32	0	0
xc7a35tcsfg324-1	324	210	20800	41600	50	0	90	32	0	0
xc7a35tcsfg325-3	325	150	20800	41600	50	0	90	32	4	4
xc7a35tcsfg325-2	325	150	20800	41600	50	0	90	32	4	4
xc7a35tcsfg325-2L	325	150	20800	41600	50	0	90	32	4	4

Back **Next** Finish Cancel

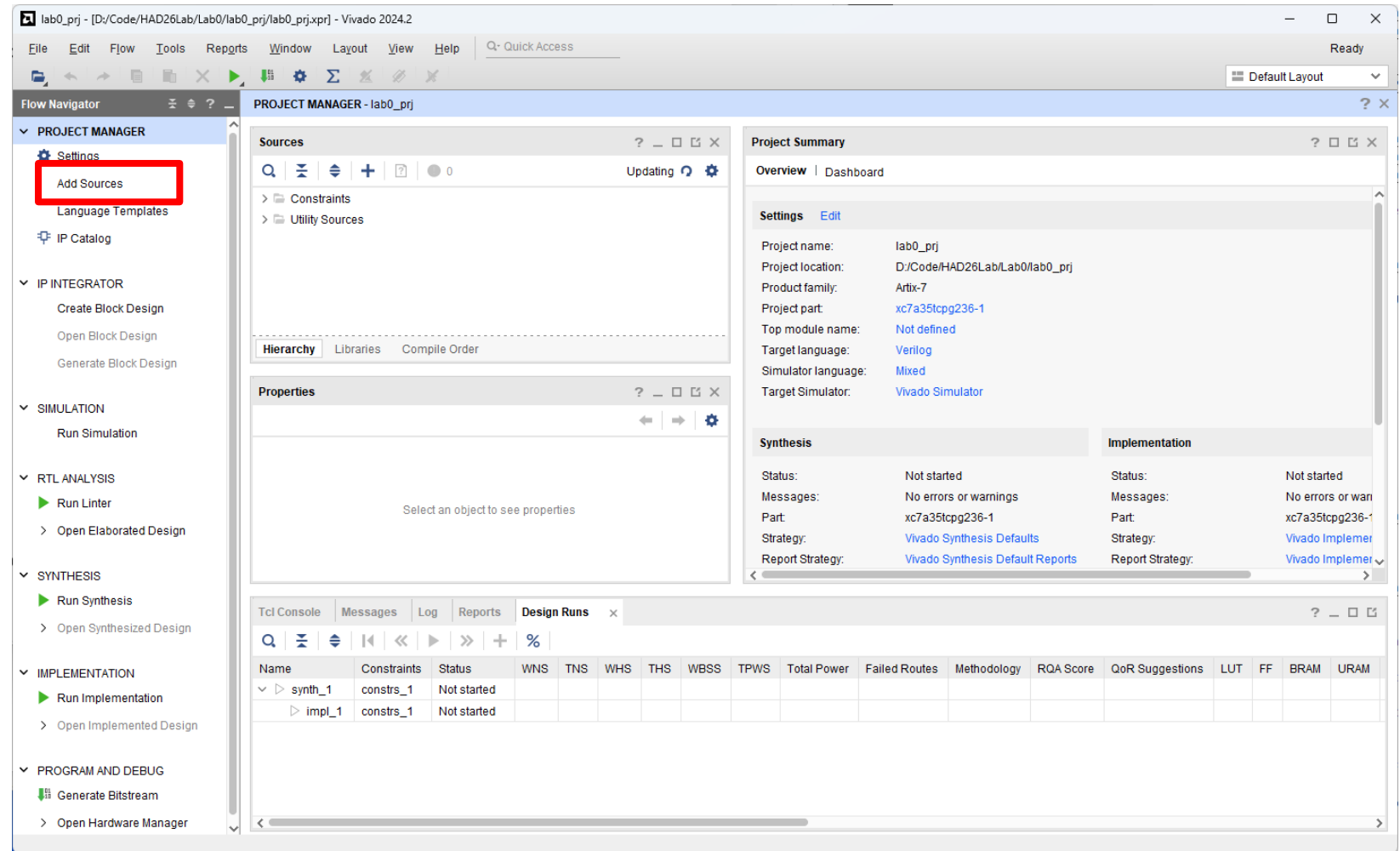
# Simulation – Start with VIVADO

- In the New Project wizard, click “Finish”.



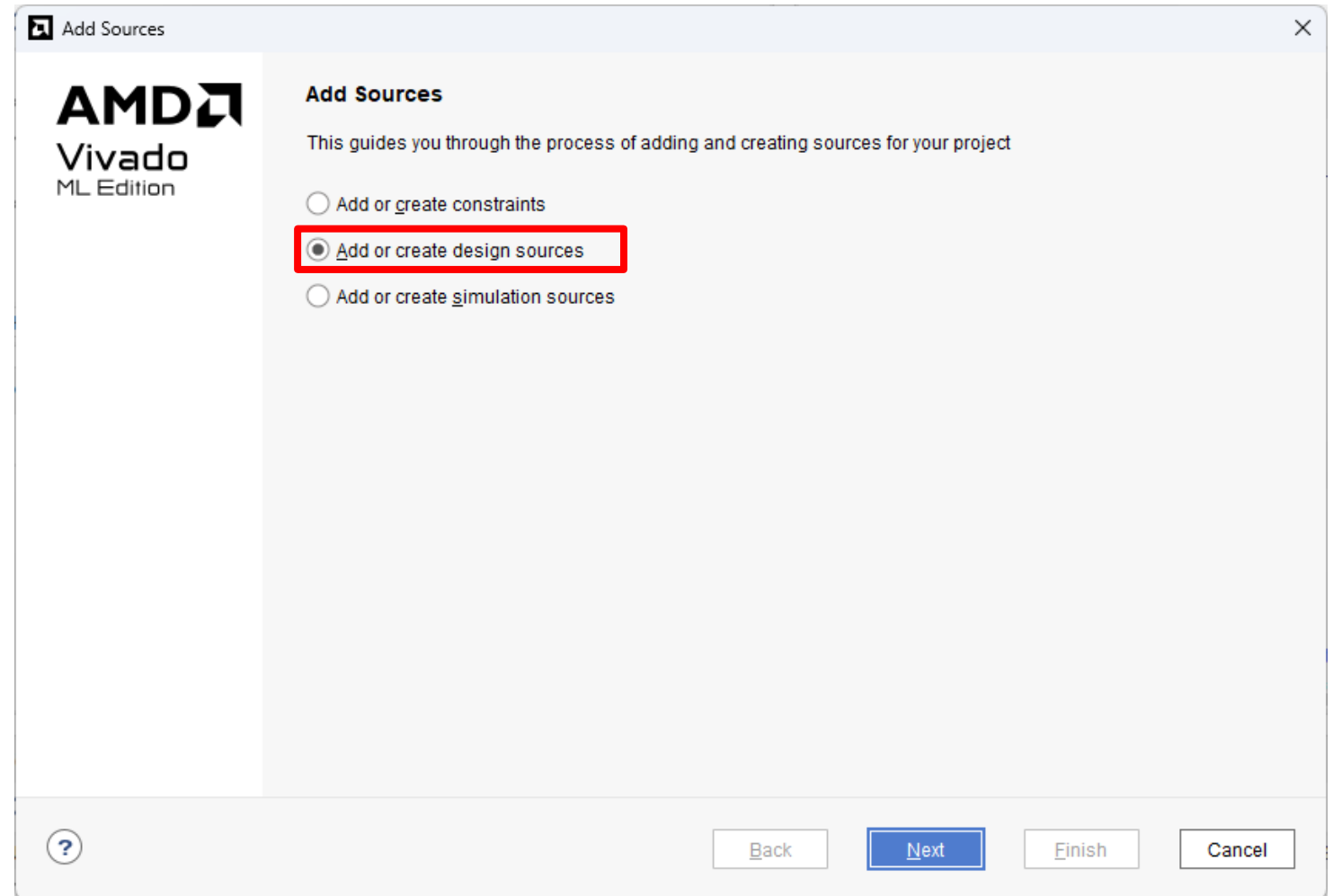
# Simulation – Start with Vivado.

- ❑ This is the workspace for your Vivado.
- ❑ On the left if your flow navigator, basically is the same workflow as our FPGA design workflow.
- ❑ Click add sources. I have provided the codes required for this lab. Import using “Add Sources”



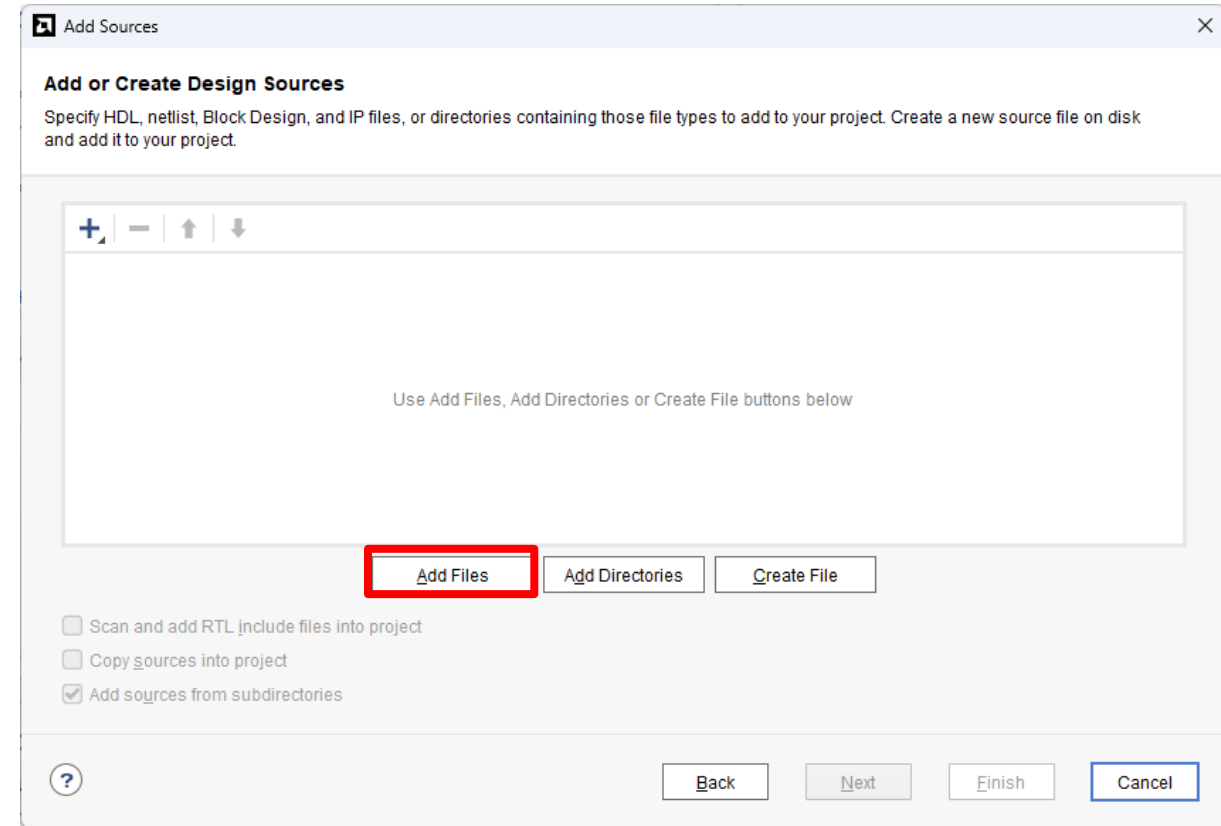
# Simulation – Start with VIVADO

- ❑ In the Add Sources wizard, choose “Add or create design sources”, then click next



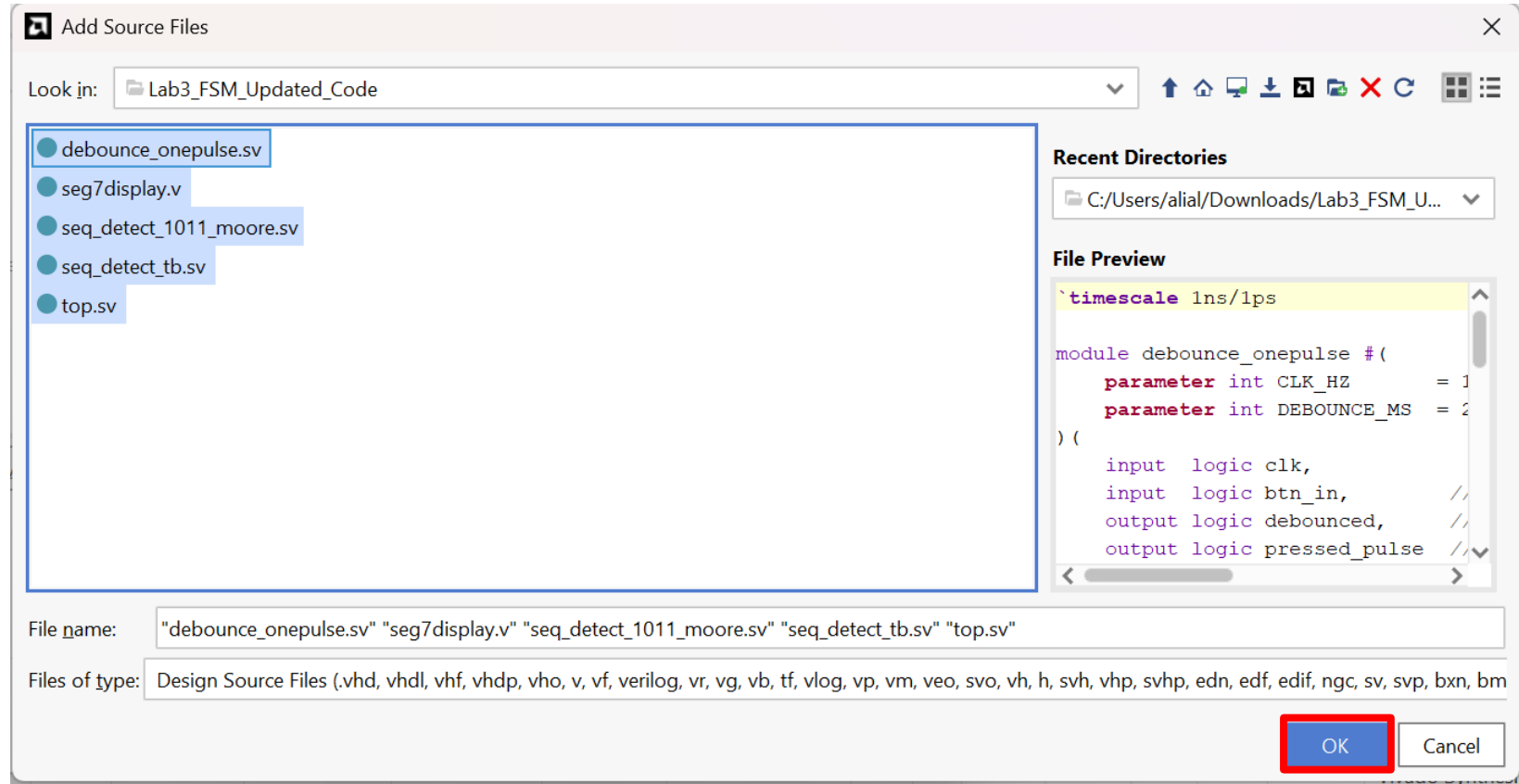
# Simulation – Start with VIVADO

- ❑ In the Add Sources wizard, click “Add files” to import the source files.



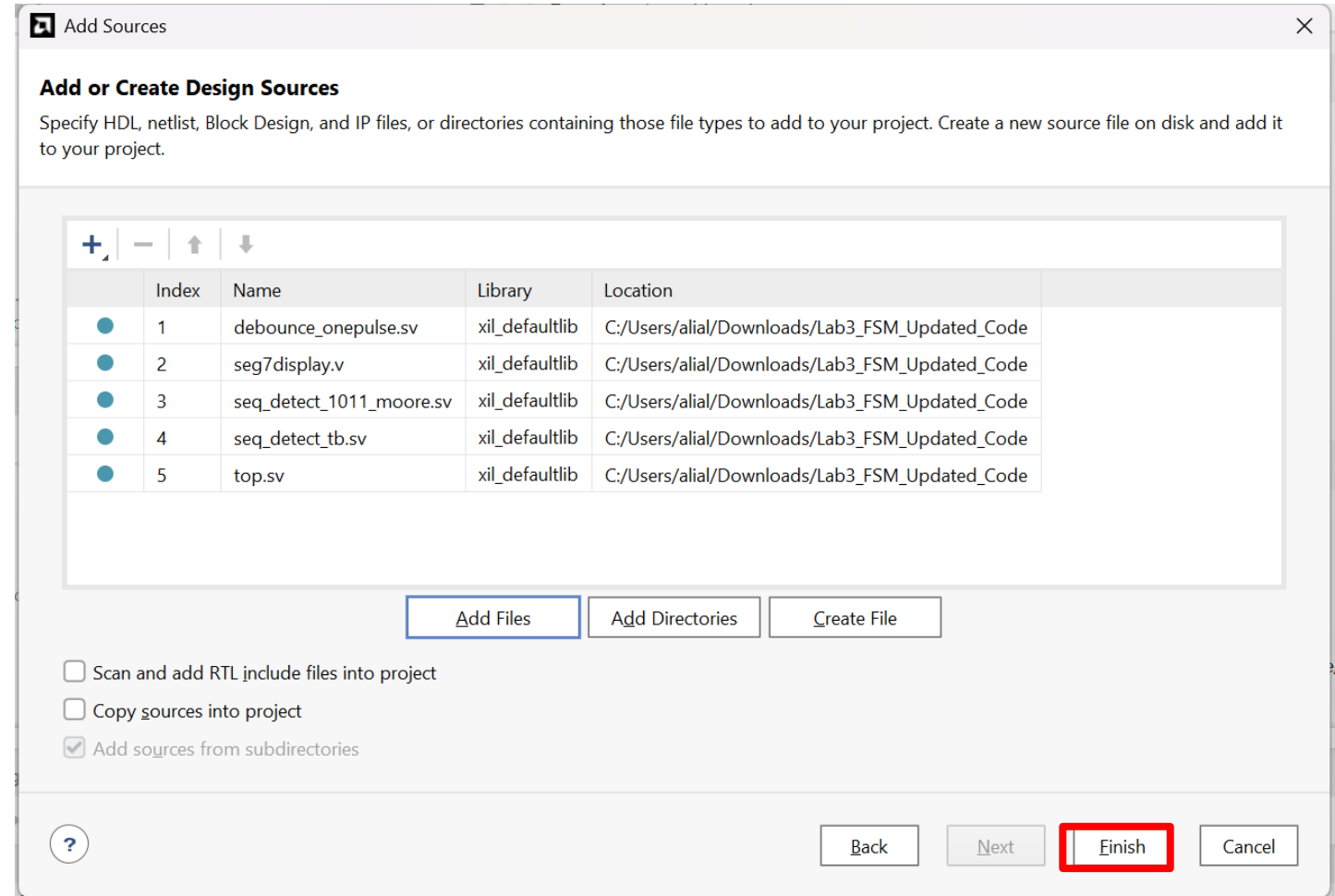
# Simulation – Start with VIVADO

- ❑ In the Add Source File wizard, navigate to the source files, and choose all the provided source files.
- ❑ Then click OK



# Simulation – Start with VIVADO

- ❑ In the Add Source wizard, you will see the imported source files.
- ❑ Then click Finish



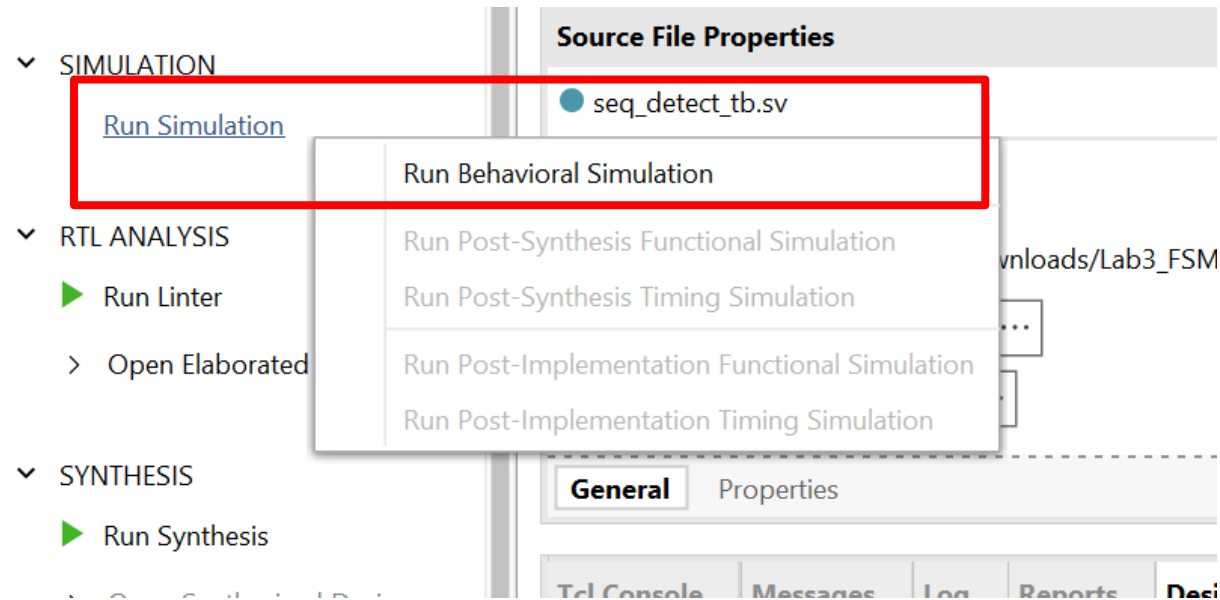
# Simulation – Starting with Vivado

- ❑ In the Vivado workspace, you will see the imported modules in the “Source” window.
- ❑ The imported sources are automatically ordered in the “hierarchy” tab.
- ❑ Lets first run the simulation, find seq\_detect\_tb.sv, right click and set as top file.

The screenshot displays the Vivado IDE interface. The 'Sources' window is open, showing a tree view of project sources. Under 'Simulation Sources', the file 'seq\_detect\_tb (seq\_detect\_tb.sv)' is selected. A context menu is open over this file, with the 'Set as Top' option highlighted in red. Below the Sources window, the 'Source File Properties' dialog is visible, showing the file 'seq\_detect\_tb.sv' with properties: Enabled, Location: C:/Users/alial/Downloads/Lal, Type: SystemVerilog, and Library: xil\_defaultlib. The 'General' tab is active in the properties dialog.

# Simulation – Starting with Vivado Project

- Under the simulation tab, click Run Simulation, then Run Behavioral Simulation



# Simulation – Starting with Vivado Project

- ❑ Then you will enter the simulation window. Click “Untitled 1” to enter the waveform window.
- ❑ Note: The waveform window name changes as you run multiple times of simulations.

The screenshot displays the Vivado 2024.2 interface during a behavioral simulation. The main window shows the simulation results for a 4-bit full adder. The left sidebar contains the Project Manager, IP Integrator, and Simulation sections. The central pane shows the Scope and Objects tables, and the right pane shows the Verilog code for the full\_adder\_4bit\_tb.v module. The Tcl Console at the bottom displays the simulation output.

Name	Value	Data Type
a[3:0]	f	Array
b[3:0]	f	Array
cin	1	Logic
sum[3:0]	f	Array
cout	1	Logic
expected_sum[3:0]	f	Array
expected_cout	1	Logic

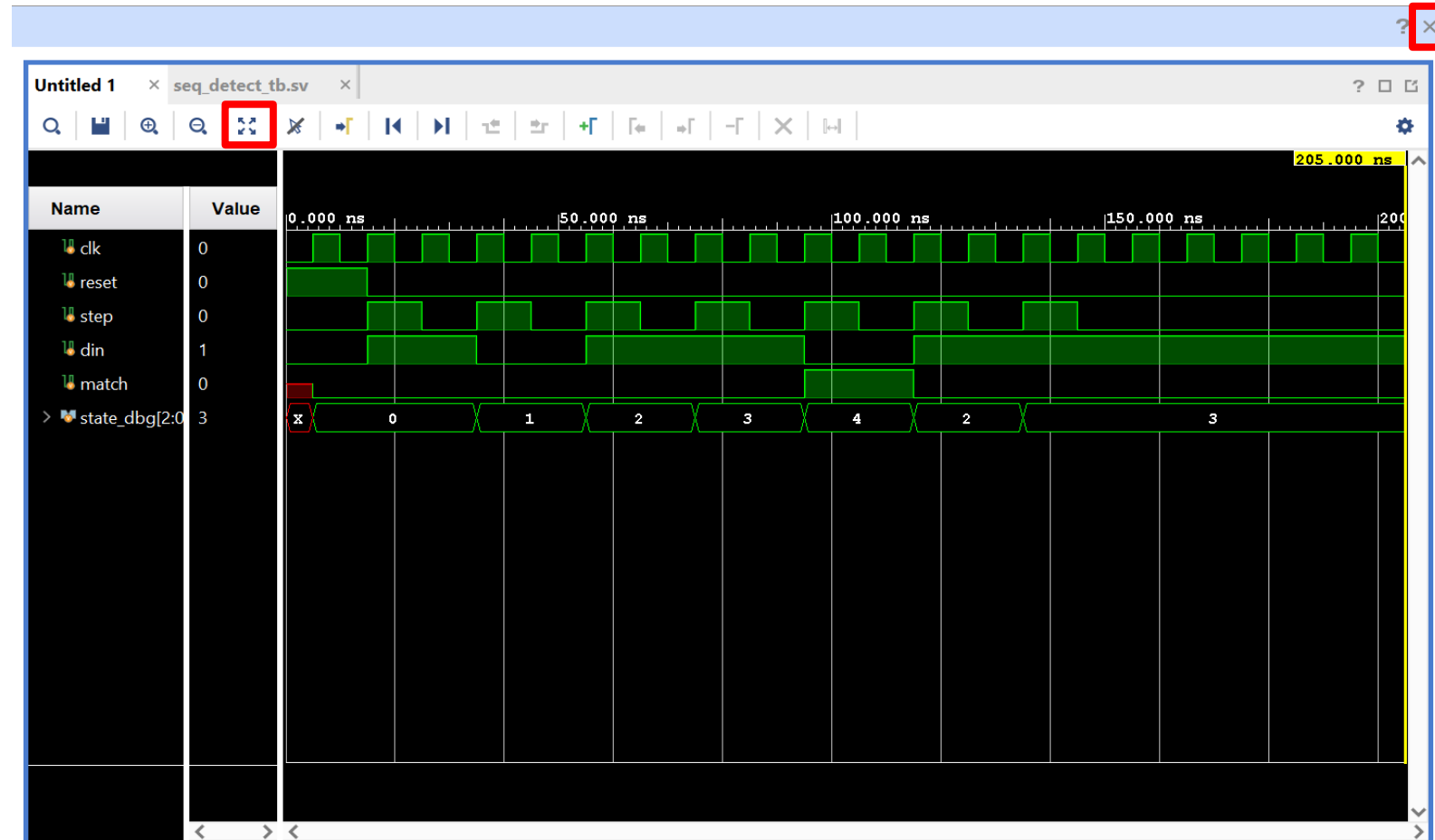
```
15
16 wire [3:0] expected_sum;
17 wire      expected_cout;
18
19 assign [expected_cout, expected_sum] = a + b + cin;
20
21 // Instantiate the 4-bit full adder module
22 full_adder_4bit dut (.a(a),.b(b),.cin(cin),.sum(sum),.cout(cout));
23
24 // Test procedure
25 initial begin
26 // Monitor changes
27 $monitor("Time=%0t | a=%b b=%b cin=%b | sum=%b cout=%b", $time, a, b, cin, sum, cout);
28
29 // Test various combinations of inputs
30 a = 4'b0000; b = 4'b0000; cin = 0; #10;
31 a = 4'b0001; b = 4'b0010; cin = 0; #10;
32 a = 4'b0011; b = 4'b0101; cin = 1; #10;
33 a = 4'b1111; b = 4'b0001; cin = 0; #10;
34 a = 4'b1010; b = 4'b0101; cin = 1; #10;
35 a = 4'b1111; b = 4'b1111; cin = 1; #10;
36
37 // Finish simulation
38 $finish;
39 end
40
41
42 endmodule
43
```

Tcl Console Output:

```
Time=50000 | a=1111 b=1111 cin=1 | sum=1111 cout=1
$finish called at time : 60 ns : File "D:/Code/HAD26Lab/Lab0/src/full_adder_4bit_tb.v" Line 38
INFO: [USF-XSim-96] XSim completed. Design snapshot 'full_adder_4bit_tb_behav' loaded.
INFO: [USF-XSim-97] XSim simulation ran for 1000ns
```

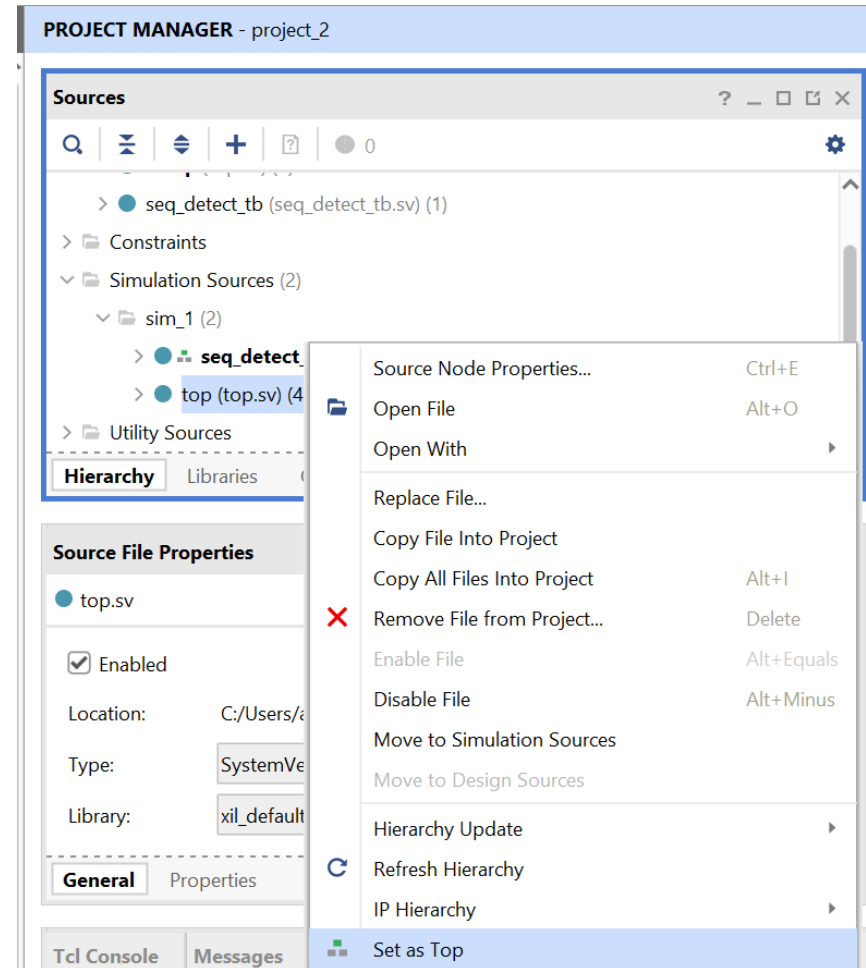
# Simulation – Starting with Vivado Project

- ❑ Click “Zoom Fit” to show the complete waveform. You will see all the interesting signals.
- ❑ Once you confirm that your signal is OK, click “x” on the top-right corner of the “SIMULATOR” window to quick simulation.



# Change TOP File

- ❑ After you've double checked your simulation, you can deploy on the FPGA.
- ❑ Change the Top file back to top.sv

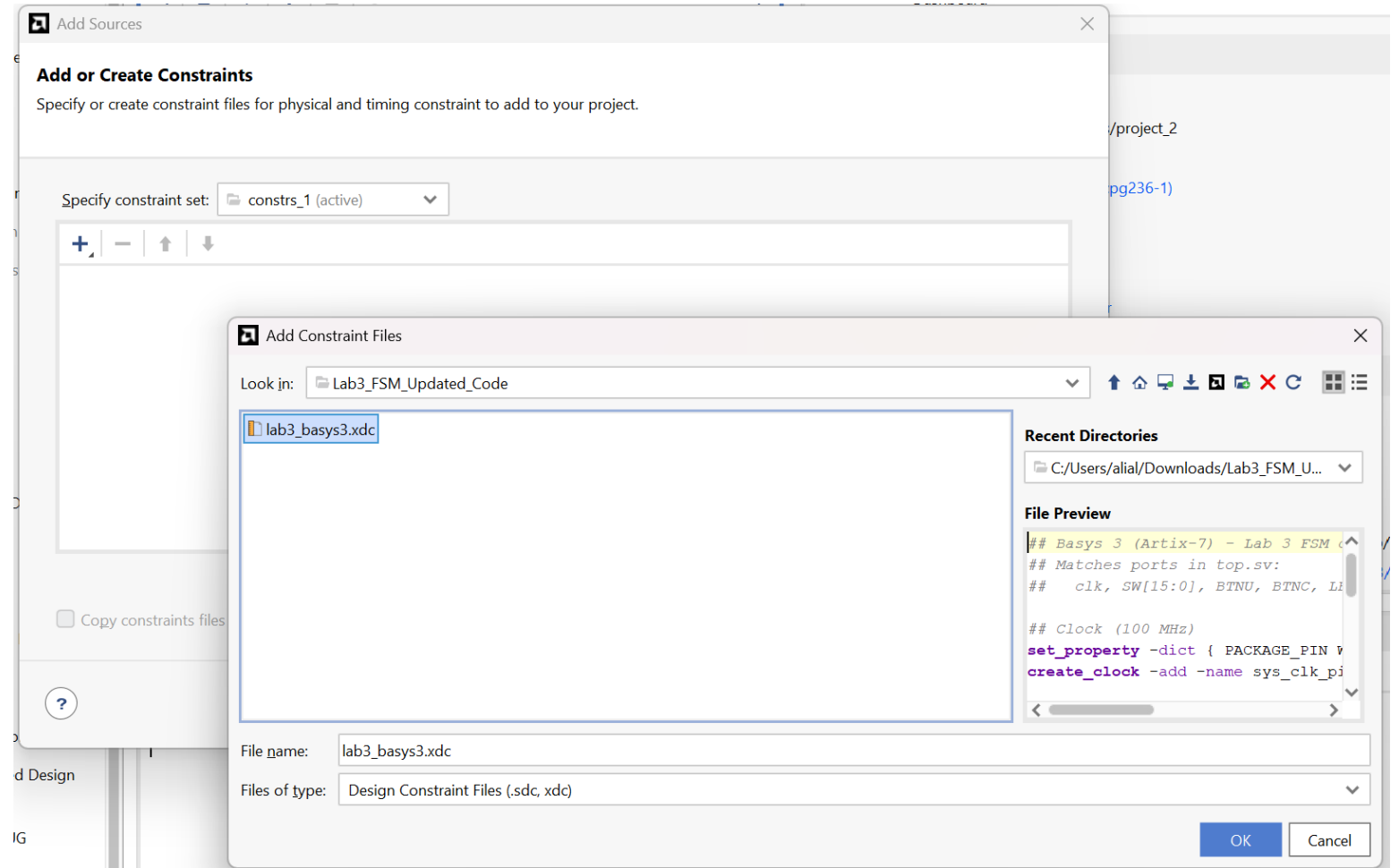


# Add Constraints

The image shows the Vivado ML Edition interface. On the left, the 'Project Manager' pane is visible, with the 'Add Sources' option under the 'PROJECT MANAGER' section highlighted with a red rectangle. The main window is the 'Add Sources' dialog box, which contains the AMD Vivado ML Edition logo and the title 'Add Sources'. Below the logo, the text reads: 'This guides you through the process of adding and creating sources for your project'. There are three radio button options: 'Add or create constraints' (selected and highlighted with a red rectangle), 'Add or create design sources', and 'Add or create simulation sources'. At the bottom of the dialog, there are four buttons: a help button (question mark), 'Back', 'Next' (highlighted in blue), and 'Cancel'.

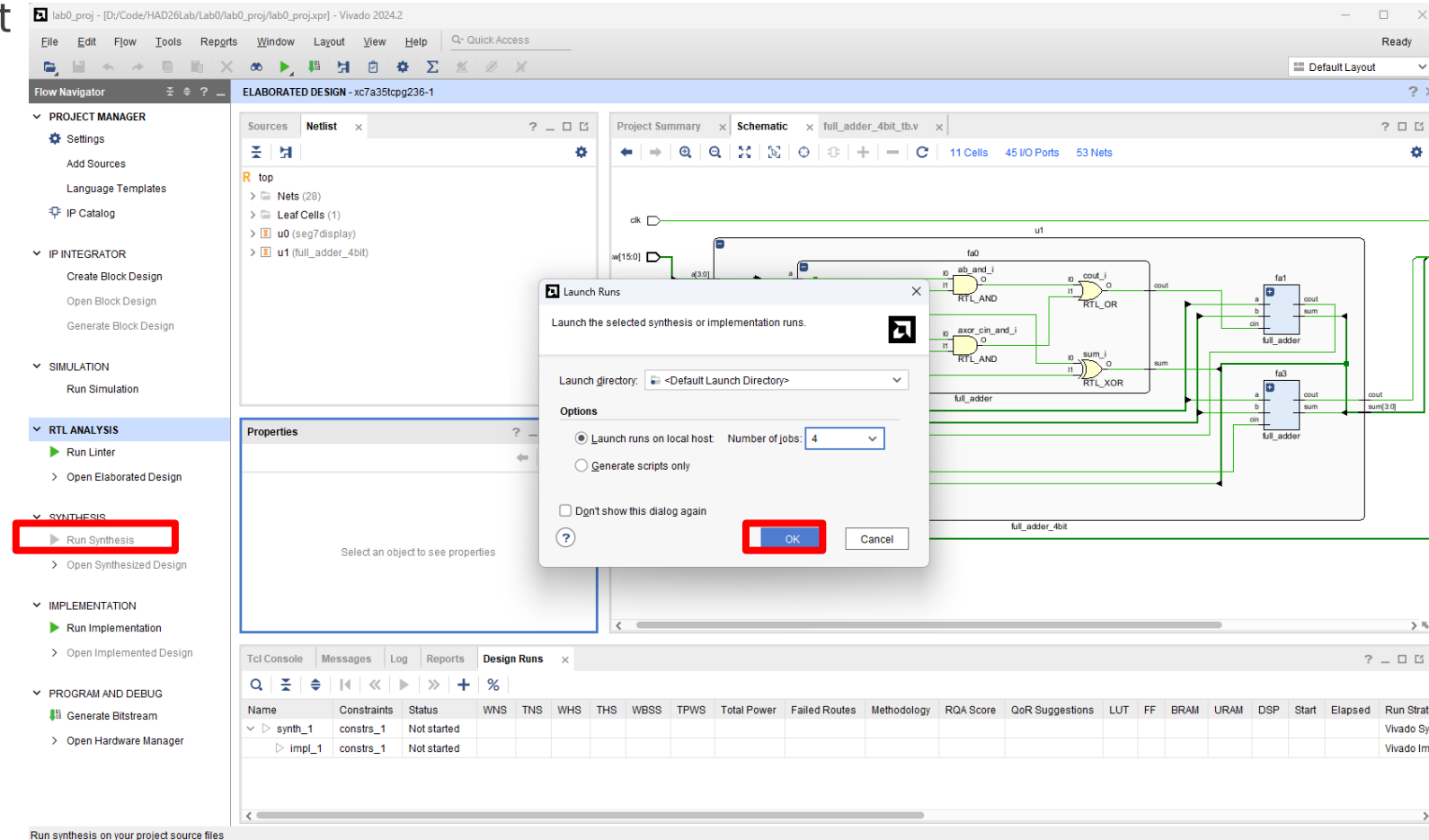
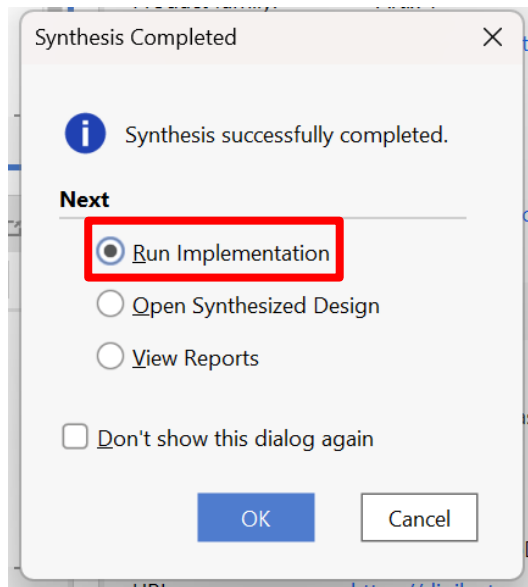
# Basys3 XDC

The XDC file (constraints) is **the translation layer between your signal names and the physical pins** on the chip. Without it, Vivado doesn't know that `sw[0]` means "the leftmost switch on pin V17", it's just a name.



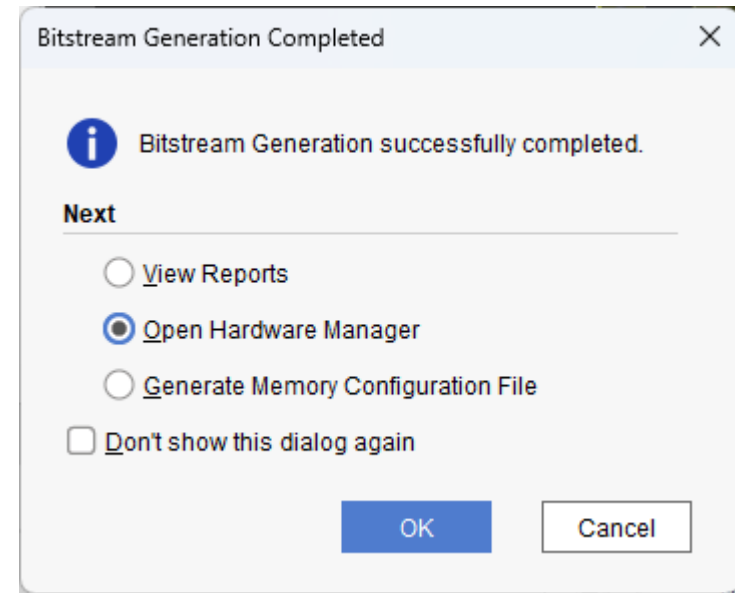
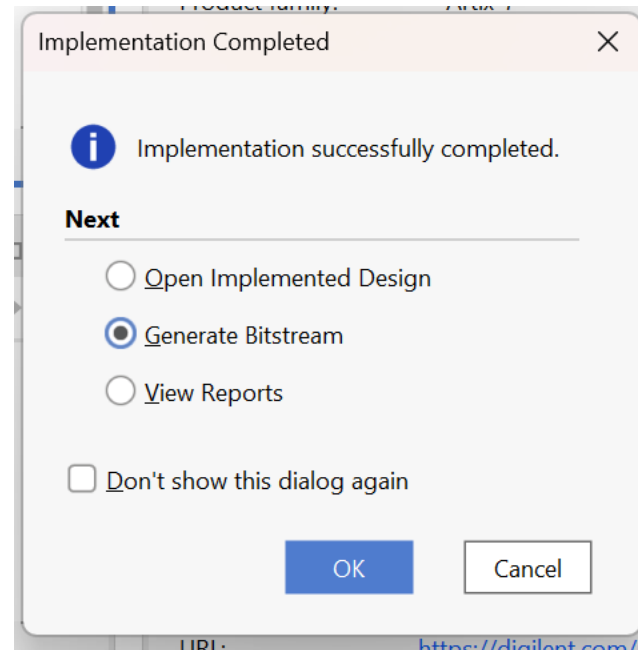
# Starting with Vivado Project - Synthesis

- ❑ In the Flow Navigator, click “Run Synthesis”, then click OK to start synthesis. This process takes a while.
- ❑ When completed, choose “Run Implementation”.



# Starting with Vivado Project - Programming

- ❑ After implementation, you can generate bitstream.
- ❑ Choose “Open Hardware Manager” when finished.



# Starting with Vivado Project - Programming

- ❑ After generating the bitstream, you are ready to program on FPGA.
- ❑ Choose “Open Hardware Manager” when finished. Connect BASYS3 with your Windows Laptop.
- ❑ In the hardware manager, click *Open Target* -> *Auto Connect*
- ❑ Then click *Program Device* -> *Program*

