KING'S College LONDON

# 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

# Learning Outcomes

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

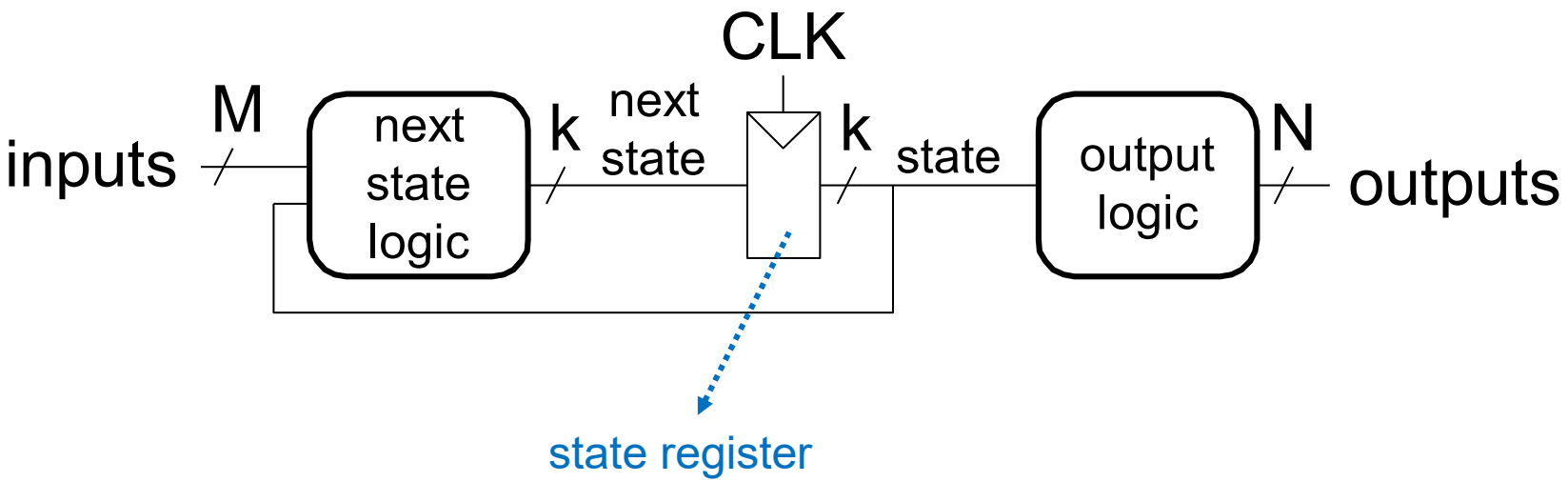
- State diagrams let you reason about behaviour before writing a single line of code.

# 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)

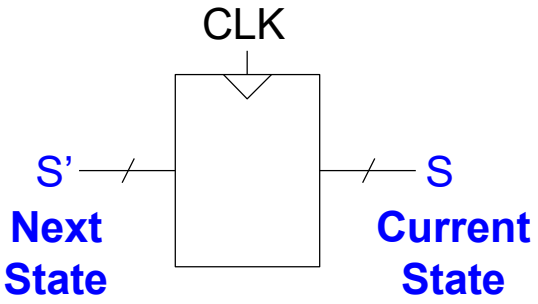
- Each FSM consists of three separate parts:
  - next state logic
  - state register
  - output logic

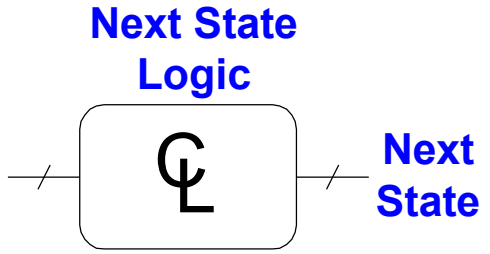


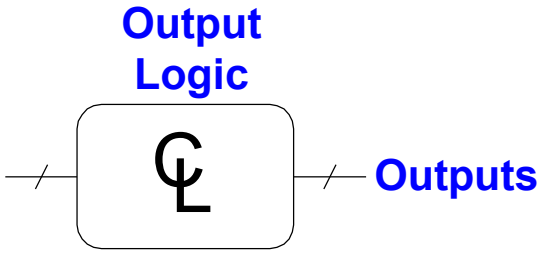

At the beginning of the clock cycle, the next state is latched into the state register

# 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



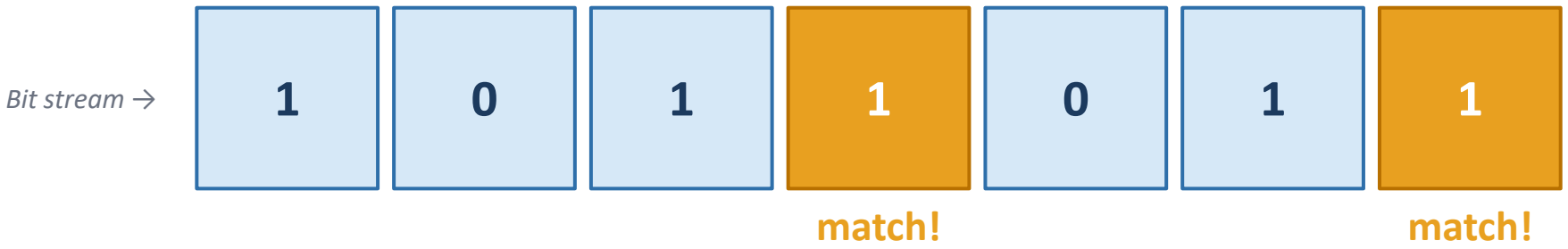



# Part 2

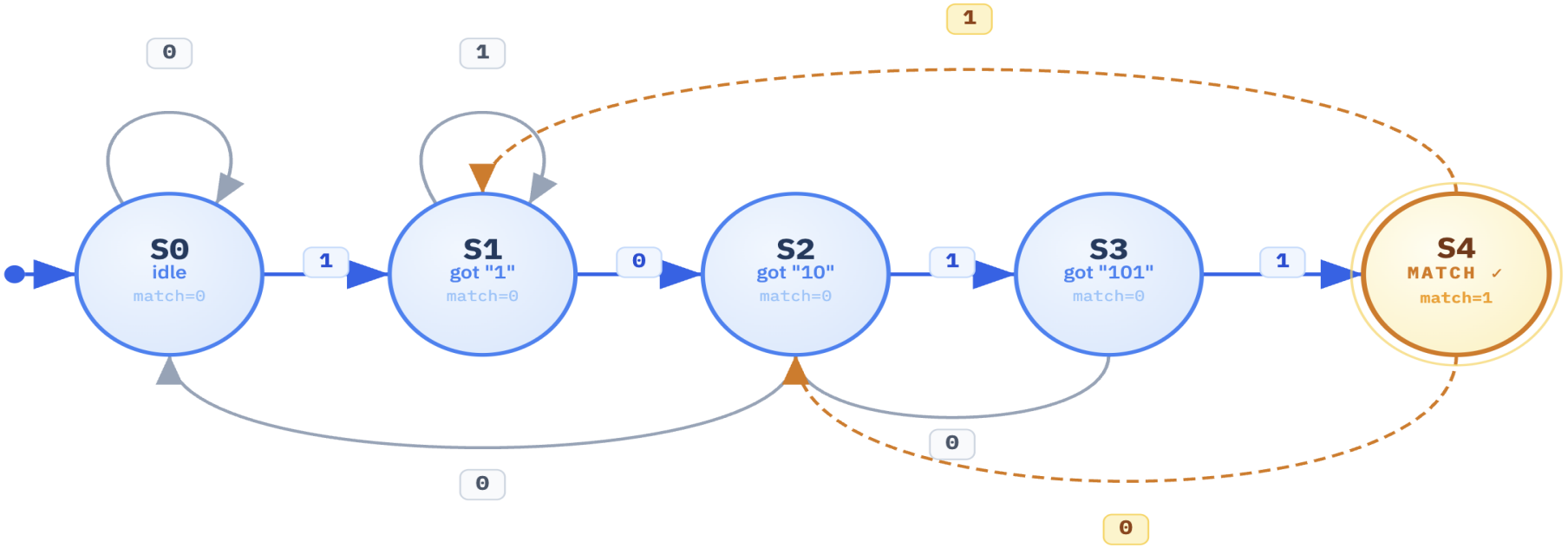
LAB 3: LAB TASK – RUNNING AN FSM ON BASYS3

# 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 |
| **S4 (MATCH)** | S2 | S1 | **1 ✓** |

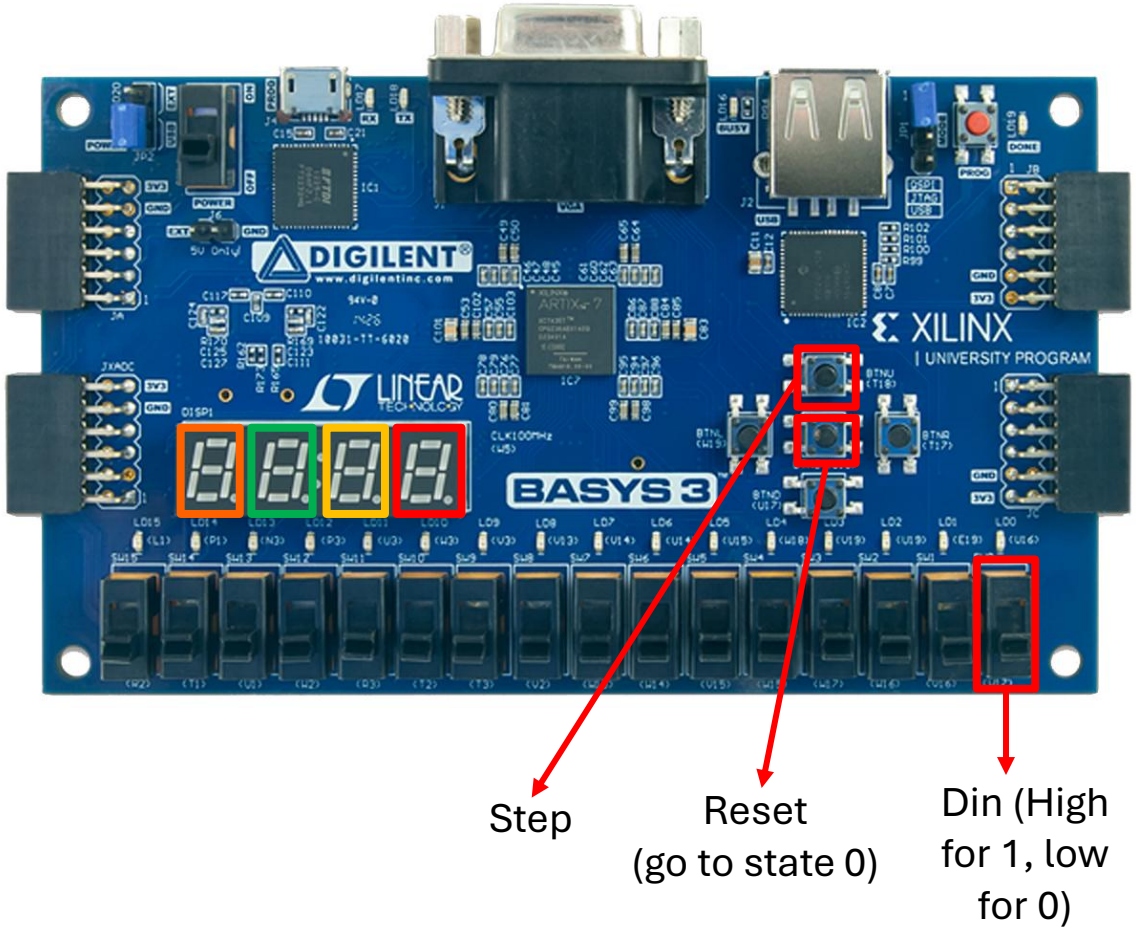
# Background: FSM Example

- 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 |
|---|---|---|---|---|---|
| ✓ Correct sequence: 1 → 0 → 1 → 1 (pattern 1011 detected) | | | | | |
| 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 |
| ✗ Incorrect sequence: 1 → 1 → 0 → 0 (no match — returns to idle) | | | | | |
| 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 — idle (no match) | 0 |

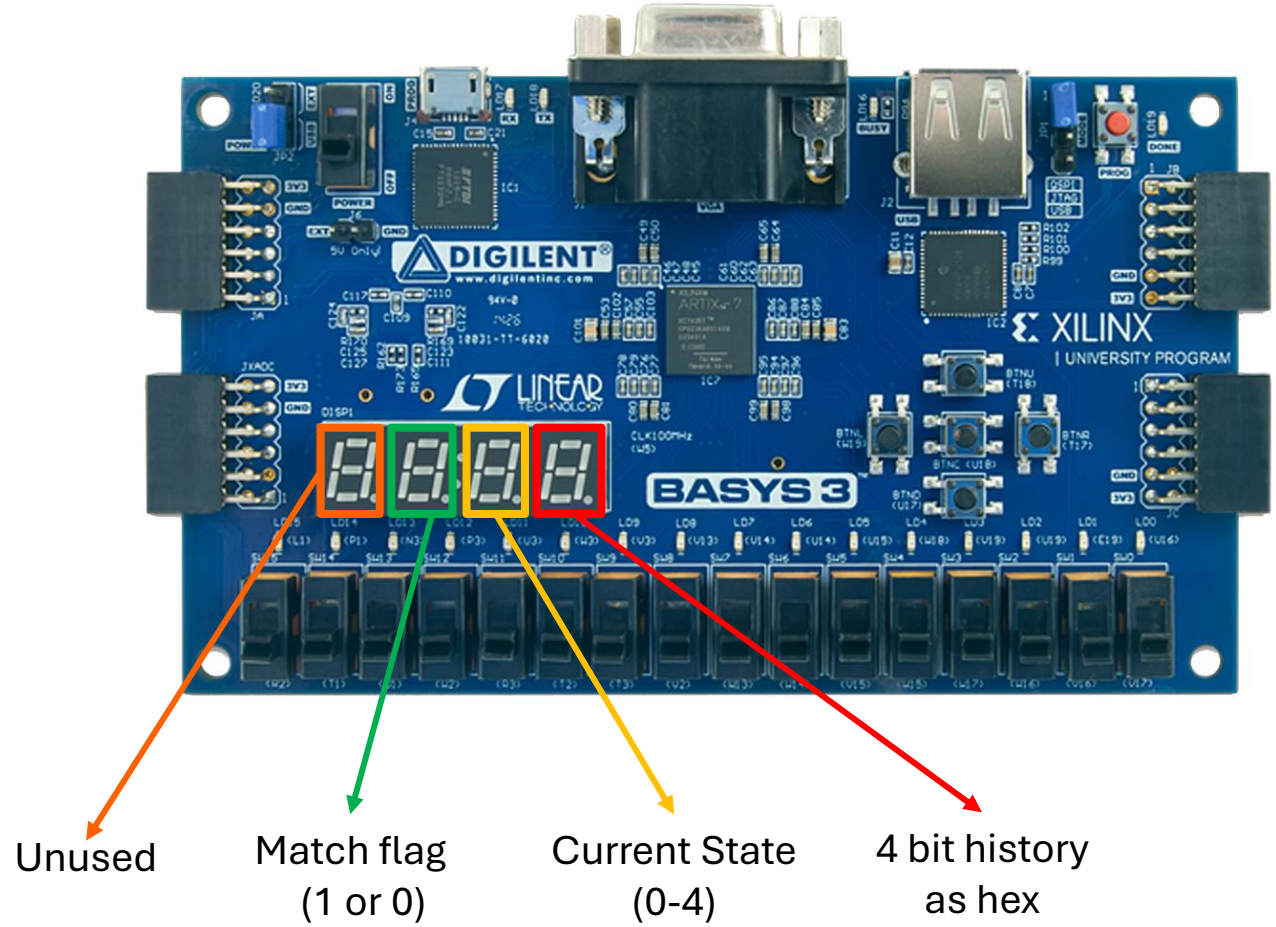
# FPGA Inputs

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

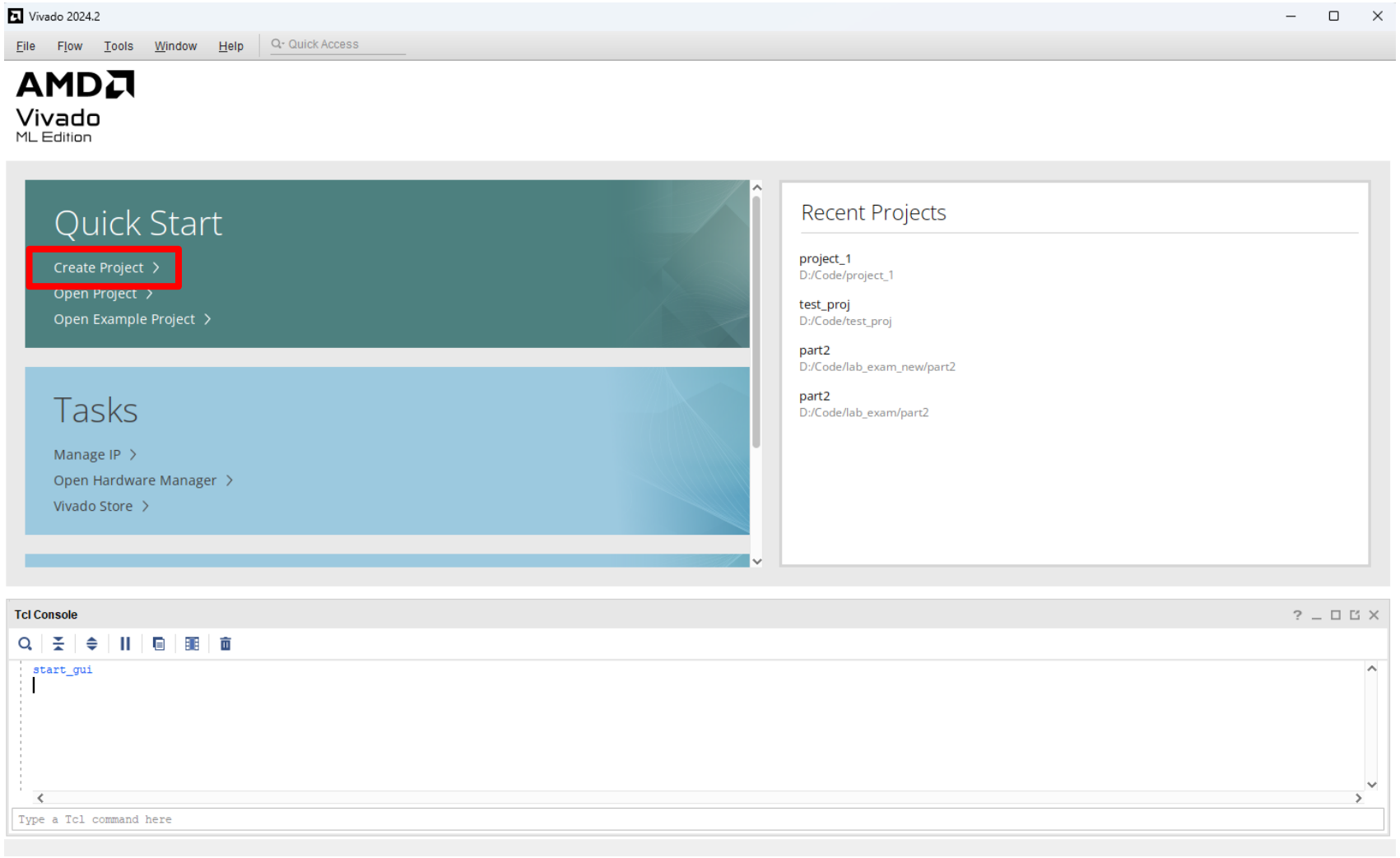
# FSM Template

| Block 1 | Block 2 | Block 3 |
| --- | --- | --- |
| **State Register (sequential)** | **Next-State Logic (combinational)** | **Output Logic (combinational)** |
| always_ff — updates state on the rising clock edge.<br>Only moves to next_state when step = 1.<br>Synchronous reset returns to S0. | always_comb — computes where we go next.<br>A case statement checks current state and din.<br>No clock — runs continuously. | always_comb — sets match = 1 only when state == S4.<br>Moore: output depends on state only, not din.<br>Clean, glitch-free output. |

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

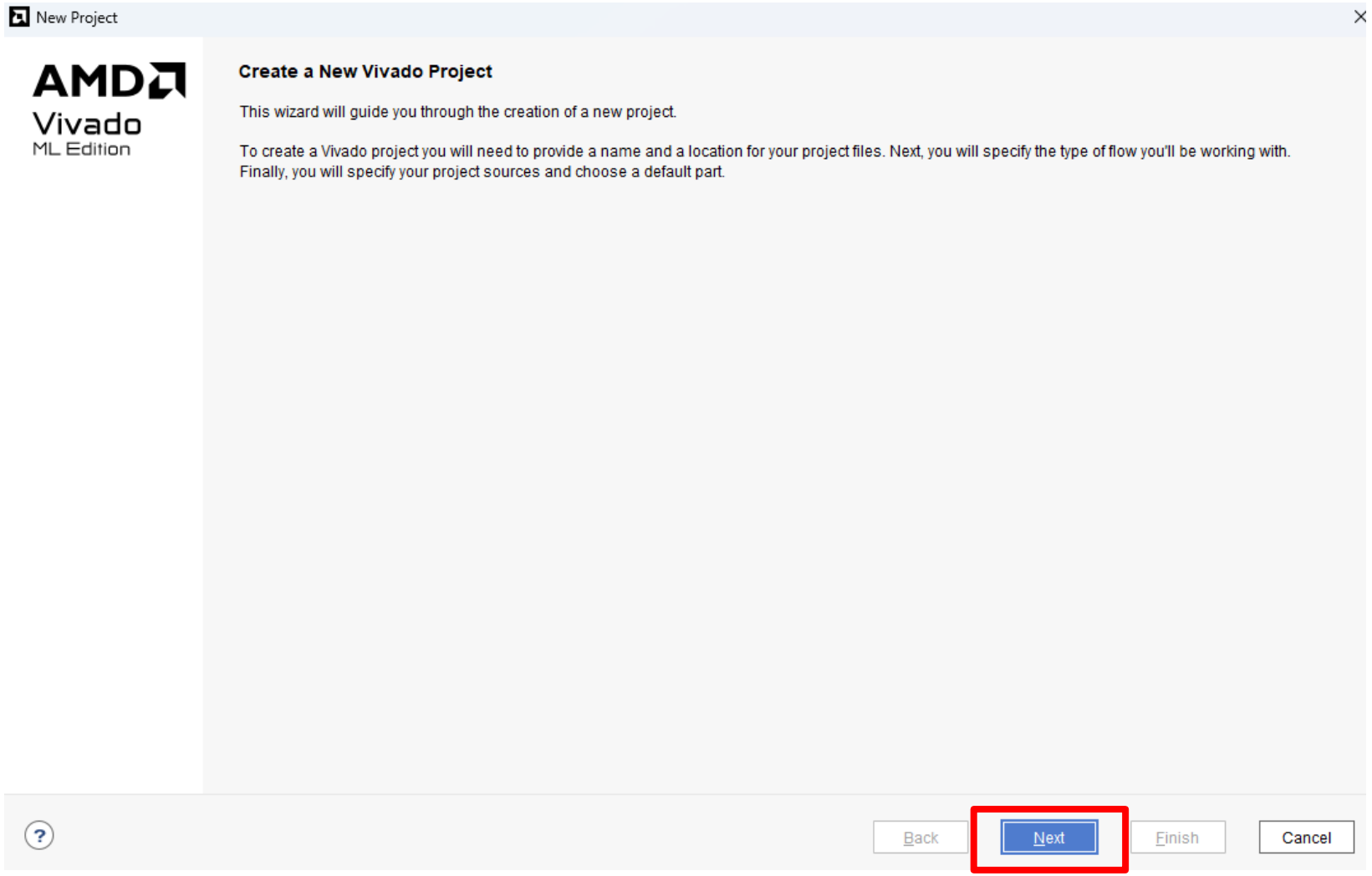
# Simulation – Start with VIVADO

- Start Vivado, click “Create Project”



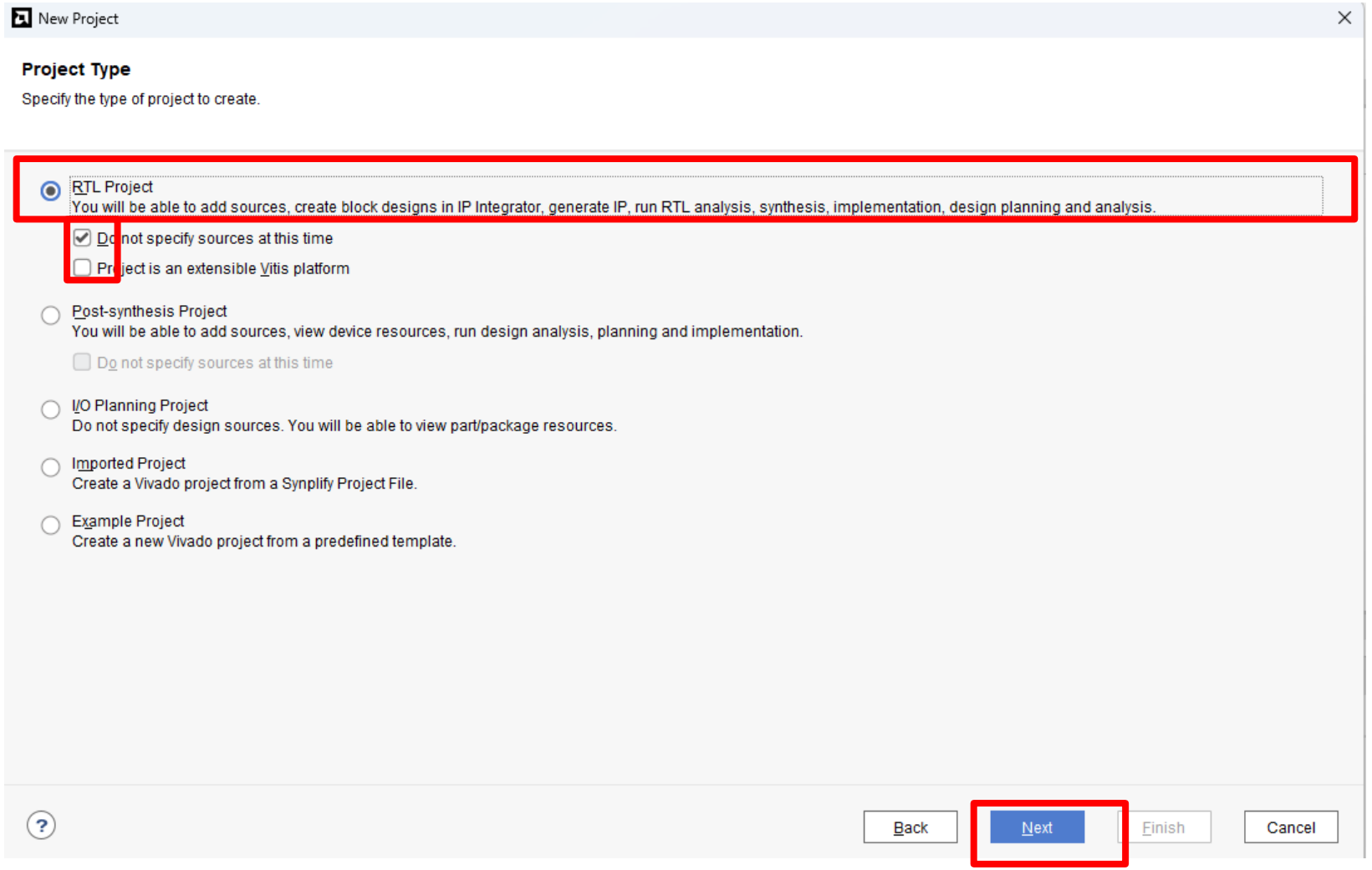
# Simulation – Start with VIVADO

- In the New Project wizard, click “next”



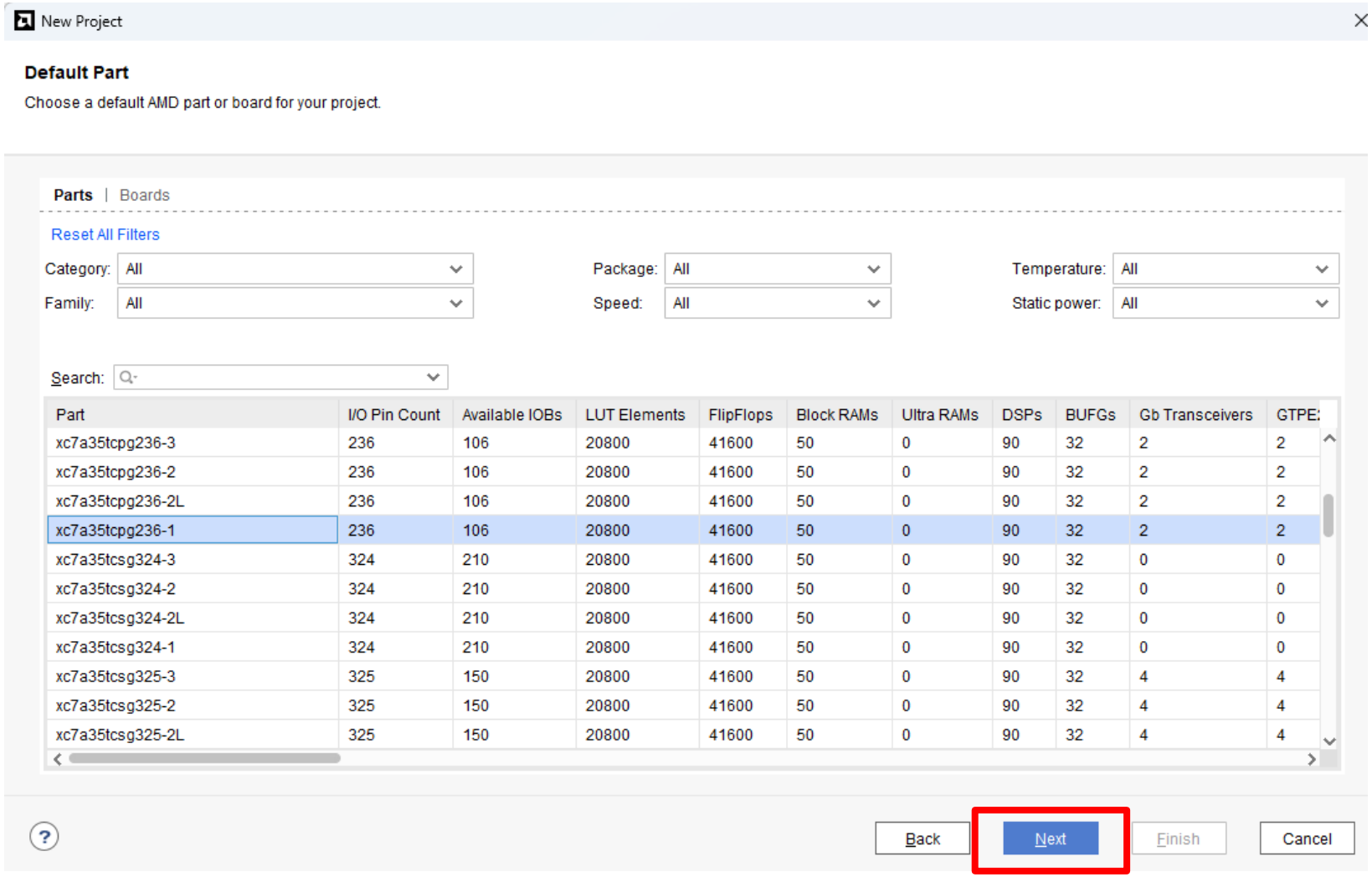
# Simulation – Start with VIVADO

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



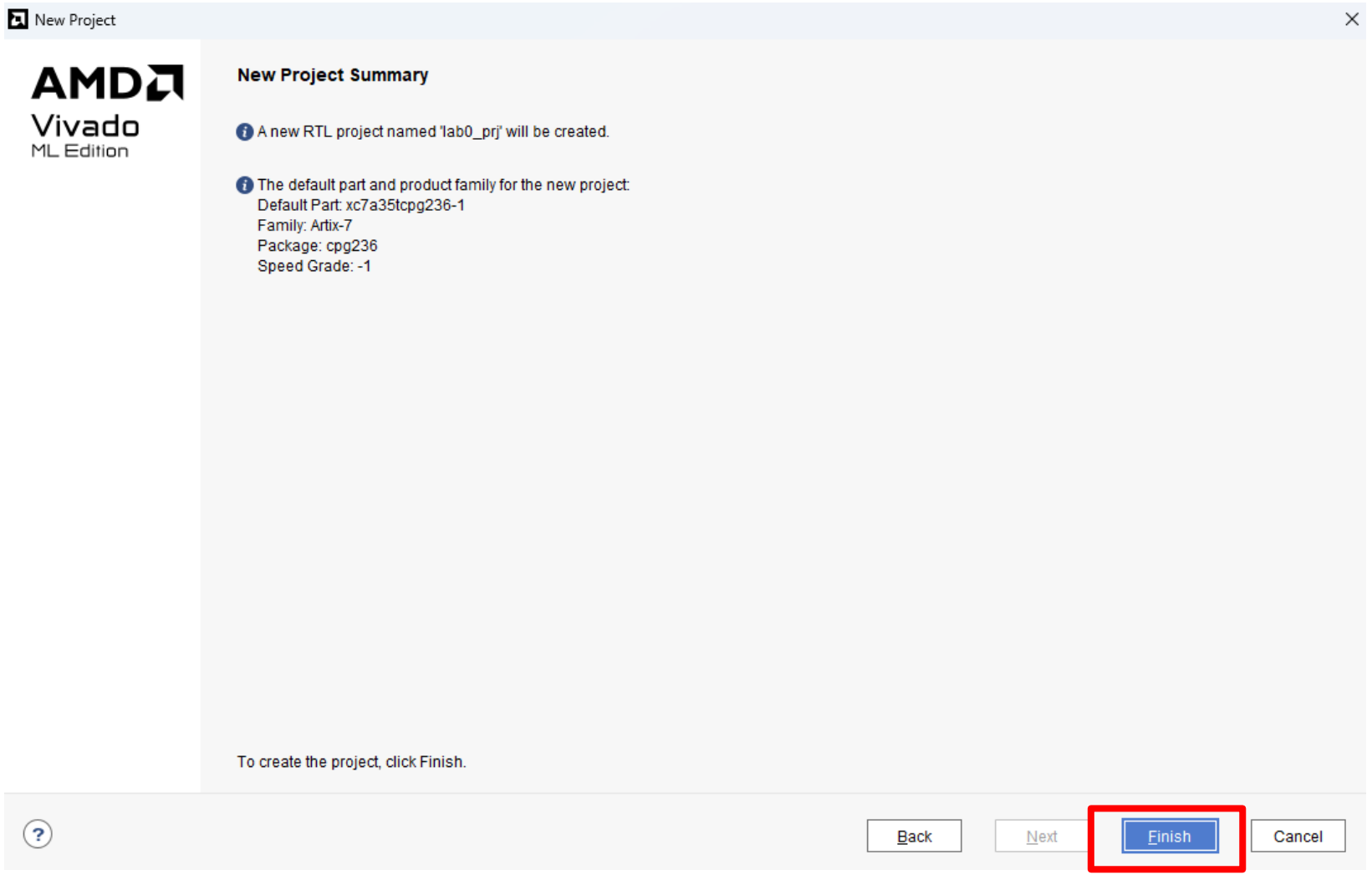
# Simulation – Start with Vivado

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



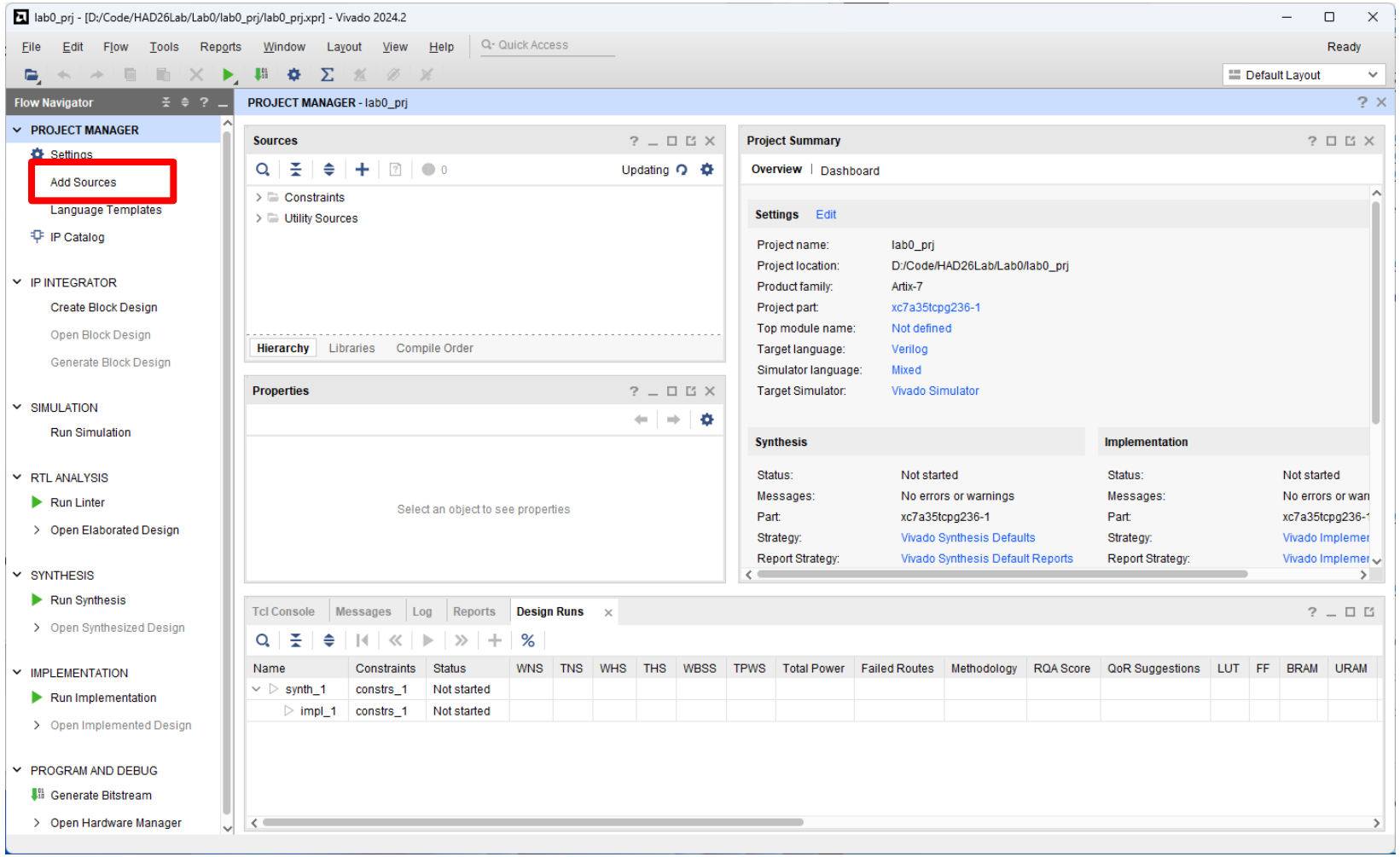
# Simulation – Start with VIVADO

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



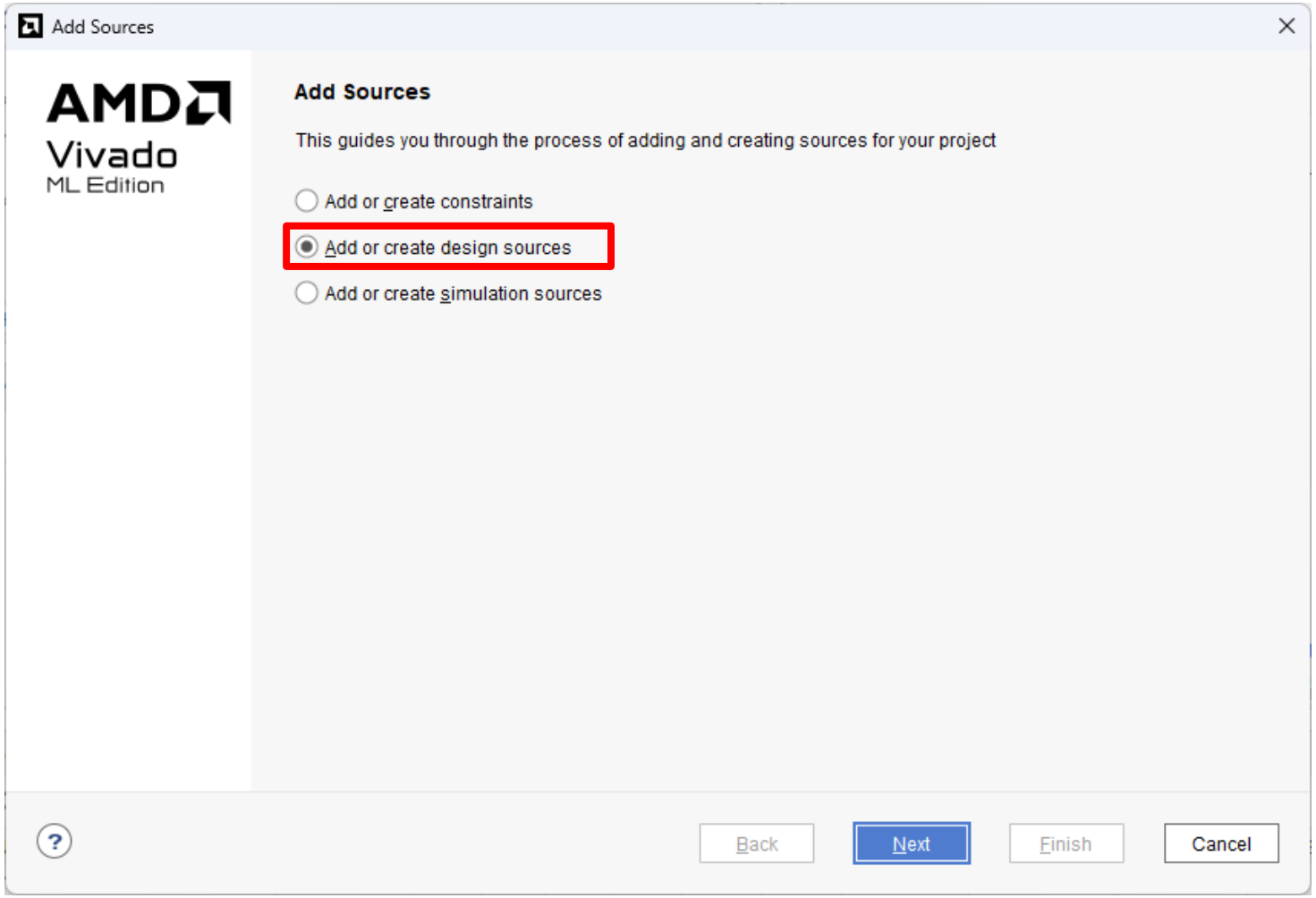
# Simulation – Start with Vivado.

- 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”



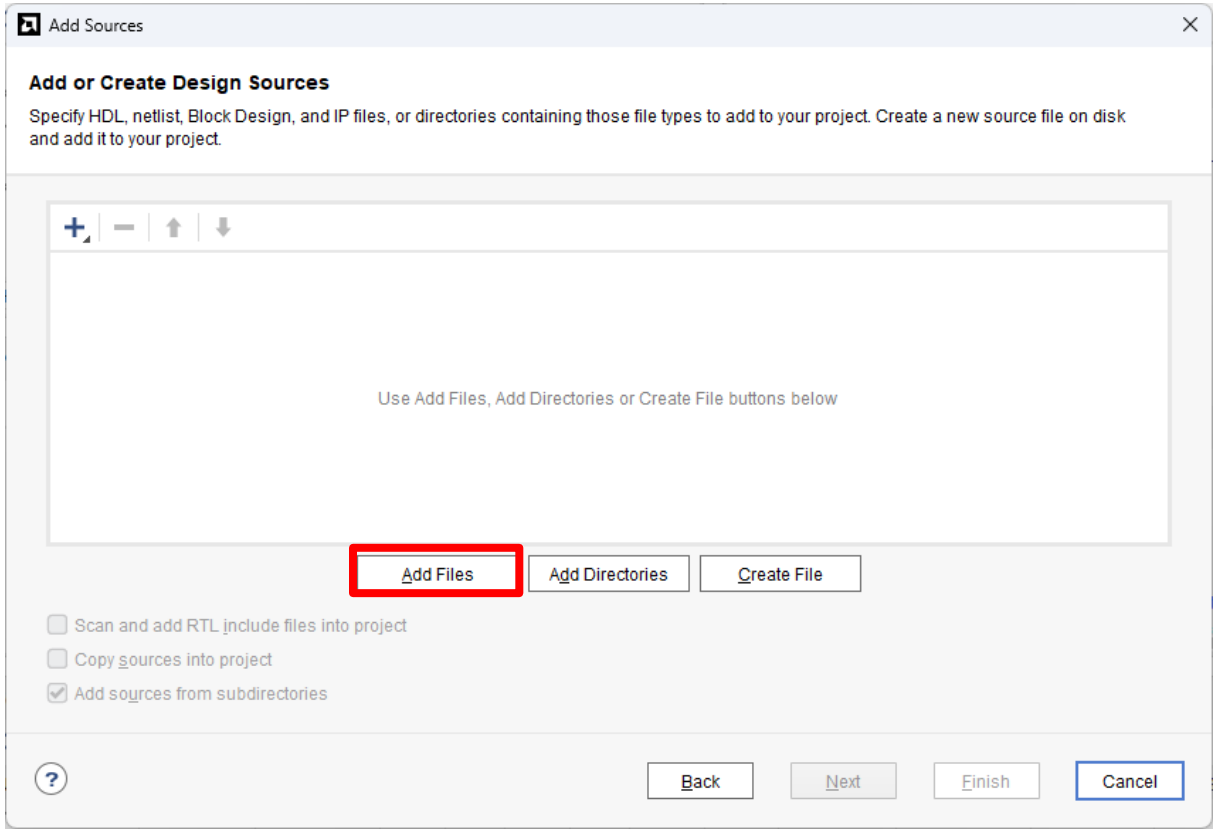
# Simulation – Start with VIVADO

- 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.

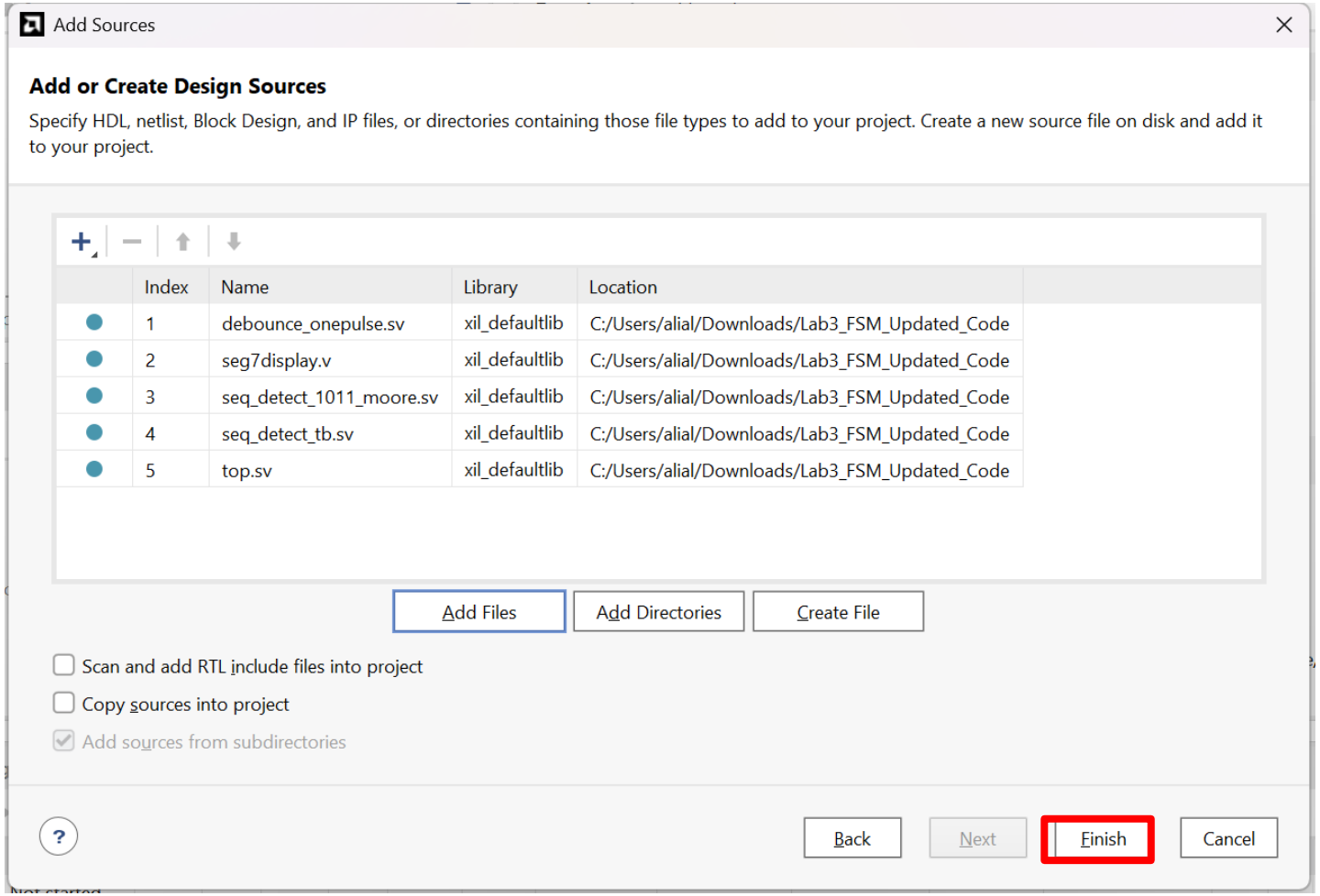


# Simulation – Start with VIVADO

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

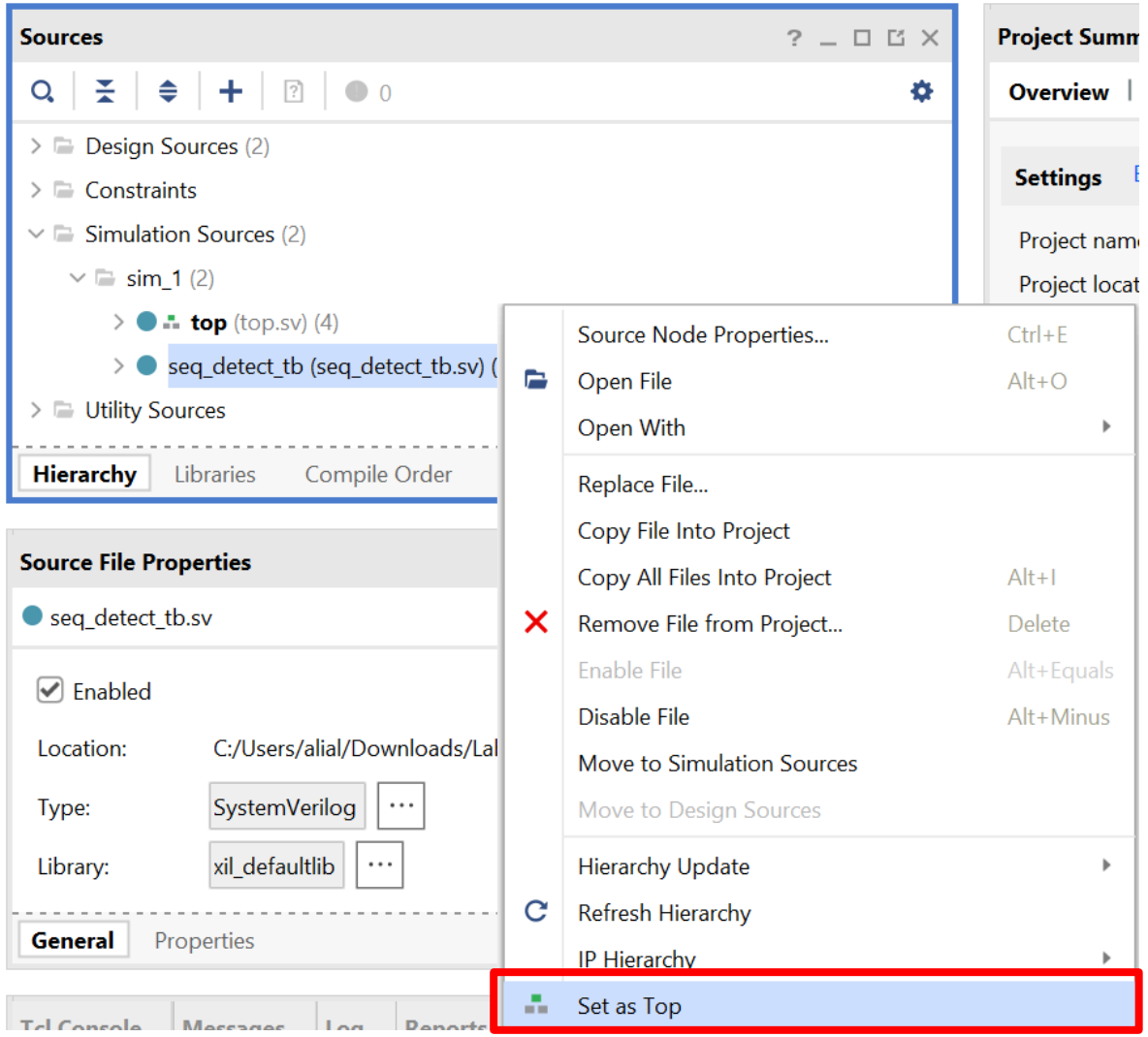
# Simulation – Start with VIVADO

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



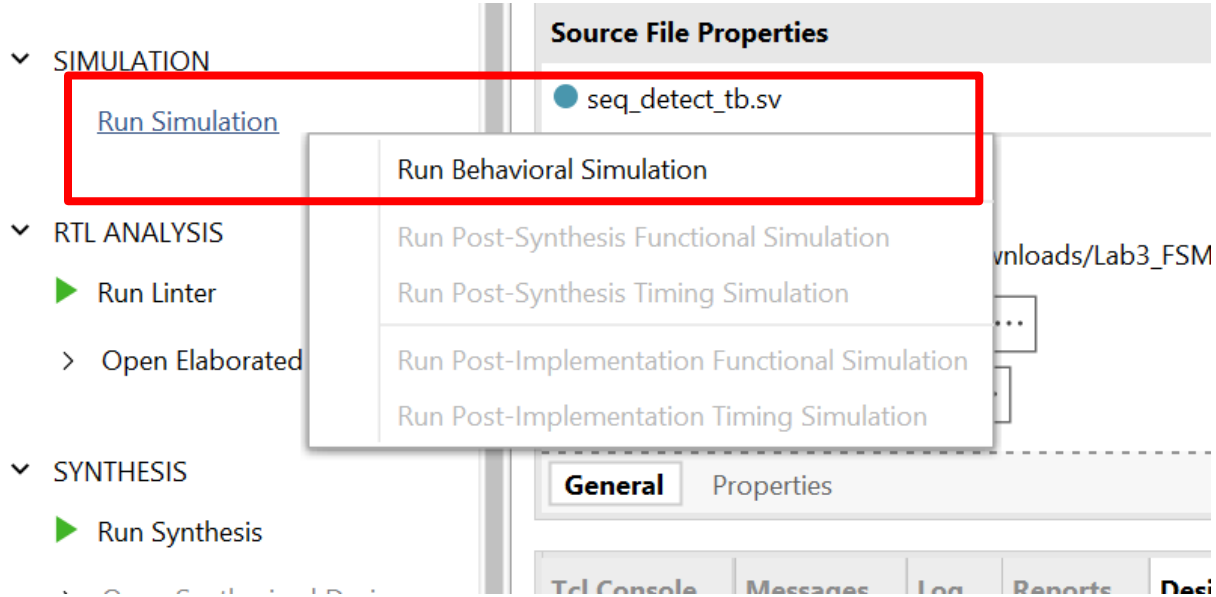
# Simulation – Starting with Vivado

- 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.



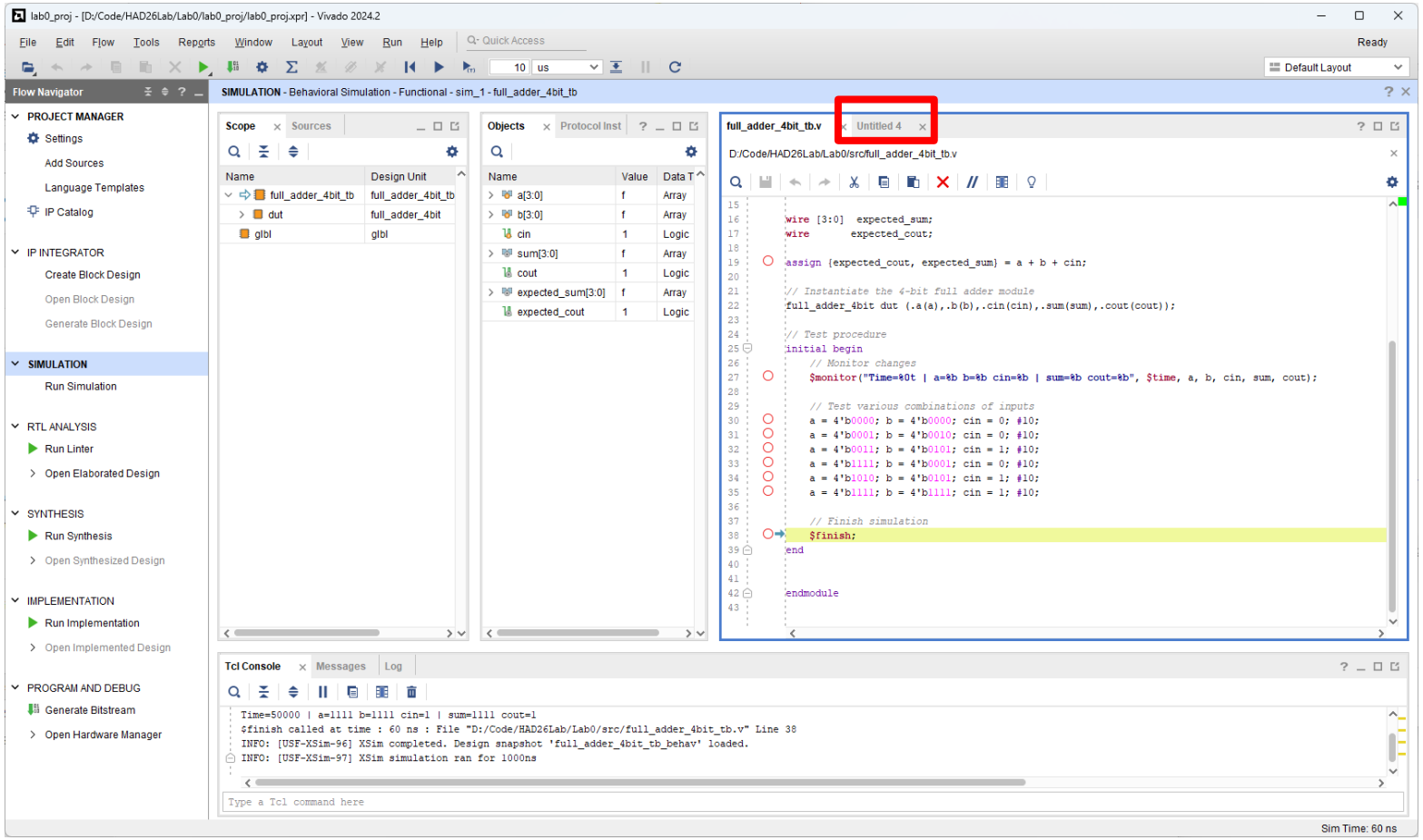
# Simulation – Starting with Vivado Project

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



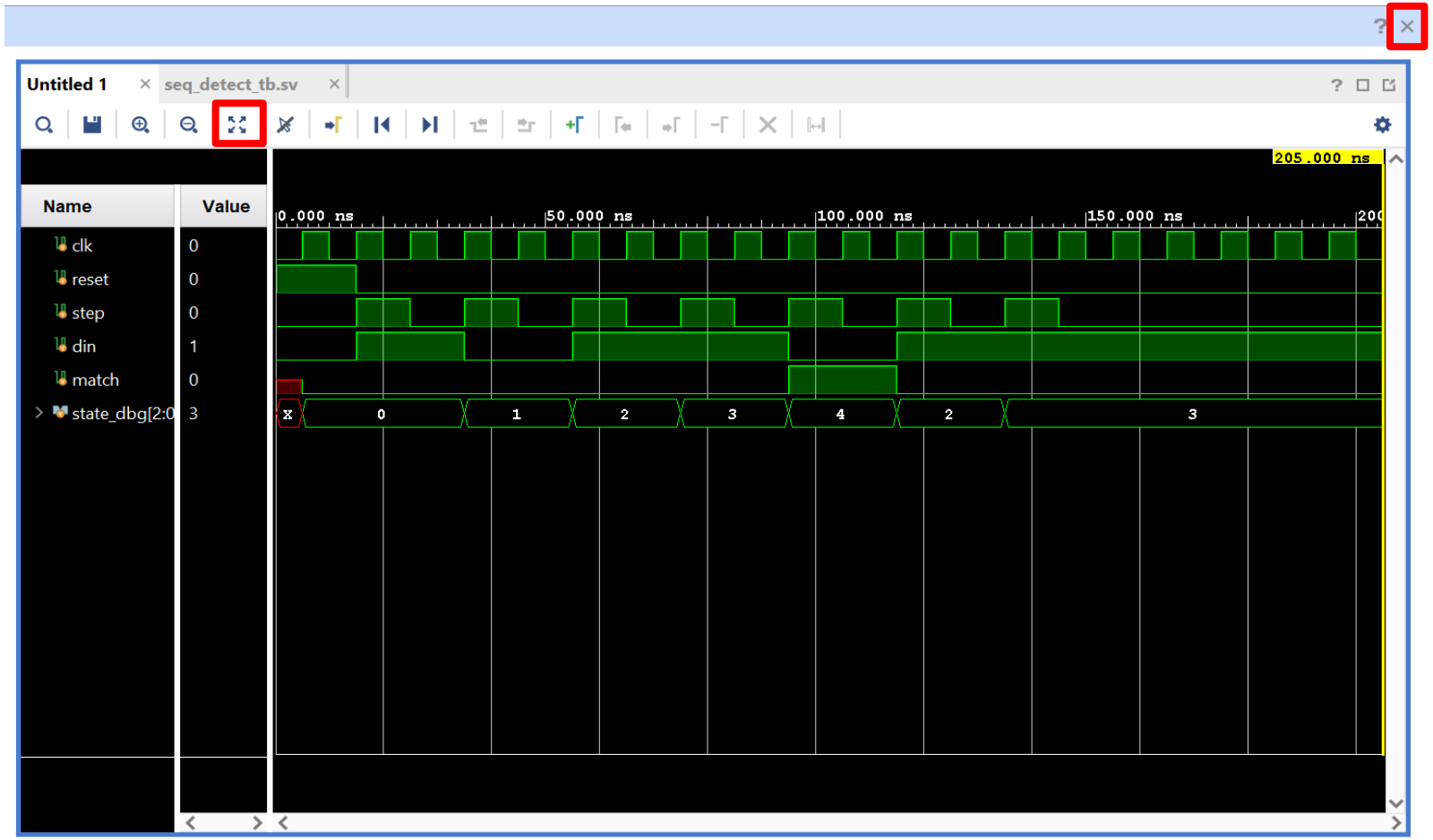
# Simulation – Starting with Vivado Project

- 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.



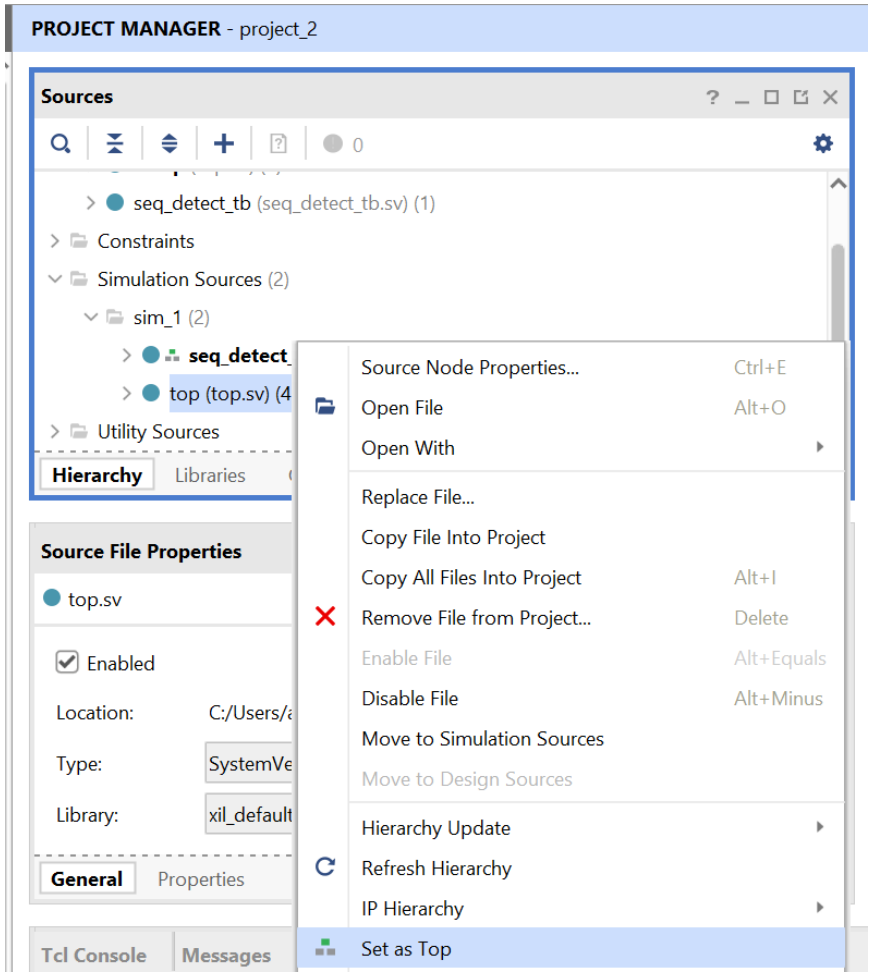
# Simulation – Starting with Vivado Project

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

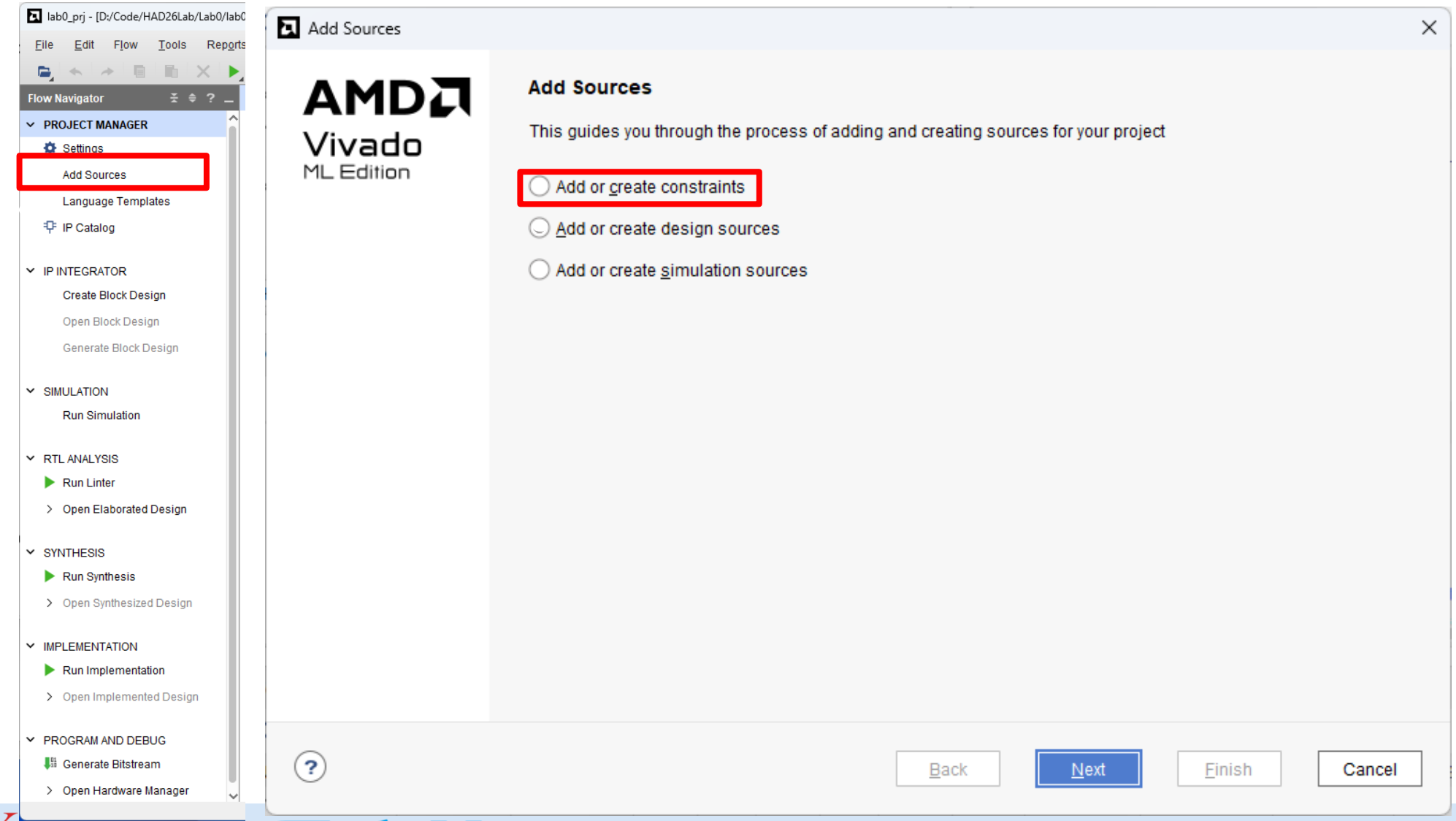


# Change TOP File

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

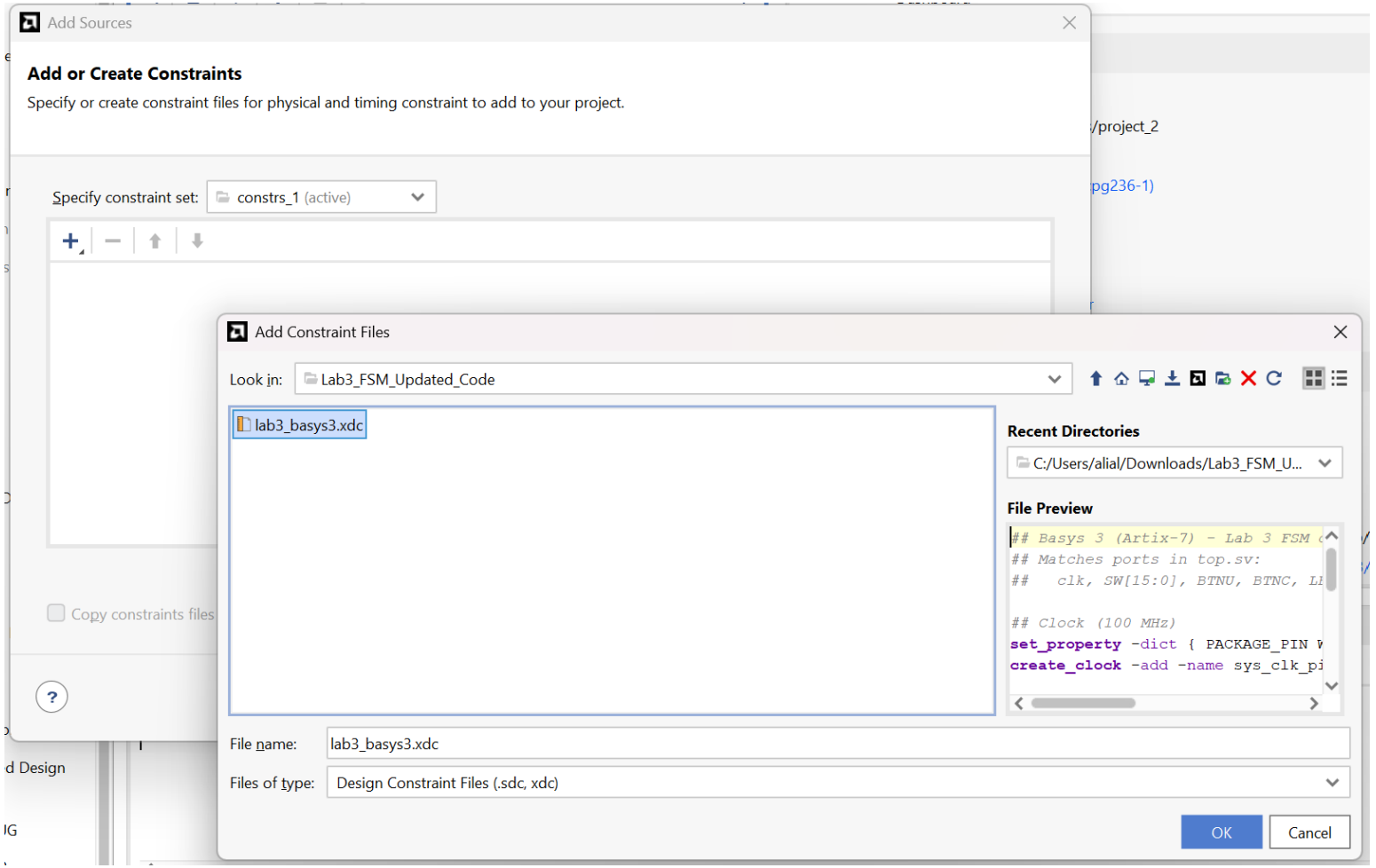


# Add Constraints



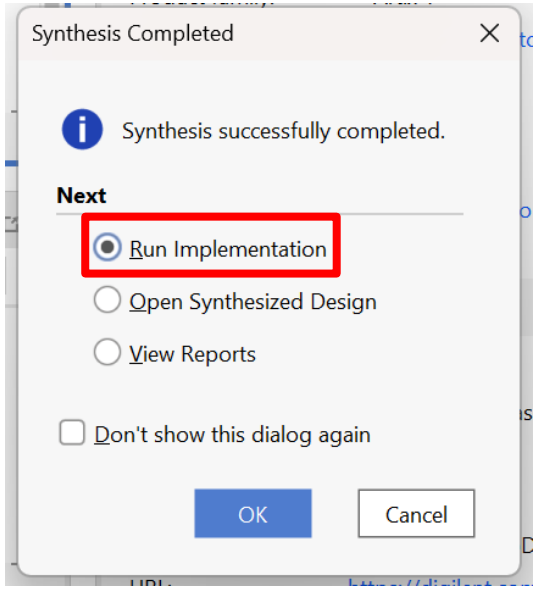
# Basys3 XDC

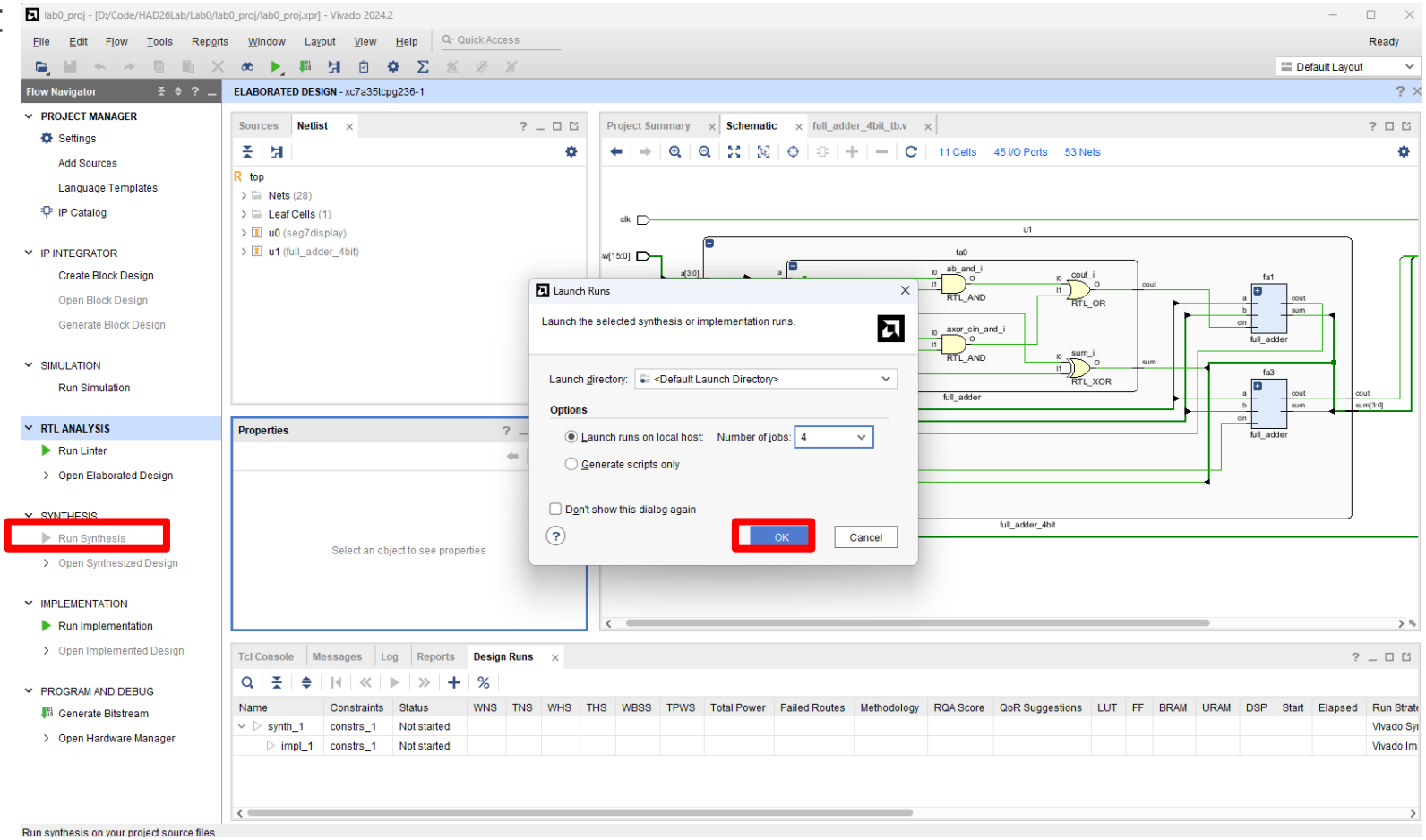
**The XDC file (constraints) is the translation layer between your signal names and the physical pins on the chip. Without it, Vivado doesn't know that sw[0] means "the leftmost switch on pin V17", it's just a name.**



# Starting with Vivado Project - Synthesis

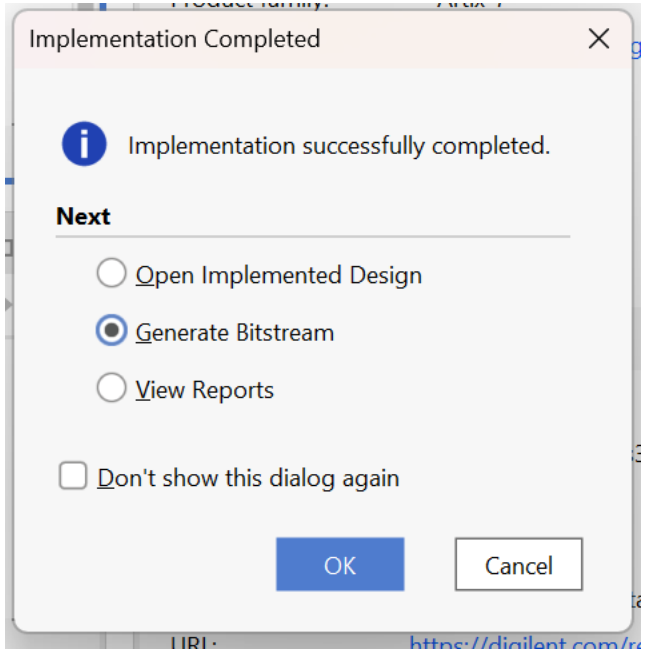
- In the Flow Navigator, click “Run Synthesis”, then click OK to start synthesis. This process takes a while.
- When completed, choose “Run Implementation”.

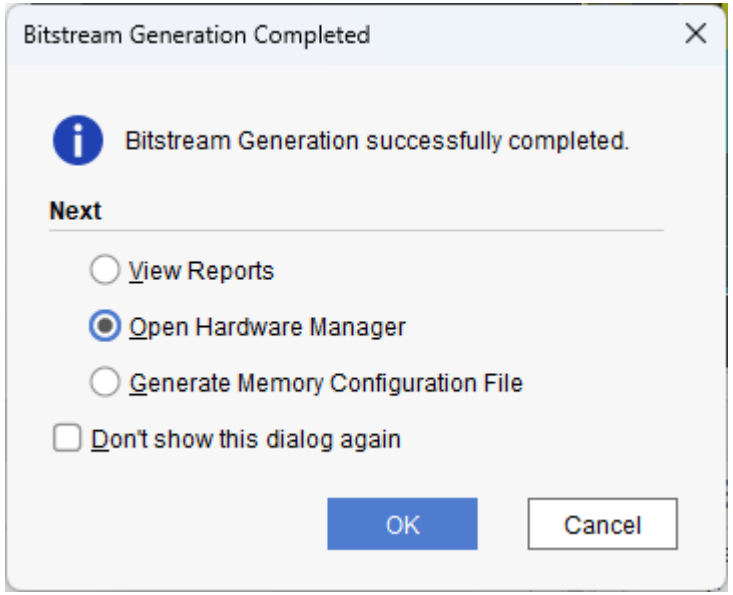


# Starting with Vivado Project - Programming

- 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*