RISC-V Education

Creating Project

In this tutorial, you will **learn** how to develop a basic RISC-V project within the ChipInventor platform. We will cover everything from creating the project and setting up the main modules (processor, memories, registers, etc.) to simulating and testing on hardware. The structure follows the same pattern used in previous tutorials (such as “Distance Sensor” or “DC Motor”).

Initial Steps:

1. Open **ChipInventor**.

2. Click on **New Project**.

3. Fill in the fields:

- **Name:** RISC-V Basic
- **Description**: Basic project to understand the RISC-V architecture in ChipInventor
- **Type**: FPGA

4. Click **Create** to generate the new project.

This environment is where you will add all the blocks (Verilog modules) and make the necessary connections to have a functional RISC-V system.

Understanding the Project and the Blocks Used

Below are the main blocks that make up our RISC-V project, in a structure similar to the previous tutorials.

franken_riscv

The central processor, responsible for fetching, decoding, and executing instructions, as well as handling memory/register reads and writes.

- **Main Inputs:**
  - clk: the system clock
  - reset: global reset signal
  - instruction: the instruction from the instruction memory (imem)
  - read_data: data returned by the data memory (blockram)

- **Main Outputs:**
  - pc: the program counter value
  - mem_write_Mem: enables writes to data memory
  - write_data: the value to be written to data memory
  - alu_result_Exec: result of arithmetic/logic operations
  - reg_write_WB: enables writes to the register file

imem

- Function: The instruction memory for RISC-V.
- Behavior: Receives pc and returns the corresponding instruction on instr.

blockram

- Function: Data memory for load and store operations.
- Behavior: Used by the processor to read (read_data) or write (write_data) data at the address (addr) provided by the ALU.

register

- Function: The RISC-V register file (x0 to x31).

- Behavior: Reads or writes values based on control signals (reg_write_WB, RS1, RS2, RD).

inverterC

- Function: Inverts the input signal (commonly used for generating an active-low reset).

one_hz_clock

- Function: Generates a 1 Hz clock for slowing down and making the system operation more observable (optional).

screen (optional)

- Function: Displays data on a screen, useful for seeing register values or debug messages.

textEngine (optional)

- Function: Converts characters into bitmaps to be displayed by screen.

mux4_8, bus_to_wires, bus_to_bus_4_5, stringbyte, alu_decoder, and others

- Function: Support modules for signal routing, bus multiplexing, and instruction decoding.

Finally, a top module integrates all these components, mapping outputs to physical pins.

Connecting the Blocks

To assemble your project in the ChipInventor block diagram:

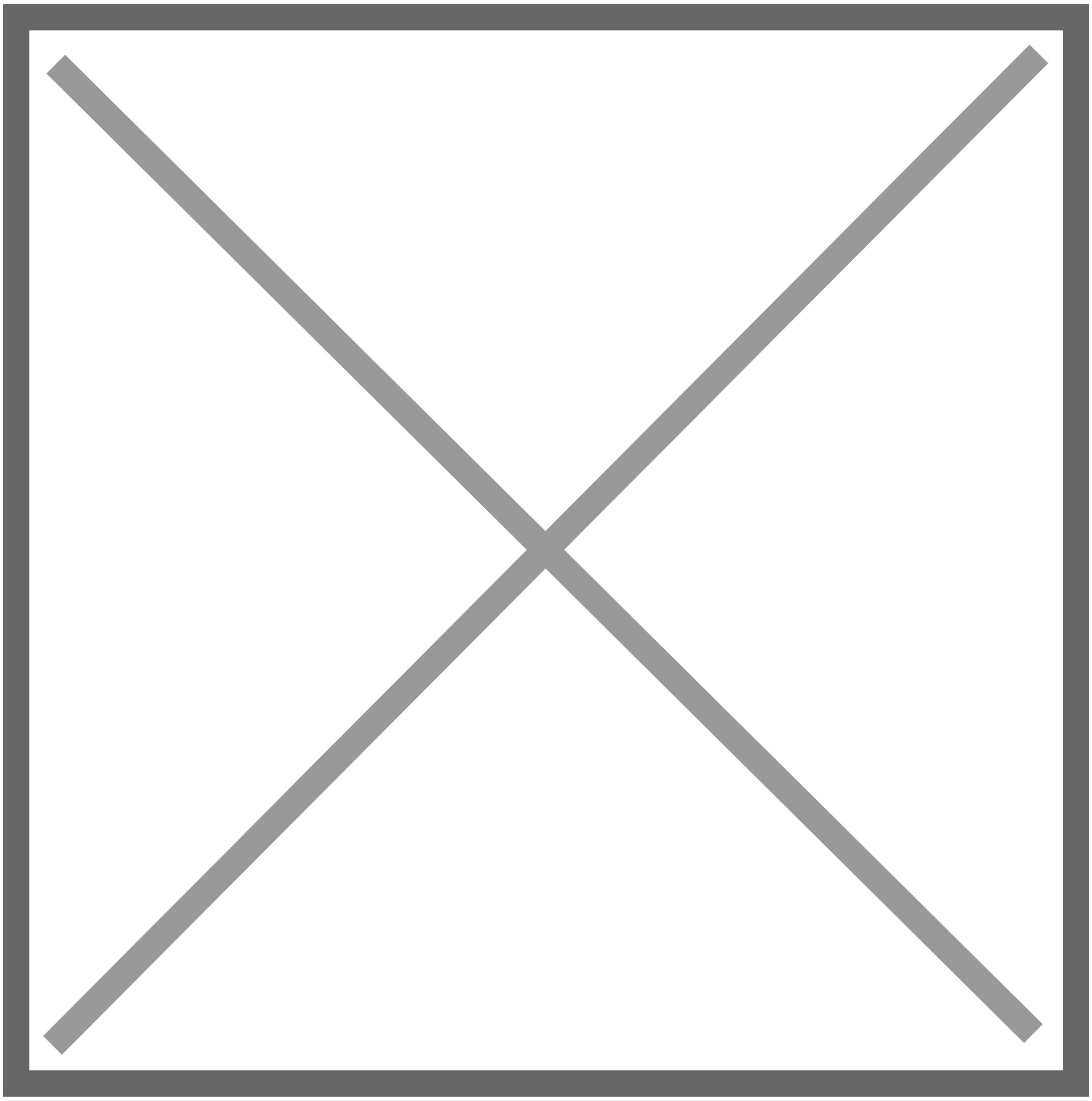
1. Drag each block (e.g., franken_riscv, imem, blockram, register) onto the workspace.

2. Connect the signals according to the Verilog code or as described in previous tutorials. Examples:

- The clk signal (from one_hz_clock or an Input Pin) should go to the clk input of franken_riscv, register, blockram, and other modules.
- reset (from inverterC or another reset block) connects to the reset inputs of these modules.
- The pc output of franken_riscv is fed into imem, which returns instr back to franken_riscv via instruction.
- For data memory, alu_result_Exec serves as the address (addr), write_data is the write input, and read_data is the output from blockram.
- RS1, RS2, RD, reg_write_WB, and write_reg_WB connect the franken_riscv to the register module.

3. If you plan to use a display, connect screen and textEngine as indicated.

4. Save your diagram and ensure all modules have correct clock, reset, and signals.



Project Simulation

Before programming your FPGA board, validate the RISC-V operation in simulation:

1. Click Simulate at the top of the ChipInventor interface.

2. Choose Advanced Simulation to see detailed waveforms.

3. Click Menu → Run Iverilog.

4. If there are no errors, observe key signals such as:

   - pc: should advance each cycle (unless there is a stall).
   - instruction: the instruction read from imem.
   - alu_result_Exec: ALU results.
   - mem_write_Mem and write_data: enabling and value of writes to memory.
   - read_data: data returned by memory on a load operation.

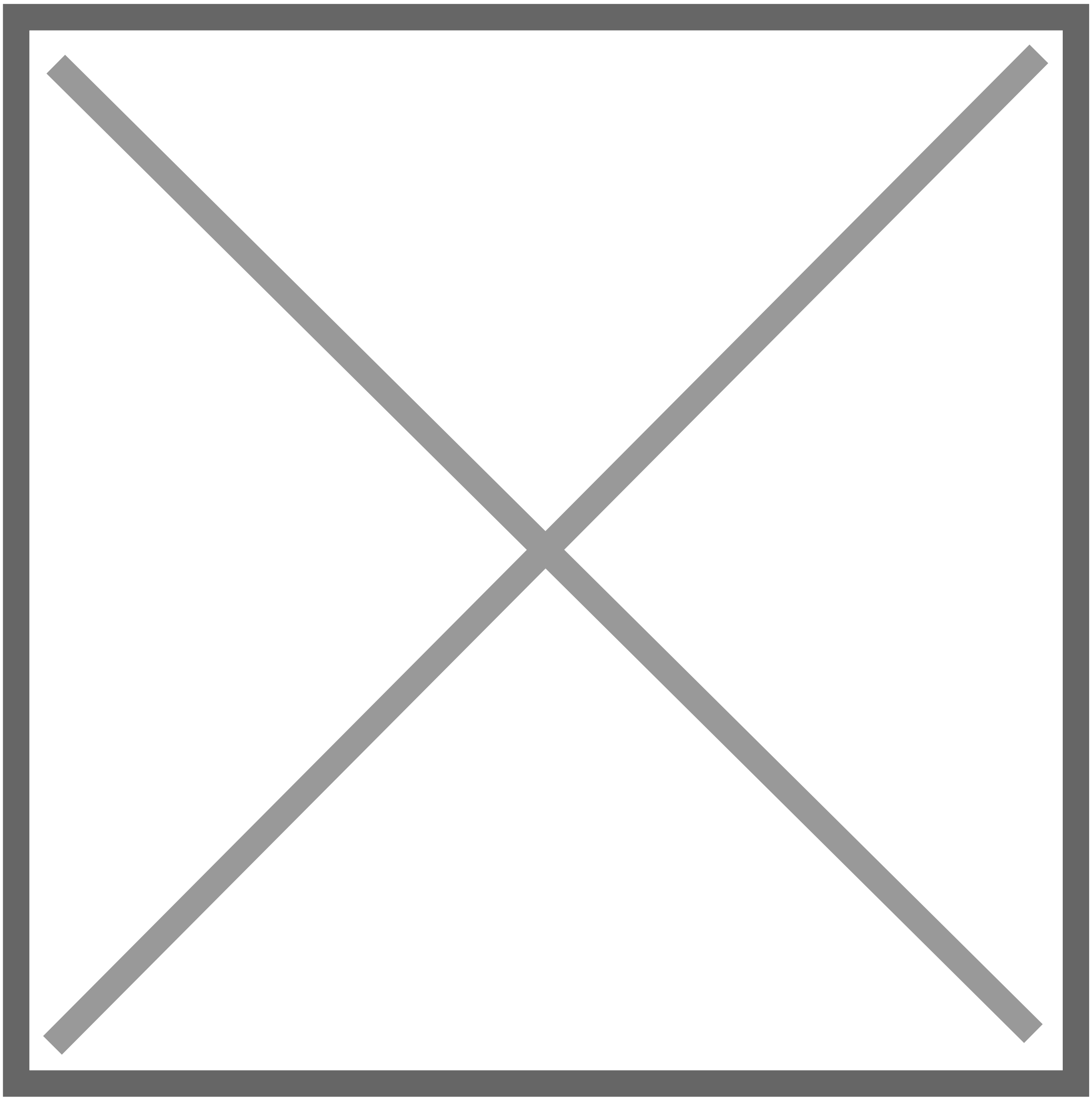
5. If you encounter errors, review connections, bus widths, or clock/reset parameters.

6. Repeat simulation until everything behaves as expected.

FPGA Synthesis and Programming

After confirming your simulation is correct:

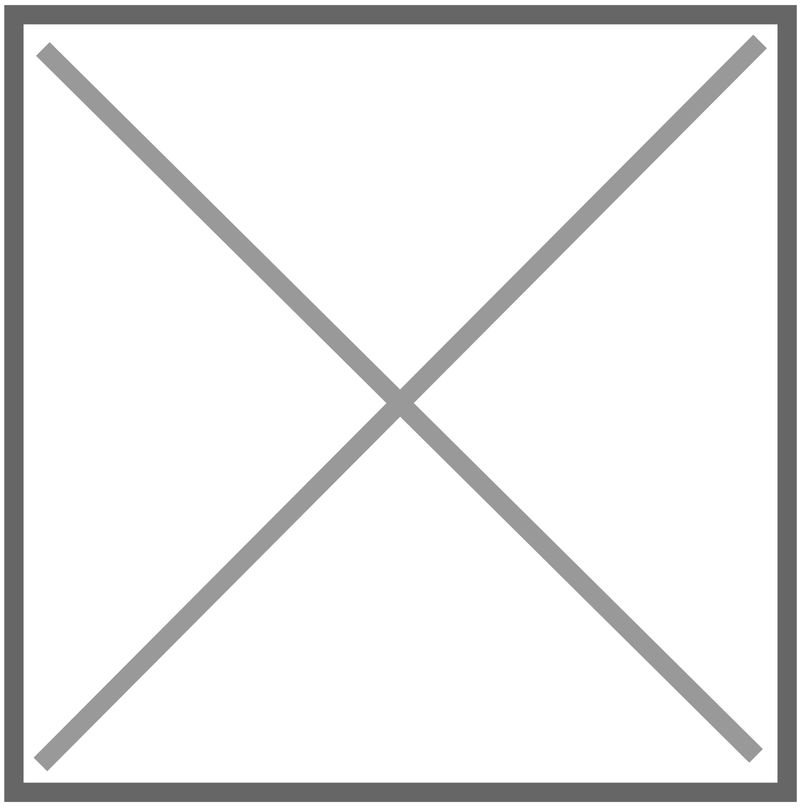
1. Go to the **Synthesize tab**.
2. Select **Start Synthesis**.
3. Check that all items **turn green** (successful synthesis).
4. **Connect your FPGA** board to the computer via USB.
5. Choose the appropriate **serial port** (often labeled “Enhanced”).
6. Click **Flashing** to program the FPGA with the bitstream.



Hardware Validation

With the FPGA programmed, proceed to practical testing:

1. If you use an external clock, ensure it is at the desired frequency. If using one_hz_clock, confirm it is generating a 1 Hz signal.
2. If you have LEDs or a screen, check if they show expected behavior (e.g., a LED lighting up when the processor writes to memory).
3. If you have a reset button, verify that pc and registers reset appropriately.
4. If there is a UART, connect it to a serial interface to observe possible messages or output values.
5. Adjust the contents of imem (the RISC-V program) or system parameters (clock, reset, bus widths) if needed.



Wrapping Up

Congratulations! You have built a **basic RISC-V project in ChipInventor**. This process included:

- Creating the project and importing modules
- Interconnecting the processor (franken_riscv) with memories and peripherals
- Detailed simulation
- Synthesis and FPGA programming
- Real hardware testing

From here, you can:

- Add custom instructions or more complex programs in imem.
- Integrate additional peripherals like I2C, SPI, UART, or displays.
- Enhance the ALU with multiplication, division, or other operations.
- Experiment with simple embedded operating systems or interrupt handling.

Keep exploring ChipInventor and creating increasingly sophisticated digital design and RISC-V projects!