CPU8 Processor

Full Technical & User Documentation

8-bit Accumulator-Based FPGA CPU

0. How to Use This Document

This document provides end-to-end coverage of the CPU8 processor project. It is designed to guide you from architectural understanding through simulation, synthesis, and hardware deployment. Following this document sequentially will result in a working, verified CPU implementation on your target FPGA platform.

The document covers the following areas:

- Architecture explanation and design philosophy
- Single-cycle execution model and timing analysis
- Verification strategy using multiple simulation tools
- Simulation procedures for Icarus Verilog and Vivado
- Build automation using Vivado TCL scripts
- Hardware wiring and external peripheral connections
- Mandatory screenshot locations for client delivery

1. Executive Summary

CPU8 is a minimal single-cycle 8-bit accumulator CPU designed for education, demonstration, and FPGA portability. The processor implements a clean, understandable architecture that executes one instruction per clock cycle, making it ideal for learning digital design principles and FPGA development workflows.

1.1 Key Characteristics

Feature	Specification
Execution Model	Single-cycle (CPI = 1)
Architecture Type	Accumulator-based
Data Width	8 bits
Instruction Set	6 instructions: LDA, ADD, XOR, AND, JMP, HLT
HDL Compatibility	Generic Verilog (no vendor IP required)
FPGA Utilization	< 1% on typical FPGAs
Vendor Support	Xilinx (Vivado) and Intel (Quartus) compatible

2. CPU Architecture Overview

The CPU8 processor follows a classic accumulator-based architecture where all arithmetic and logical operations use a single accumulator register as one of the operands. This design choice minimizes hardware complexity while providing sufficient functionality for educational purposes and simple control applications.

2.1 Architectural Registers

The processor maintains three architectural registers that define its complete state at any given moment:

Register	Width	Description
PC	8 bits	Program Counter — holds address of current instruction
ACC	8 bits	Accumulator — primary data register for all operations
halt	1 bit	Execution stop flag — when set, CPU stops fetching instructions

2.2 Instruction Format

Each instruction is encoded in a single 8-bit word, divided into two fields:

[opcode (3 bits) | immediate/address (5 bits)]

The 3-bit opcode field allows for up to 8 different instructions, of which 6 are currently implemented. The 5-bit immediate field provides values ranging from 0 to 31, which is sufficient for small programs and demonstrations.

2.3 Instruction Set Architecture (ISA)

The CPU8 implements a minimal but complete instruction set that supports data loading, arithmetic operations, logical operations, control flow, and program termination:

Opcode	Mnemonic	Operation
000	**LDA imm**	ACC ← imm (Load immediate value into accumulator)
001	**ADD imm**	ACC ← ACC + imm (Add immediate to accumulator)
010	**XOR imm**	ACC ← ACC ⊕ imm (Bitwise XOR with immediate)
011	**AND imm**	ACC ← ACC ∧ imm (Bitwise AND with immediate)
100	**JMP addr**	PC ← addr (Unconditional jump to address)
111	**HLT**	halt ← 1 (Stop program execution)

3. Single-Cycle Execution Model

The CPU8 implements a pure single-cycle execution model, meaning that every instruction completes in exactly one clock cycle. This architectural choice has profound implications for both the hardware design and the processor's performance characteristics.

3.1 What is a CPU Cycle?

A CPU cycle (also called a clock cycle or machine cycle) represents one complete oscillation of the processor's clock signal. During each cycle, the processor performs a specific unit of work. In a single-cycle architecture like CPU8, this unit of work is an entire instruction from fetch to completion.

The clock signal serves as the heartbeat of the processor, synchronizing all internal operations. When the clock transitions from low to high (rising edge), registers capture new values, and the combinational logic begins computing the next state.

3.2 Cycle Phases in CPU8

Although CPU8 executes each instruction in a single cycle, the cycle itself can be conceptually divided into phases that occur sequentially within that cycle:

Instruction Fetch Phase:

The Program Counter (PC) provides an address to the Program ROM. The ROM is implemented as combinational logic, so the instruction appears at its output almost immediately after the address is applied. This fetched instruction contains both the operation to perform and any immediate data needed.

Instruction Decode Phase:

The instruction decoder extracts the opcode from bits [7:5] and the immediate value from bits [4:0]. This is purely combinational logic implemented as simple wire assignments, requiring no additional clock cycles.

Execute Phase:

The ALU performs the operation specified by the opcode. For arithmetic operations (ADD), the ALU adds the immediate value to the current accumulator contents. For logical operations (XOR, AND), it performs the bitwise operation. For LDA, it simply passes the immediate value through.

Write-Back Phase:

On the rising edge of the clock, the ALU result is captured into the accumulator register, and the program counter is updated (either incremented or loaded with a jump target). The halt flag is set if a HLT instruction was executed.

3.3 Cycles Per Instruction (CPI)

The Cycles Per Instruction (CPI) metric indicates how many clock cycles are needed on average to execute one instruction. For CPU8:

CPI = 1 (constant for all instructions)

This is the defining characteristic of a single-cycle processor. Every instruction, regardless of complexity, takes exactly one cycle. This predictability simplifies timing analysis but means that the clock period must be long enough to accommodate the slowest instruction path.

3.4 Critical Path and Clock Frequency

The critical path in CPU8 runs through the following components:

1. PC register output delay
2. Program ROM access time
3. Instruction decoder propagation
4. ALU computation time
5. Accumulator register setup time

The sum of these delays determines the minimum clock period. For CPU8 on a Basys 3 board running at 100 MHz (10 ns period), timing analysis shows comfortable positive slack, indicating the design could run even faster if needed.

3.5 Execution Timing Example

Consider a simple program that loads a value and adds to it repeatedly:

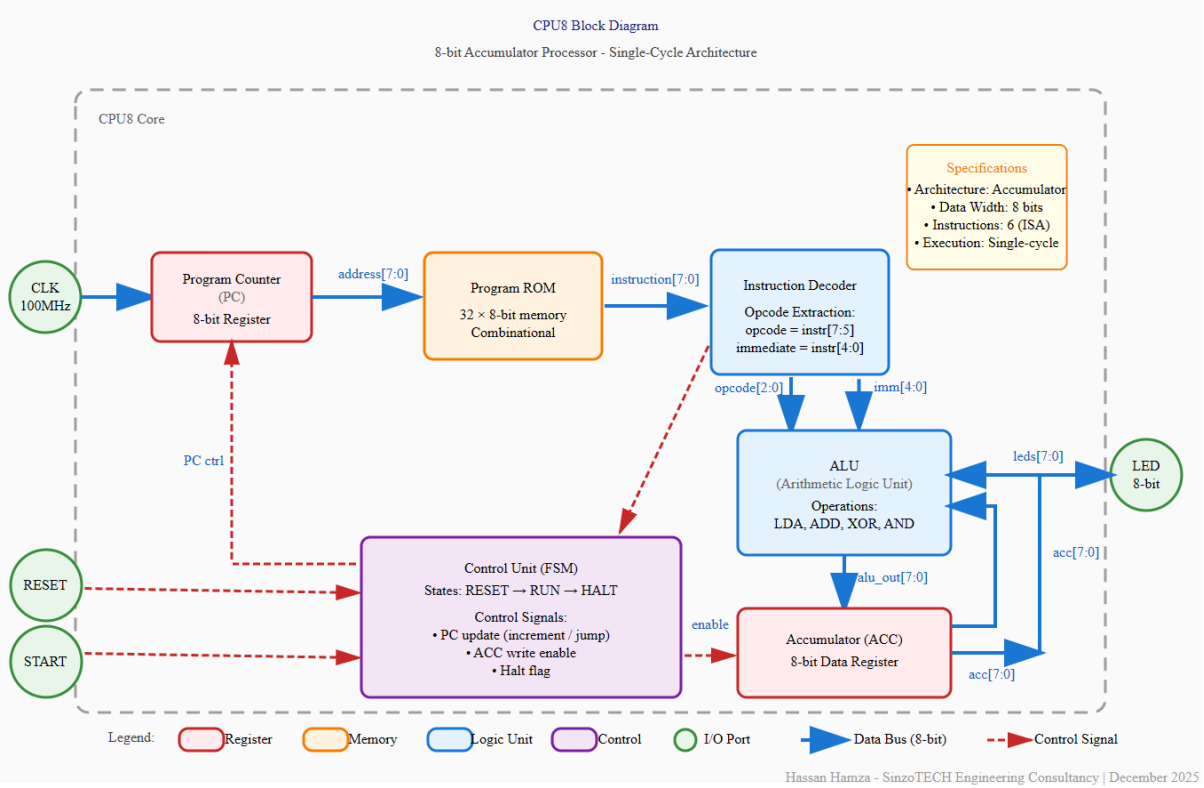
```
LDA 1       ; Cycle 1: ACC = 1

ADD 1       ; Cycle 2: ACC = 2
ADD 1       ; Cycle 3: ACC = 3
JMP 1       ; Cycle 4: PC = 1, loop back to ADD
```

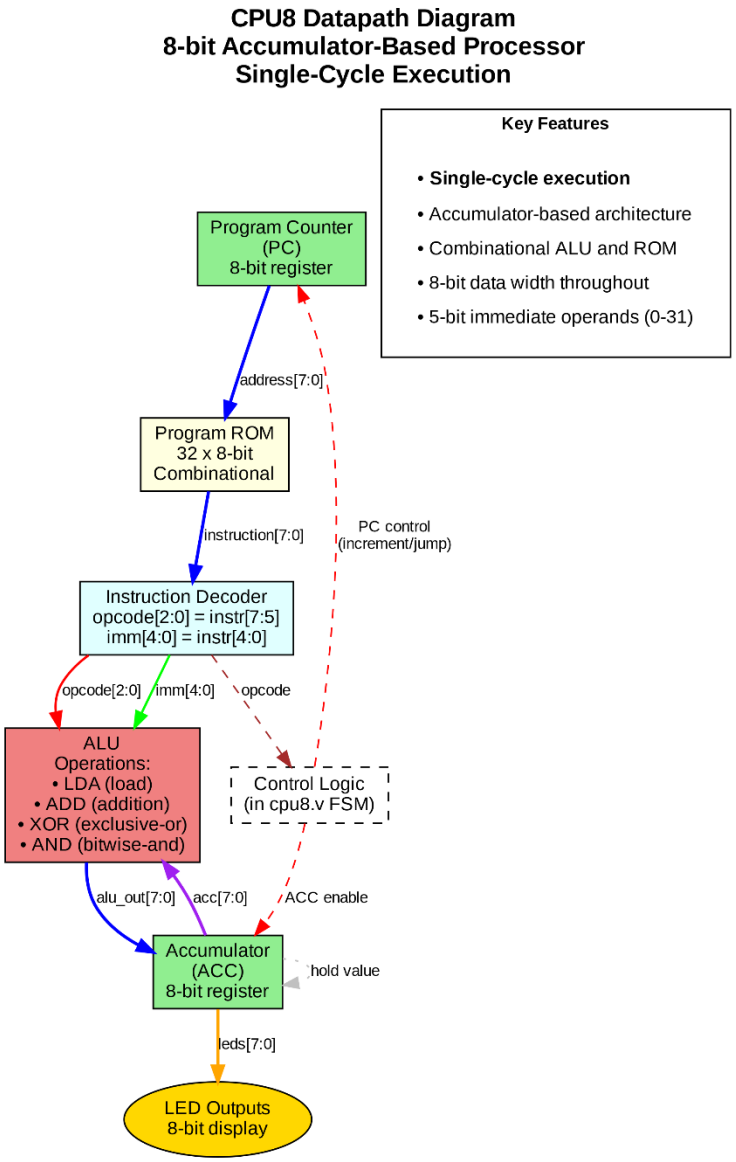
At 100 MHz, each instruction executes in 10 nanoseconds. The entire four-instruction loop completes in 40 nanoseconds, after which the pattern repeats.

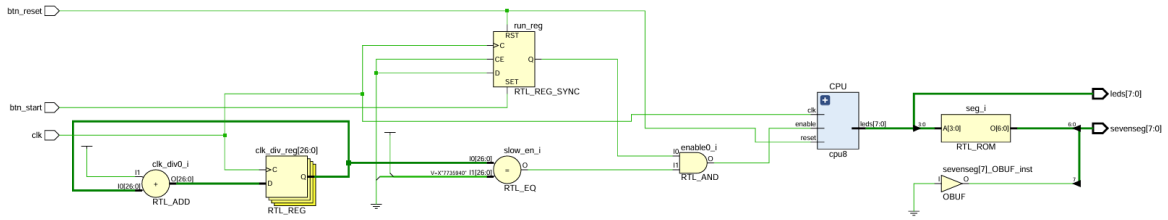
4. Architecture Diagrams

This section specifies the required diagrams for client delivery. Each diagram must be created and inserted as specified to provide complete documentation of the CPU8 architecture.

4.1 Datapath Block Diagram





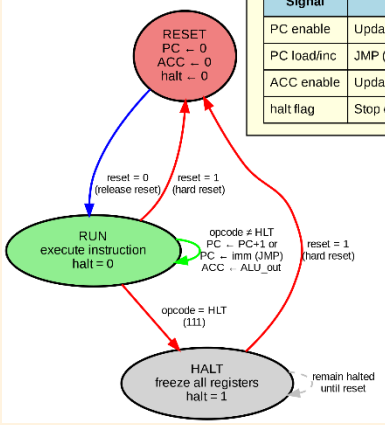


The datapath diagram must clearly illustrate the following signal flows and components:

- Program Counter (PC) → Program ROM address input
- Program ROM → Instruction Decoder (8-bit instruction bus)
- Instruction Decoder → ALU (opcode[2:0] and immediate[4:0])
- ALU → Accumulator (alu_out[7:0])
- Accumulator feedback loop to ALU input
- Accumulator → LED output register
- Control Unit connections (dashed lines for control signals)
- Clock and Reset inputs

4.2 Control FSM State Diagram

CPU8 Control FSM (Finite State Machine)
Single-Cycle Execution Control



Control Signals Generated

Signal	Purpose
PC enable	Update program counter
PC load/inc	JMP (load) vs sequential (inc)
ACC enable	Update accumulator
halt flag	Stop execution

Instruction Decode Logic

Opcode	Instruction	Action
000	LDA imm	ACC ← imm, PC ← PC+1
001	ADD imm	ACC ← ACC+imm, PC ← PC+1
010	XOR imm	ACC ← ACC⊕imm, PC ← PC+1
011	AND imm	ACC ← ACC∧imm, PC ← PC+1
100	JMP addr	PC ← addr
111	HLT	halt ← 1

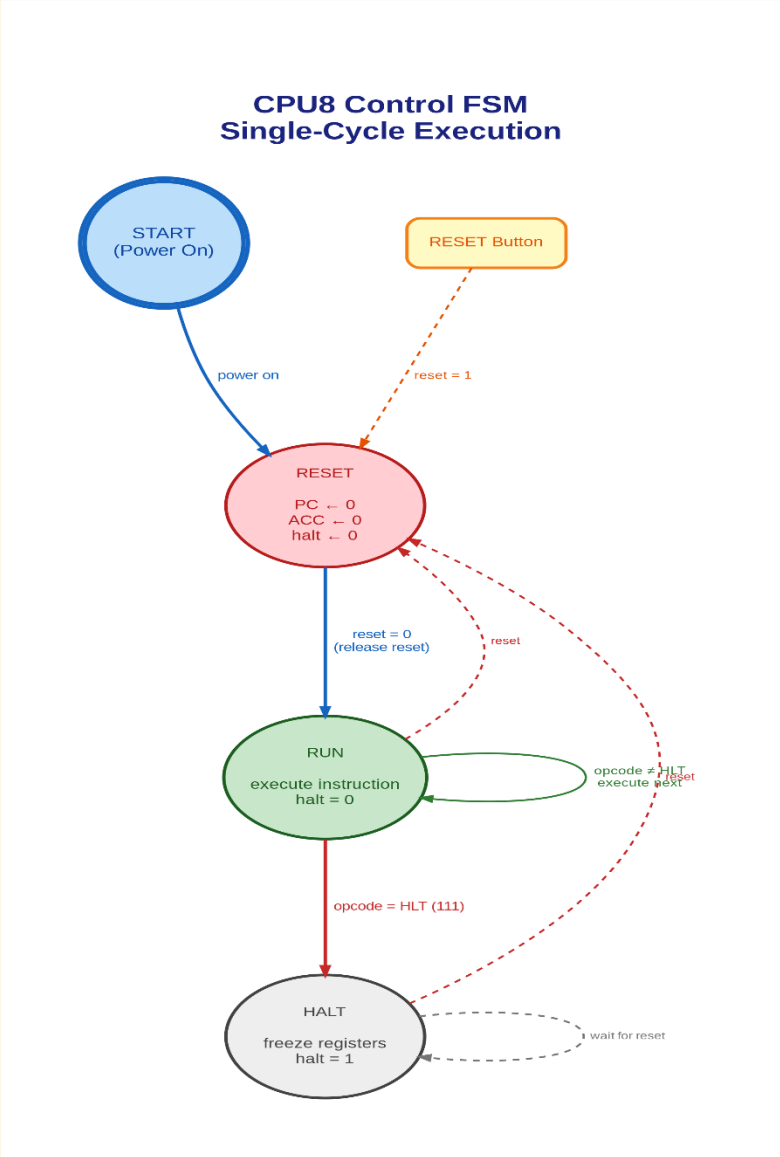
Execution Timing

Clock Cycle N:

1. Fetch instruction from ROM[PC]
2. Decode opcode and immediate
3. Execute ALU operation
4. Update ACC and/or PC

All steps complete in 1 clock cycle

With start and reset buttons :



Filename: cpu8_control_fsm.png

The Control Unit implements a simple three-state finite state machine:

State	Description
RESET	Initial state after power-on or reset assertion. PC and ACC cleared to zero.
RUN	Normal execution state. Instructions fetched and executed each cycle.
HALT	Execution stopped after HLT instruction. CPU idle until reset.

5. Verification Strategy

A robust verification strategy ensures the CPU8 design functions correctly before hardware deployment. The verification flow progresses from fast functional simulation to vendor-specific validation and finally to physical hardware testing.

5.1 Simulation Levels

Level	Tool	Purpose
RTL	Icarus Verilog	Fast functional validation, cross-platform
RTL	Vivado xsim	Vendor-official validation for Xilinx targets
Hardware	FPGA Board	Physical proof of correct operation

6. Simulation — Icarus Verilog

Icarus Verilog provides fast, open-source simulation for initial design validation. Its quick compilation and execution make it ideal for iterative development and debugging.

6.1 Running the Simulation

Navigate to the simulation directory and execute the following commands:

```
cd cpu8_project/sim

iverilog -o cpu8_sim ../src/*.v tb_cpu.v
vvp cpu8_sim
```

6.2 Expected Console Output

```
  [PASS]
                                Edge: Zero during reset : Expected=0x00 Got=0x00
  [INFO] Maximum immediate value: 0x1F (31 decimal, 5-bit limit)
  [PASS] Edge: 5-bit immediate range [0-31] correct
  [PASS] Edge: 8-bit overflow wraps (0xFF+1=0x00)

TEST 8: Single-Cycle Execution Timing
--------------------------------------
  [PASS]
                                 ADD instruction timing: Single-cycle execution verified
  [INFO]
                                 JMP instruction timing: Value unchanged (may be JMP cycle)
  [INFO] All instructions execute in 1 clock cycle

TEST 9: X/Z Value Detection
---------------------------
  [PASS] No X/Z values detected during 20-cycle run

TEST 10: Enable Freeze
----------------------------
  [PASS]
                               ACC frozen when enable=0 : Expected=0x03 Got=0x03
  [PASS]
                              CPU resumes after enable=1 : No X/Z values

TEST 11: Enable blocks execution
----------------------------
  [PASS]
                         PC/ACC unchanged when enable=0 : Expected=0x02 Got=0x02
  [PASS]
                            Execution resumes correctly : No X/Z values

======================================
TEST SUMMARY
======================================
Total Tests:    31
Passed:         31
Failed:         0
Success Rate:   100%
======================================

*** ALL TESTS PASSED ***
V All ISA instructions verified
V ALU regression complete
V PC regression complete
V Edge cases tested
V Single-cycle timing verified
V No X/Z values detected

tb_cpu.v:193: $finish called at 1105000 (1ps)
PS C:\cpu8_project\sim>
```

Filename: sim_iverilog_pass.png

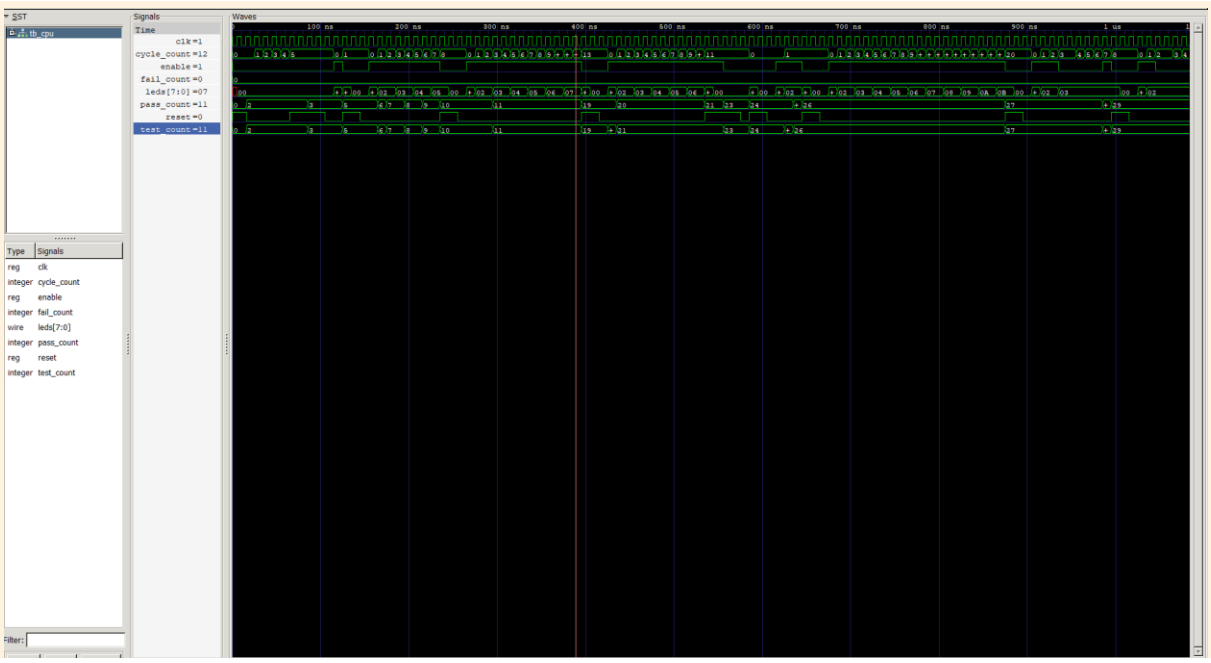
The simulation should complete with the following output:

```
ALL TESTS PASSED

Success Rate: 100%
```

6.3 Waveform Viewing (Optional)

For detailed signal analysis, generate and view waveforms using GTKWave:

```
gtkwave cpu8.vcd
```



Filename: sim_waveform.png

7. Simulation — Vivado (Scripted)

Vivado simulation provides vendor-official validation and ensures the design will synthesize correctly for Xilinx FPGAs.

7.1 Simulation Script

The simulation is automated using the provided TCL script:

```
run_simulation_vivado.tcl
```

7.2 Execution Command

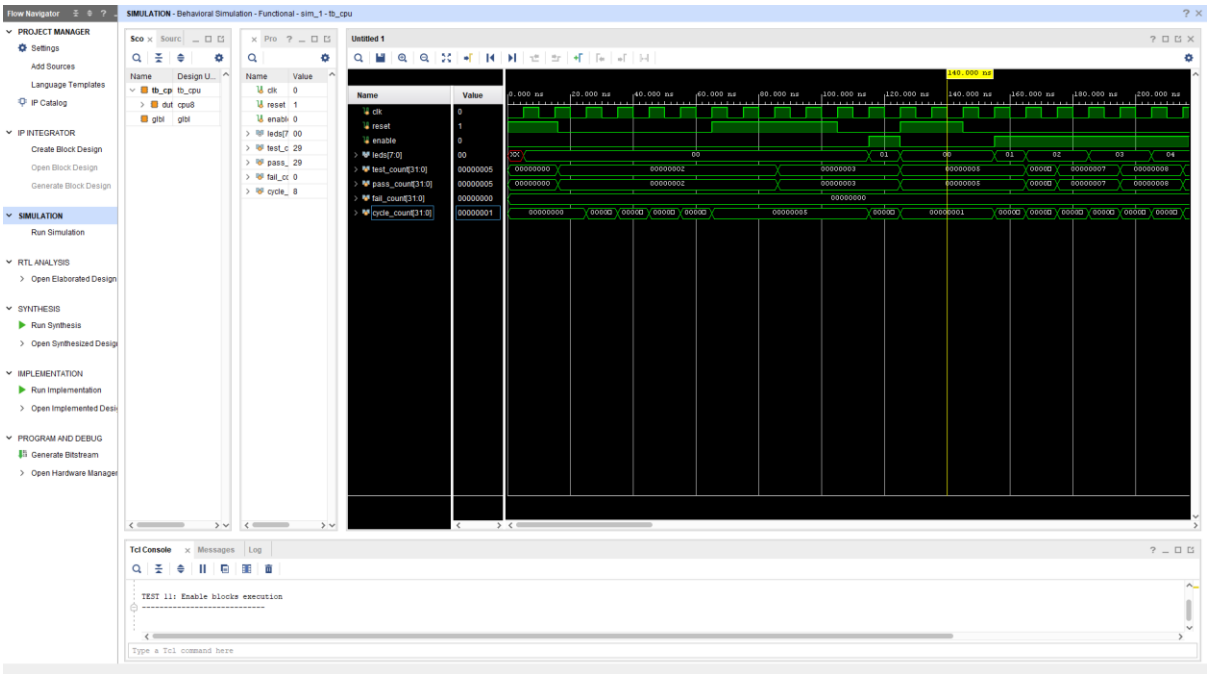
```
vivado -mode tcl -source run_simulation_vivado.tcl
```

```
TEST 9: X/Z Value Detection
----------------------------
  [PASS] No X/Z values detected during 20-cycle run

TEST 10: Enable Freeze
----------------------------
  [PASS]
                                ACC frozen when enable=0 : Expected=0x02 Got=0x02
  [PASS]
                              CPU resumes after enable=1 : No X/Z values

TEST 11: Enable blocks execution
----------------------------
INFO: [USF-XSim-96] XSim completed. Design snapshot 'tb_cpu_behav' loaded.
INFO: [USF-XSim-97] XSim simulation ran for 1000ns
launch_simulation: Time (s): cpu = 00:00:00 ; elapsed = 00:00:11 . Memory (MB): peak = 1024.406 ; gain = 0.000
# puts "Running simulation..."
Running simulation...
# run all
  [PASS]
                           PC/ACC unchanged when enable=0 : Expected=0x02 Got=0x02
  [PASS]
                              Execution resumes correctly : No X/Z values

========================================
TEST SUMMARY
========================================
Total Tests:    31
Passed:         31
Failed:         0
Success Rate:   100%
========================================

*** ALL TESTS PASSED ***
V All ISA instructions verified
V ALU regression complete
V PC regression complete
V Edge cases tested
V Single-cycle timing verified
V No X/Z values detected

V Freeze Enabled

V blocks execution Enabled

$finish called at time : 1105 ns : File "C:/cpu8_project/sim/tb_cpu.v" Line 195
# puts "================================================"
================================================
# puts "Simulation completed successfully"
Simulation completed successfully
# puts "================================================"
================================================
Vivado%
```

7.3 Waveform Configuration

Due to Vivado 2020.1 limitations, waveform signals must be added manually after simulation starts. Configure the waveform viewer to display:

- clk — System clock
- btn_reset — Reset button input
- btn_start — Start button input
- CPU.PC — Program Counter value
- CPU.ACC — Accumulator value
- leds[7:0] — LED output signals



Filename: vivado_waveform.png

8. FPGA Build — Xilinx Vivado

The FPGA build process synthesizes the Verilog design into a bitstream file that can be programmed onto the target device.

8.1 Automated Build (Recommended)

Use the provided TCL script for automated synthesis, implementation, and bitstream generation:

```
cd cpu8_project/scripts

vivado -mode batch -source build_vivado_cora_z7_7S.tcl
```

8.2 Build Artifacts

The build process generates the following artifacts:

- top.bit — Programming bitstream file
- Timing Summary Report — Shows timing closure status
- Utilization Report — FPGA resource usage
- Power Report — Estimated power consumption
- BUILD_SUMMARY.txt — Consolidated build results

8.3 Intel Quartus Support

The CPU8 design is fully compatible with Intel Quartus Prime for targeting Intel/Altera FPGAs such as the DE0-Nano. The generic Verilog code requires no modification; only the constraint file (pin assignments) needs to be adapted for the target board. Quartus build scripts are available if needed for Intel FPGA deployment.

9. FPGA Programming

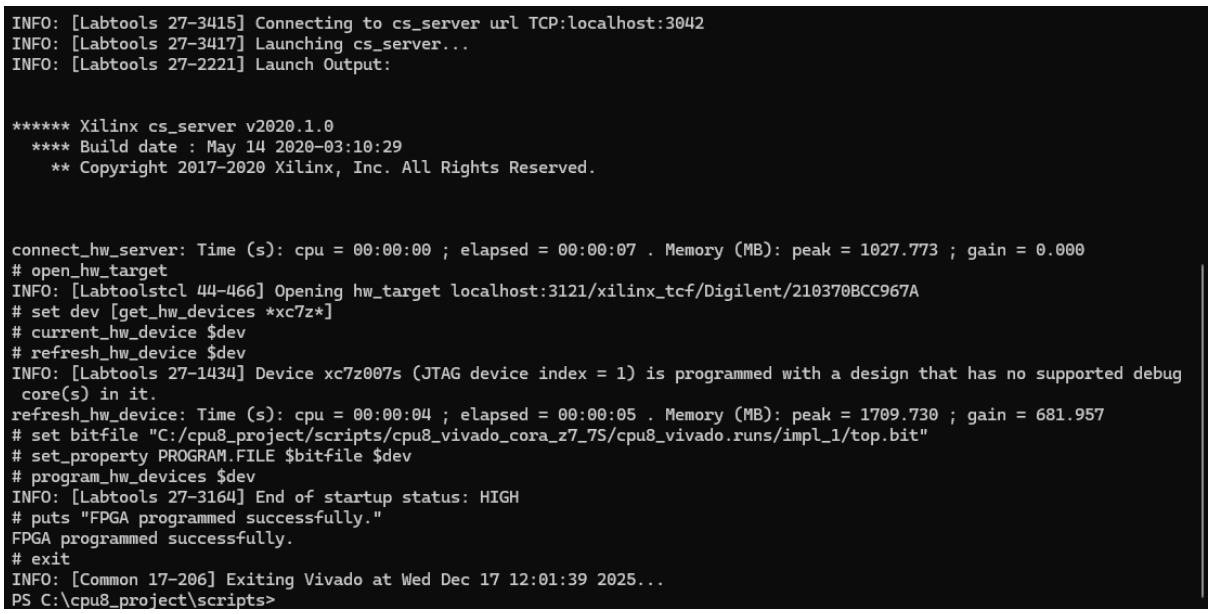
9.1 Vivado Hardware Manager

Follow these steps to program the FPGA:

6. Open Vivado Hardware Manager
7. Click "Auto Connect" to detect the connected board
8. Right-click on the device and select "Program Device"
9. Select the top.bit file from the build output directory
10. Click "Program" and wait for completion

or use this command :

```
cd cpu8_project/scripts C:\Xilinx\Vivado\2020.1\bin\vivado.bat -mode batch -source .\program_fpga.tcl
```



Filename: vivado_programming.png

10. Hardware Wiring & Bring-Up

This section details the physical connections required for each supported platform. Note that the 7-segment display is an external component that must be wired separately — it is not an on-board peripheral on these platforms.

10.1 Basys 3 Configuration

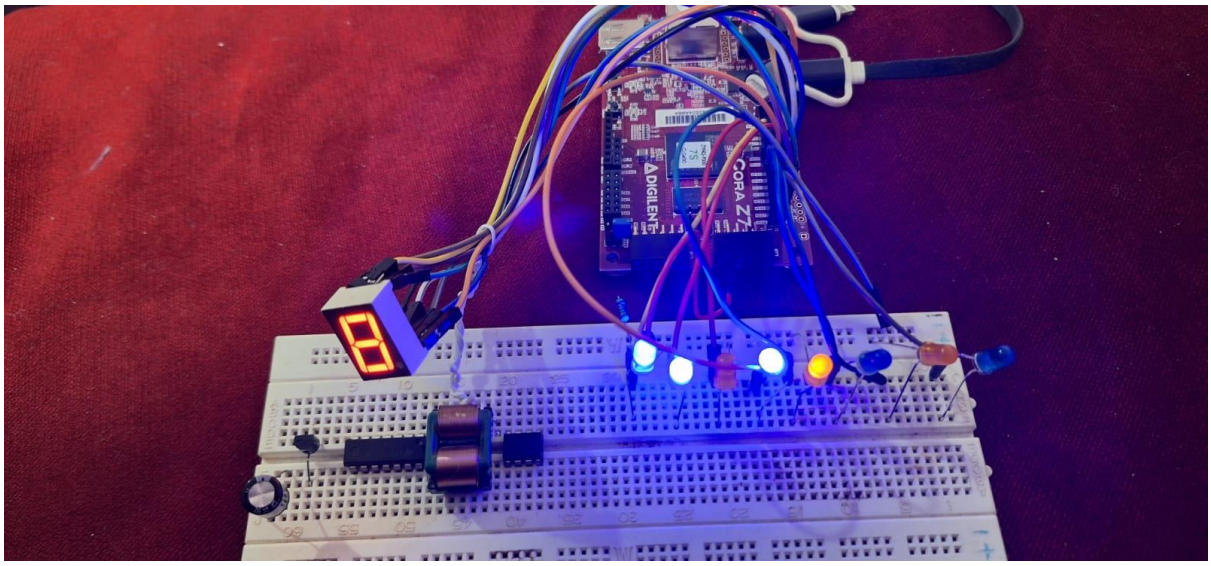
Signal	Board Connection
clk	CLK100MHZ (on-board 100 MHz oscillator)
btn_start	BTNU (active high)
btn_reset	BTNC (active high)
leds[7:0]	LD7...LD0 (directly on-board LEDs)
sevenseg[7:0]	External — wire to Pmod connector JA or JB

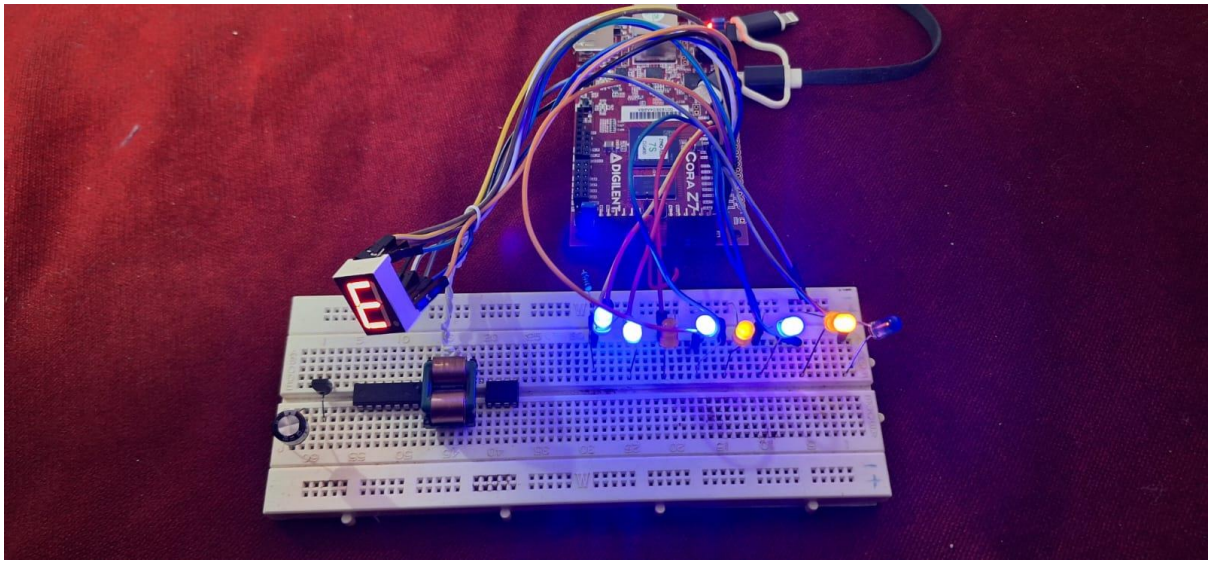
10.2 Cora Z7-07S Configuration

The Cora Z7 has limited on-board I/O, requiring external wiring for both LEDs and 7-segment display:

Signal	External Wiring
clk	125 MHz on-board oscillator
leds[7:0]	External — wire to IO26...IO33 on header
sevenseg[7:0]	External — wire to IO0...IO7 on header

INSERT CORA Z7 PHOTO HERE





10.3 DE0-Nano Configuration (Intel FPGA)

Signal	Board Connection
clk	CLOCK_50 (on-board 50 MHz oscillator)
reset	KEY0 (active low — requires inversion in HDL)
leds[7:0]	LED0...LED7 (directly on-board)
sevenseg[7:0]	External — wire to GPIO header

10.4 External 7-Segment Display Wiring

Since the 7-segment display is not integrated on these boards, it must be connected externally using the following typical wiring:

- Segments a-g: Connect to sevenseg[6:0] through appropriate current-limiting resistors (220Ω typical)
- Decimal point: Connect to sevenseg[7] if used
- Common cathode/anode: Connect to GND or VCC depending on display type
- Power: 3.3V from FPGA board Pmod or header pins

11. Expected Hardware Behavior

Once programmed and properly wired, the CPU8 demonstrates correct operation through the following sequence:

11. Power on the board — LEDs may show random state
12. Press RESET — Accumulator clears, LEDs show 00000000
13. Press START — Execution begins, counting starts
14. Observe LEDs counting: 1 → 2 → 3 → ... → 255 → 0 (wraps around)
15. If 7-segment display is connected, it shows hexadecimal value of accumulator

The counting speed depends on the clock divider configuration. At full 100 MHz speed, the count would be too fast to observe visually — a clock divider is typically used to slow the visible counting rate.

12. Common Issues & Solutions

Issue	Solution
LEDs remain off	Release reset button; verify bitstream programmed successfully
LEDs static (not counting)	Check clock constraint in XDC file; verify START button pressed
Count too fast to see	Enable clock divider in top module; adjust divider ratio
DE0 reset inverted	KEY0 is active-low; add inverter or modify HDL reset logic
7-segment not displaying	Verify external wiring; check resistor values; confirm display type (common cathode/anode)
External LEDs dim	Reduce current-limiting resistor value (ensure within GPIO current spec)