FPGA CPU “RG Sonic32”

v0 Architecture & RTL Specification

Author: Ryan Gaffere

Version: v0

Document Version: v1

Last Updated: January 20th, 2026

This document covers version 0 exclusively, which is a barebones implementation of the RTL. If you wish to see the modules work together or the ISA specifications. Please start with version 1.

Table of Contents

Introduction

1.1. Project Overview

The goal of this project is to design and implement a custom CPU on an Artix-7 FPGA. In its complete form, this processor is intended to support:

- Execution of bare-metal software
- Peripheral interfaces such as UART and basic video output
- Running complex software workloads (including DOOM)
- Hardware acceleration for AI-oriented workloads

Version v0 represents the foundational stage of this effort. In this version, all modules are implemented explicitly and manually, without reliance on FPGA vendor IP cores. Future versions will transition select components, such as system memory, to on-chip BRAM and external memory, accessed through dedicated and potentially proprietary memory interface logic.

At this stage, the project should be viewed as a structural and architectural groundwork, rather than a complete, integrated system. The modules implemented in v0 are not intended to operate together. Instead, each module exists in isolation, with its internal logic designed, implemented, and validated independently where appropriate.

An analogy for v0 is a piece of furniture before assembly: every component has been individually crafted and inspected, but no attempt has yet been made to connect them into a functional whole. As a result, reviewers may notice characteristics indicative of early development, such as limited operation support, reduced data widths (such as an 8-bit memory model), and placeholder assumptions. These decisions are deliberate and reflect the exploratory and educational nature of this version.

The primary objective of v0 is to build and verify a deep understanding of CPU microarchitecture by constructing each fundamental block from first principles. This approach prioritizes clarity, correctness, and insight over performance or completeness, and serves as a stable base upon which later integrated and optimized versions will be built.

The processor is named RG Sonic32:

- **RG** : my initials
- **Sonic** : referencing the project's long-term goal of accelerating AI workloads
- **32** : denoting the intended 32-bit architecture of the final design

Version v0 establishes the conceptual and technical foundation for all subsequent iterations of the RG Sonic32 CPU.

1.2. Design Goals

The primary goal of version v0 is to ensure that each module functions correctly in isolation, independent of any other subsystem. Each design is developed, implemented, and validated as a standalone unit, with correctness and clarity taking precedence over integration or performance.

From a higher-level perspective, these modules are intended to serve as robust architectural templates for the eventual implementation of a custom instruction set architecture (ISA). Care is taken to define clean interfaces, explicit control behavior, and predictable data flow so that each module can be extended, refined, or replaced without impacting the overall system design.

A key objective of v0 is to preserve strong modularity. This modular approach enables incremental development, simplifies verification, and allows future versions to integrate vendor-specific resources (such as BRAM or external memory controllers) without altering the fundamental CPU architecture.

In summary, v0 prioritizes:

- Independent functional correctness
- Architectural clarity and extensibility
- ISA-agnostic module design
- Long-term scalability through modularity

1.3. Target Hardware

This project targets the Alchitry Au V2 FPGA development board, based on the Xilinx Artix-7 family. The Au V2 provides a strong balance of logic resources, high-speed I/O, onboard memory, and power capability, making it well-suited for custom CPU development and future accelerator work.

FPGA

- **Device:** XC7A35T-2FTG256I
- **Notes:** Speed and temperature grade upgrade over the Au V1
- **Architecture:** Xilinx Artix-7

Clocking

- **Onboard oscillator:** 100 MHz

I/O Capabilities

- **Total I/O pins:** 104 (broken out across two headers)

- **Triple-voltage I/O:** 20 pins configurable for 3.3 V, 2.5 V, or 1.8 V
    - 18 of these are LVDS_25 capable
- **Differential I/O:**
    - 44 pins routed as 100 Ω differential pairs
    - Includes 18 triple-voltage pins
- **Single-ended I/O:**
    - Remaining I/O routed as ~50 Ω single-ended
    - ~90 Ω when used as differential pairs
- **Bank-specific features:**
    - Bank B includes two 1.35 V pins
    - 8 differential pairs can be used as XADC inputs (0–1 V range)
    - All differential pairs support LVDS_25 inputs except three pairs on Bank B
- **Default I/O voltage:** 3.3 V

Control and User I/O

- **Control header:**
    - 8 I/O pins also connected to onboard LEDs
    - 1 I/O pin also connected to the onboard reset button
- **User indicators:**
    - 8 general-purpose LEDs
    - 1 pushbutton (typically used as reset)

Memory

- **External memory:**
    - 256 MB DDR3L @ 800 Mb/s (400 MHz clock)
- **Configuration memory:**
    - 32 Mbit configuration flash

Analog and XADC

- Dedicated analog voltage inputs
- XADC input range: 0–1 V
- 8 differential input pairs routed to XADC

USB and Debug Interfaces

- **USB interface:** FT2232HQ
    - USB → JTAG
    - USB → UART (up to 12 Mbaud)

Power

- **Input voltage:** 5–12 V

- **Onboard power rails:**
  - 3.3 V @ 4 A (I/O)
  - 2.5 V @ 500 mA (triple-voltage pins, derived from 3.3 V)
  - 1.0 V @ 4 A (VCCINT)
  - 1.8 V @ 1.2 A (VCCAUX, triple-voltage pins)
  - 1.35 V @ 1.2 A (DDR3L)
  - 1.8 V @ 200 mA (analog)

Expansion

- **QWIIC connector:**
  - -Shares pins with Bank B

Architectural Overview

Version 0 of the RG Sonic32 CPU is structured around three primary architectural components at the highest level: the Memory Module, the Arithmetic Logic Unit (ALU), and the Control Unit.

The **Memory Module** is responsible for data storage and retrieval. In Version 0, it serves both as the primary data memory. Its implementation is intentionally simplified to support early validation and module-level testing rather than full system integration.

The **Arithmetic Logic Unit (ALU)** performs all computational operations defined by the instruction stream. In this version, the ALU supports single-cycle execution of basic arithmetic and logical operations. More complex multi-cycle operations, such as multiplication and division, are explicitly excluded in Version 0 to maintain architectural clarity and minimize control complexity during early development.

The **Control Unit** orchestrates CPU operation by sequencing instruction execution and coordinating interactions between the ALU and the Memory Module. It is responsible for issuing control signals, directing data flow within the datapath, and managing read/write operations to memory. Together, the ALU and Memory Module form the datapath, while the Control Unit provides the decision-making logic that governs datapath behavior.

At a lower level, Version 0 also includes a dedicated **Program Counter (PC) module**, implemented separately as `pc.v`. The Program Counter tracks the address of the current instruction and updates its value each cycle, either through sequential incrementation or control-directed jumps. This module enables the CPU to execute ordered instruction sequences and forms the basis for control-flow mechanisms in later versions.

The decision to implement the Program Counter as an independent module reflects the **modular design philosophy** of the project. In Version 0, all components are intentionally

developed and validated in isolation, with clear functional boundaries. This approach prioritizes architectural understanding, testability, and future extensibility over immediate end-to-end execution.

RTL Module Specification

3.1 Program Counter (pc.v)

The Program Counter (PC) module maintains the address of the current instruction in instruction memory. It updates the instruction address each clock cycle, enabling sequential instruction execution and control-directed changes in program flow.

In Version 0, instruction memory is not yet implemented; however, the PC module is designed to interface cleanly with a future instruction memory subsystem.

Interface

Signal	Type	Width	Description
PC_W	Parameter	User-defined	Width of the program counter.
clk	Input	1 bit	Rising-edge system clock.
reset	Input	1 bit	Synchronous reset; sets PC to zero.
pc_we	Input	1 bit	Program counter write enable.
pc_next	Input	PC_W bits	Next program counter value.
pc	Output	PC_W bits	Current program counter value.

Operation Description

The Program Counter is a single-cycle, synchronous register. On each rising edge of the clock, the PC updates its stored value based on control inputs:

- If reset is asserted, the PC is set to zero, regardless of other control inputs.
- If pc_we is asserted, the PC loads the value provided on pc_next.
- Otherwise, the PC retains its current value.

Design Notes

- The PC module is intentionally minimal and does not implement instruction fetch or address translation.
- Instruction addressing granularity is assumed to be word-based in Version 0.
- In future versions, the PC logic is expected to evolve to support byte addressing, branching, and more advanced control-flow mechanisms.

3.2 Memory Module (mm.v)

The Memory Module provides temporary data storage for the CPU datapath. In Version 0, it implements a simple multi-port memory abstraction supporting two asynchronous read ports and one synchronous write port. This module is intended for functional validation and architectural exploration rather than performance optimization.

Interface

Signal	Type	Width	Description
WORD_W	Parameter	User-defined	Width of a memory word.
ADDR_W	Parameter	User-defined	Width of a memory address.
waddr	Input	ADDR_W bits	Address of data to be written.
wdata	Input	WORD_W bits	Data to be written.
we	Input	1 bit	Write enable.
clk	Input	1 bit	Rising-edge system clock.
raddr0	Input	ADDR_W bits	Address for read port 0.
raddr1	Input	ADDR_W bits	Address for read port 1.
rdata0	Output	WORD_W bits	Data output for read port 0.
rdata1	Output	WORD_W bits	Data output for read port 1.

Operation Description

The Memory Module implements a two-read, one-write (2R1W) memory interface with the following behavior:

- Writes are synchronous and occur on the rising edge of the clock.
    - If we is asserted, the memory location addressed by waddr is updated with wdata.
- Reads are asynchronous.
    - The outputs rdata0 and rdata1 continuously reflect the contents of the memory locations addressed by raddr0 and raddr1, respectively.

To ensure deterministic behavior, read-after-write conflicts are explicitly handled:

- If a read address matches the active write address during a write cycle, the corresponding read output reflects the value of wdata.

Design Notes

- The memory implementation is intentionally simple and not optimized for FPGA block RAM inference.
- Parameter values in Version 0 are provisional and chosen to support early module-level validation.
- Future versions are expected to adopt a fixed 32-bit word width, byte addressing, and a more realistic memory hierarchy, leveraging on-chip BRAM or external memory.
- This module does not implement instruction memory, caching, or access arbitration.
- In future versions, a standalone register file module (24–32 registers) will be introduced to replace this usage.

3.3 Arithmetic Logic Unit (alu.v)

The Arithmetic Logic Unit (ALU) performs arithmetic, logical, and shift operations on two datapath operands. In addition to producing a data result, the ALU generates a set of condition flags that reflect properties of the computed result. All supported operations in Version 0 execute with a single-cycle latency.

Interface

Signal	Type	Width	Description
WORD_W	Parameter	User-defined	Width of data input word.
OP_W	Parameter	User-defined	Opcode width; Determines the number of supported operations.
FLAG_N	Parameter	NA	Negative flag bit index (Value = 3).
FLAG_Z	Parameter	NA	Zero flag bit index (Value = 2).
FLAG_C	Parameter	NA	Carry / No-borrow flag bit index (Value = 1).
FLAG_V	Parameter	NA	Signed Overflow flag bit index (Value = 0).
d_in0	Input	WORD_W bits	Datapath operand 0.
d_in1	Input	WORD_W bits	Datapath operand 1.
clk	Input	1 bit	Rising-edge system clock.
op_code	Input	OP_W bits	Operation code.
d_out	Output	WORD_W bits	Result of ALU Operation.

flags	Output	4 bits	Condition flags {N, Z, C, V}

Operation Description

The ALU computes results combinationally based on d_in0, d_in1, and op_code. The computed result and associated condition flags are registered on the rising edge of the clock, producing a one-cycle latency from input to output.

Supported operations in Version 0 include basic arithmetic, bitwise logical operations, and logical and arithmetic shifts. Shift operations use the lower bits of d_in1 to determine the shift amount. Condition flags are updated on every operation:

- **N (Negative):** Set when the result is negative (MSB = 1)
- **Z (Zero):** Set when the result equals zero
- **C (Carry / No-borrow)**: Indicates carry-out for addition and no borrow for subtraction
- **V (Overflow):** Indicates signed overflow for arithmetic operations

Design Notes

- The ALU is intentionally limited to single-cycle operations and does not support multi-cycle functions such as multiplication or division.
- Parameter values in Version 0 are provisional and selected for early functional validation.
- Future versions are expected to support 32-bit data width, expanded operation sets, and tighter integration with the control unit and register architecture.

3.4 Control Unit (cu.v)

The Control Unit sequences datapath operations to execute instructions and produce the intended architectural behavior. It is implemented as a finite state machine (FSM) that generates control signals for the Program Counter, Memory Module, ALU, and Register File.

In Version 0, the Control Unit is intentionally minimal and serves primarily as a structural placeholder for future ISA-driven control logic.

Interface

Signal	Type	Width	Description
WORD_W	Parameter	User-defined	Width of instruction word.
OPC_W	Parameter	User-defined	Opcode width; Determines the number of supported operations.

OPC_MSB	Parameter	WORD_W bits	Index of most significant opcode bit.
OPC_LSB	Parameter	WORD_W - OPC_W	Index of least significant opcode bit.
clk	Input	1 bit	Rising-edge system clock.
reset	Input	1 bit	Synchronous reset.
ir	Input	WORD_W bits	Instruction register.
flags	Input	4 bits	Condition flags {N, Z, C, V}
mem_ready	Input	1 bit	Memory wait state indicator.
pc_we	Output	1 bit	Program Counter write enable.
pc_next_sel	Output	2 bits	PC next-value selector.
ir_we	Output	1 bit	Instruction Register write enable.
mem_we	Output	1 bit	Memory write enable.
mem_re	Output	1 bit	Memory read enable.
mem_addr_sel	Output	1 bit	Memory address source selector.
alu_op	Output	OPC_W bits	ALU operation code.
alu_a_sel	Output	2 bits	ALU operand A selector.
alu_b_sel	Output	2 bits	ALU operand B selector.
reg_we	Output	1 bit	Register File write enable.
wb_sel	Output	2 bits	Writeback source selector.

Operation Description

The Control Unit is implemented as a five-state finite state machine:

- **FETCH:** Initiates instruction fetch and updates the program counter.
- **DECODE:** Decodes the opcode field from the instruction register (ir[OPC_MSB:OPC_LSB]) and prepares control signals for execution.
- **EXEC:** Performs the core ALU operation or address calculation.
- **MEM:** Handles memory read or write operations when required.
- **WB:** Writes results back to the register file.

State transitions occur synchronously on the rising edge of the clock. Although the Control Unit includes support for memory wait states via mem_ready, this signal is assumed to be asserted in Version 0 and does not stall execution.

Design Notes

- The Control Unit logic is intentionally rudimentary, as its detailed behavior is tightly coupled to the instruction set architecture (ISA).
- The ISA for Version 0 is not defined; therefore, opcode meanings and instruction semantics are placeholders.
- Full ISA definition, branching behavior, and memory stall handling are deferred to Version 1.
- This module establishes the structural control framework required for future CPU integration.

FSM Table

State	Purpose	Key Signals Asserted
FETCH	Fetch instruction, advance PC.	pc_we, mem_re, ir_we
DECODE	Decode opcode, prep operands.	none (combinational decode)
EXEC	Execute ALU operation.	alu_op, alu_a_sel, alu_b_sel
MEM	Perform data memory access.	mem_re or mem_we
WB	Write result back.	reg_we, wb_sel

Development Roadmap

4.1 Phase 1: Logic Implementation (Skeleton Construction)

Scope:

- Define and implement core architectural modules (ALU, Control Unit, Memory, etc.)
- Emphasis on structural correctness and signal behavior
- Modules are implemented and tested in isolation
- No system-level integration is assumed

Notes:

- Parameter values, bit widths, and supported operations may be simplified
- Certain architectural decisions are intentionally deferred to later phases

4.2 Phase 2: Verification & Simulation (Testbenches)

Scope:

- Develop directed and self-checking testbenches
- Validate functional correctness of individual modules
- Verify corner cases, timing assumptions, and control behavior
- Establish confidence prior to system integration

Status:

- Limited module-level tests exist in v0
- Full verification is deferred to v1

4.3 Phase 3: Prototype Deployment on FPGA

Scope:

- Integrate modules into a complete system
- Synthesize and deploy onto the Artix-7 FPGA
- Validate clocking, reset behavior, and real hardware timing
- Prepare for peripherals (UART, memory interfaces, video output)

Status:

- Not applicable to v0
- Targeted for later versions

Appendix

Appendix A. Testbench Outputs

Memory Module Testbench (tb_mm.v)

Figure C.1: Vivado Simulation console output from execution of the memory module testbench (tb_mm.v). The testbench verifies basic write/read functionality, dual asynchronous read behavior, bulk memory initialization, and write-first semantics during read/write address collisions. All tests complete successfully, as indicated by the final *ALL TESTS PASSED* message.

```
---- TEST: basic writes then dual reads ----
PASS @81000: r0[0]=a5 | r1[1]=3c
PASS @91000: r0[15]=ff | r1[0]=a5
---- TEST: bulk fill ----
PASS @421000: r0[5]=0f | r1[10]=1e
PASS @431000: r0[0]=00 | r1[15]=2d
---- TEST: write-first collision behavior ----
INFO pre-edge @461000: r0=22 (expected old=22), r1=09
PASS write-first @466000: r0=22
ALL TESTS PASSED
$finish called at time : 470 ns : File "C:/Users/rgaff/Desktop/CPU
project/CPU project.srcs/sources_1/new/tb_mm.v" Line 130
INFO: [USF-XSim-96] XSim completed. Design snapshot 'tb_mm_behav'
loaded.
INFO: [USF-XSim-97] XSim simulation ran for 1000ns
```

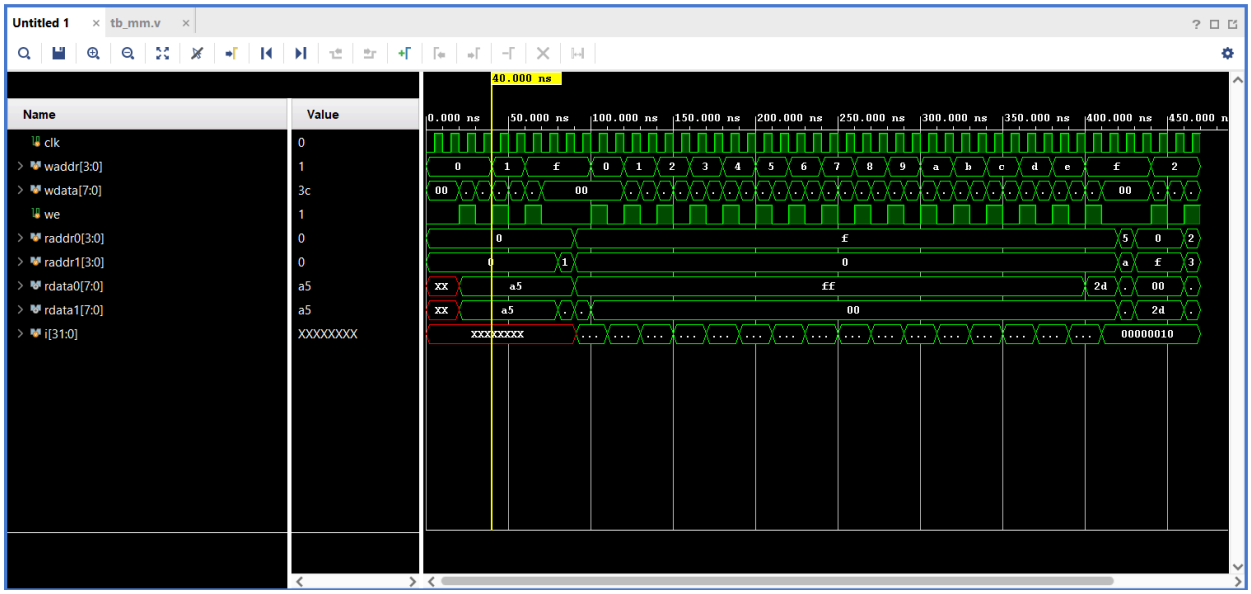

Figure C.2: Complete simulation waveform for the memory module testbench (tb_mm.v). The waveform illustrates synchronous write behavior, dual asynchronous read ports, sequential address writes during bulk initialization, and correct readback of stored values. The simulation includes verification of read/write timing and culminates in a write-first read/write collision scenario, confirming deterministic memory behavior across all tested conditions.

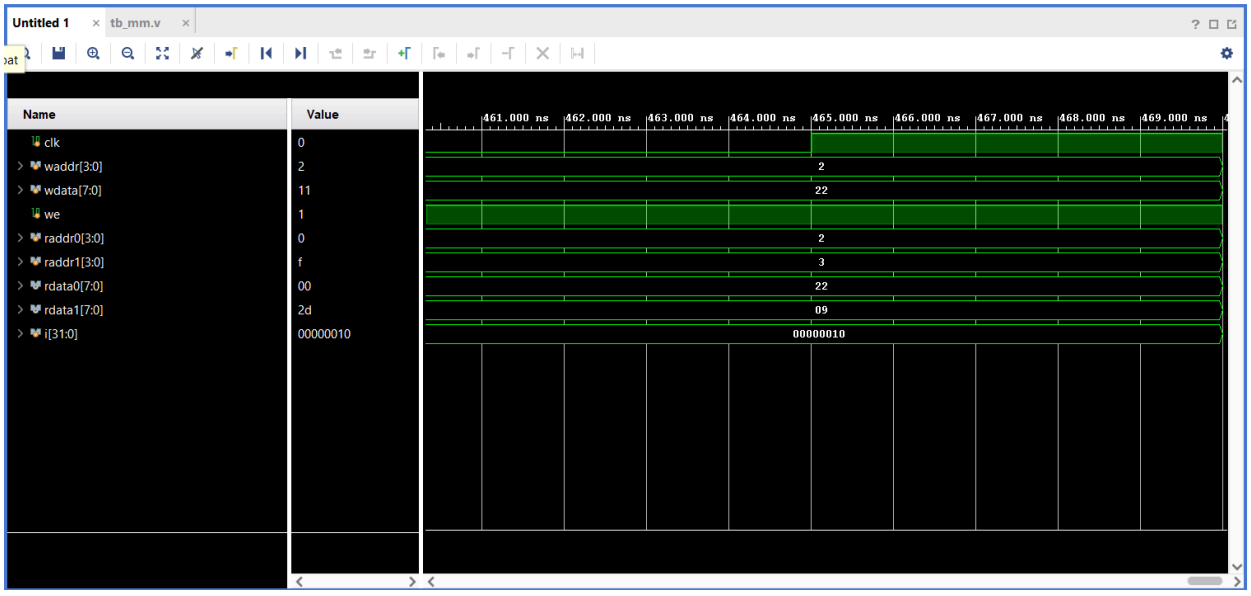

Figure C.3: Simulation waveform demonstrating write-first behavior in the memory module. A write and read target the same address during a single clock cycle; immediately after the rising clock edge, the read data reflects the newly written value, confirming write-first semantics. A simultaneous read from a different address remains unaffected.

ALU Testbench (tb_alu.v)

Console Output

Figure C.4: Vivado XSim console output from execution of the ALU testbench (tb_alu.v). The testbench exercises all supported ALU operations and validates both result and condition flag outputs. Each operation reports a successful pass, and the testbench terminates with *ALL TESTS PASSED*.

```
PASS @26000  op=0  a=0x1 b=0x1 | out=0x2 flags=0000
PASS @36000  op=0  a=0xff b=0x1 | out=0x0 flags=0110
PASS @46000  op=0  a=0x7f b=0x1 | out=0x80 flags=1001
PASS @56000  op=1  a=0x5 b=0x3 | out=0x2 flags=0000
PASS @66000  op=1  a=0x0 b=0x1 | out=0xff flags=1010
PASS @76000  op=1  a=0x80 b=0x1 | out=0x7f flags=0001
PASS @86000  op=2  a=0xaa b=0xf | out=0xa flags=0000
PASS @96000  op=3  a=0xa0 b=0xf | out=0xaf flags=1000
PASS @106000  op=4  a=0xff b=0xf | out=0xf0 flags=1000
PASS @116000  op=5  a=0x0 b=0x0 | out=0xff flags=1000
PASS @126000  op=6  a=0x1 b=0x3 | out=0x8 flags=0000
PASS @136000  op=7  a=0x80 b=0x1 | out=0x40 flags=0000
PASS @146000  op=8  a=0x80 b=0x1 | out=0xc0 flags=1000
PASS @156000  op=9  a=0x12 b=0x34 | out=0x12 flags=0000
PASS @166000  op=10  a=0x12 b=0x34 | out=0x34 flags=0000
ALL TESTS PASSED.
$finish called at time : 166 ns : File "C:/Users/rgaff/Desktop/CPU
project/CPU project.srcs/sources_1/new/tb_alu.v" Line 213
INFO: [USF-XSim-96] XSim completed. Design snapshot 'tb_alu_behav'
loaded.
INFO: [USF-XSim-97] XSim simulation ran for 1000ns"
```

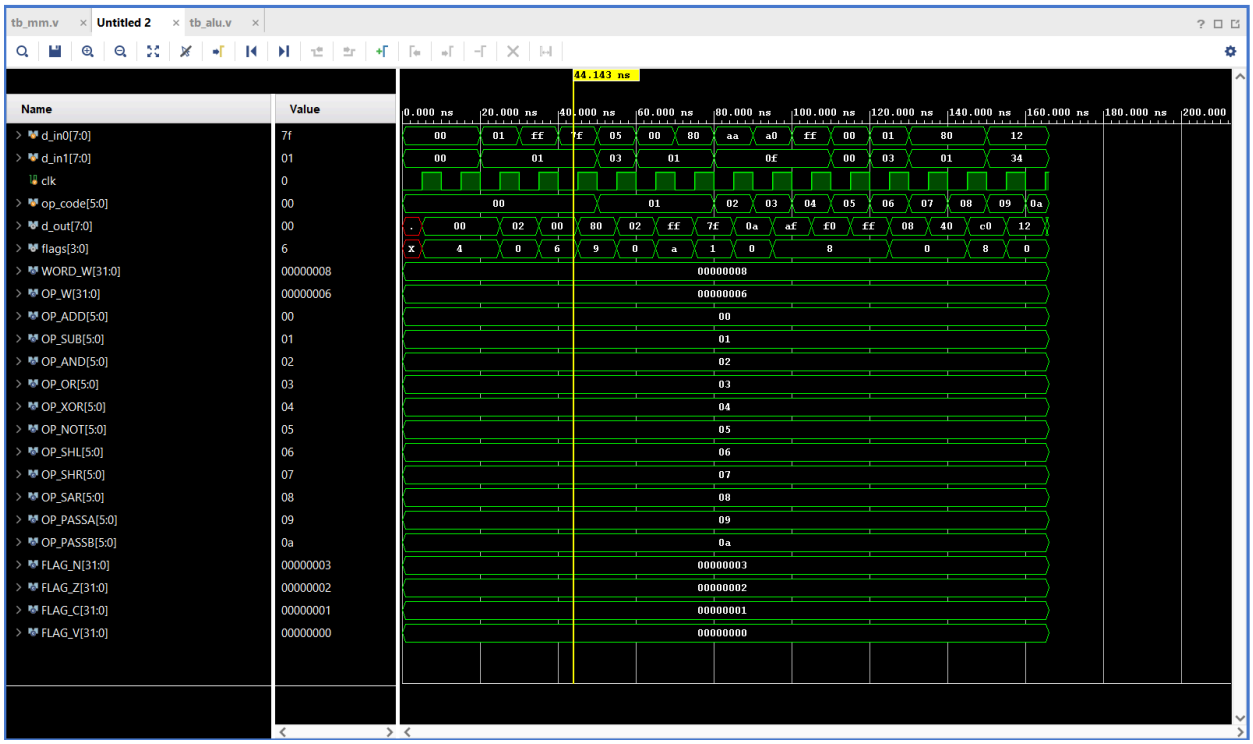

Figure C.5: Simulation waveforms for the ALU module showing opcode sequencing, input operands, registered outputs, and condition flags.
The waveforms demonstrate correct operation selection, stable registered results after clock edges, and proper generation of condition flags across all tested operations.

Appendix B. File Directory Structure

```
v0/
├── alu.v
├── cu.v
├── mm.v
├── pc.v
├── testbenches/
│   ├── tb_alu.v
│   └── tb_mm.v
└── docs/
    └── v0 Architecture & RTL Specification.pdf
```