**ELEC6234 – Embedded Processor Synthesis**

Peter Alexander

pa4g17

Electronic Engineering with Computer Systems

Dr Sasan Mahmoodi

**ABSTRACT:** A synthesisable n-bit picoMIPS processor was developed in SystemVerilog to implement a given algorithm in Software. The design has been verified in simulation and when synthesised on an FPGA.

1. **1. Introduction**

The goal of this assignment as given in [1] was to implement a synthesisable n-bit picoMIPS processor in SystemVerilog. An algorithm is provided that this processor must be programmed to complete. An 8-bit version of this processor should be synthesised on the DE1 FPGA development board.

The processor has been implemented successfully. Custom hardware was designed to implement I/O (see Section 1. 2. 1.). An assembler was written in Python to streamline software development. The design was verified using a suit of self-checking testbenches. The correct execution of the algorithm has been demonstrated in software (see Section 1. 2. 3.) and on the DE1 board (see Section 1. 4.).

1. **2.** **Overall architecture of the design**

Figure 2 shows the system block diagram for the final design.

1. **2. 1. I/O**

Each ALU input has been multiplexed with the register module output, SW[7:0], and {8{SW[8]}}. When Rd or Rs is set to 2 or 3, CPU-level logic sets the appropriate ALU input to SW[7:0] or {8{SW[8]}} respectively. This logic has been represented with the **Input Selector** block in Figure 1. The switches can be read as follows:

|  |
| --- |
| As Rd or Rs in a BEQ/BNE evaluation. This allows branching based on the value of SW[8] to be performed in a single cycle:  BEQ %0 %3 0 # If SW[8] == 0 Don’t increment PC |
| The MOV instruction, added to facilitate I/O, is used to place the value of SW[7:0] or SW[8] into any register.  MOV %4 %2 # Move SW[7:0] into register 4 |

The output of the ALU is connected to the LEDs so both the program result and the workings of the CPU can be observed.

1. **2. 2. Instruction Set and ALU functions Implemented**

Table 1 and Table 2 give the instructions and ALU functions implemented.

|  |  |  |
| --- | --- | --- |
| **Instruction** | **ALU Function** | **Description** |
| NOP | N/A | Do nothing |
| ADDI Rd Rs 9 | RADD | Rd = Rs + Imm |
| ADD Rd Rs | RADD | Rd = Rs + Rd |
| MULL Rd Rs | RMULL | Rd = lower half of Rs \* Rd (signed) |
| BEQ Rd Rs Imm | RSUB | PC = PC + Imm if Rd == Rs, else PC = PC + 1 |
| BNE Rd Rs Imm | RSUB | PC = PC + Imm if Rd != Rs, else PC = PC + 1 |
| MOV Rd Rs | RB | Rd = Rs |

Table : A table showing the instructions implemented

|  |  |
| --- | --- |
| **ALU Function** | **Description** |
| RA | alu\_result = a\_in |
| RB | alu\_result = b\_in |
| RADD | alu\_result = a\_in + b\_in |
| RSUB | alu\_result = a\_in – b\_in |
| RMULL | alu\_result = lower half of a\_in \* b\_in (signed) |

Table : A table showing the ALU functions implemented

1. **2. 3. Application software and Testbench**

Figure 3 shows the software implementing the algorithm given in [2]. Figure 1 shows part of the output of the ./rtl/picoMIPS4test\_stim.v testbench .

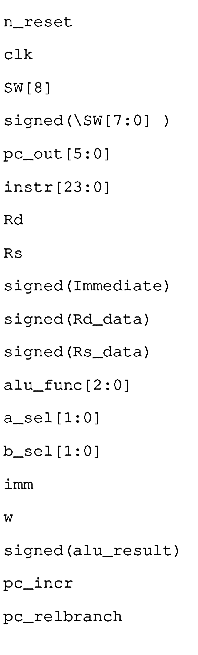
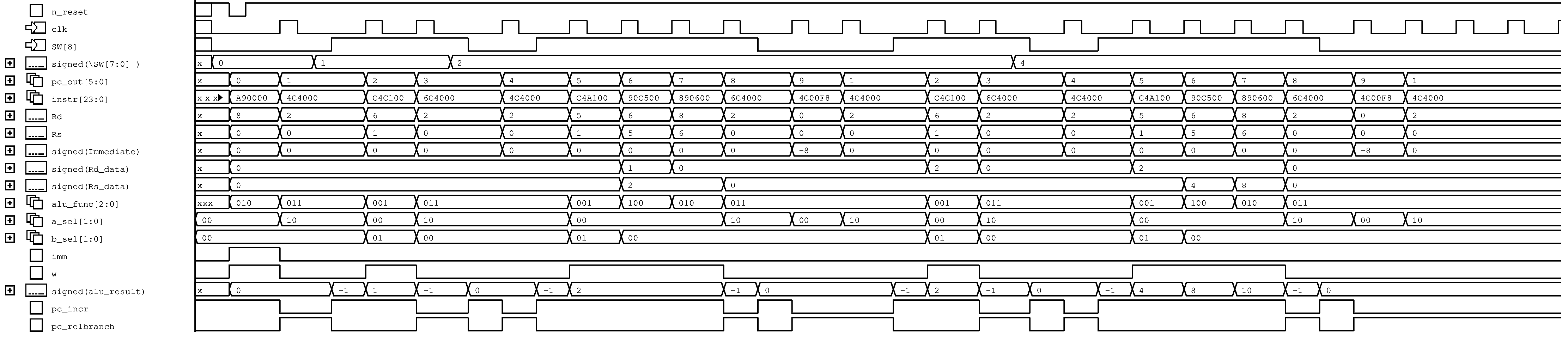


Figure : The second half of an application software test Inputs are read from SW[7:0] into registers on instructions 2 and 5, the result of the multiplication is given on instruction 6, and the result of the accumulation is give on instruction 7. R8 contained the value 2 after a from test.

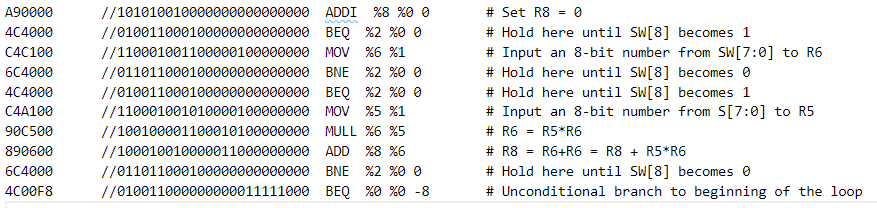
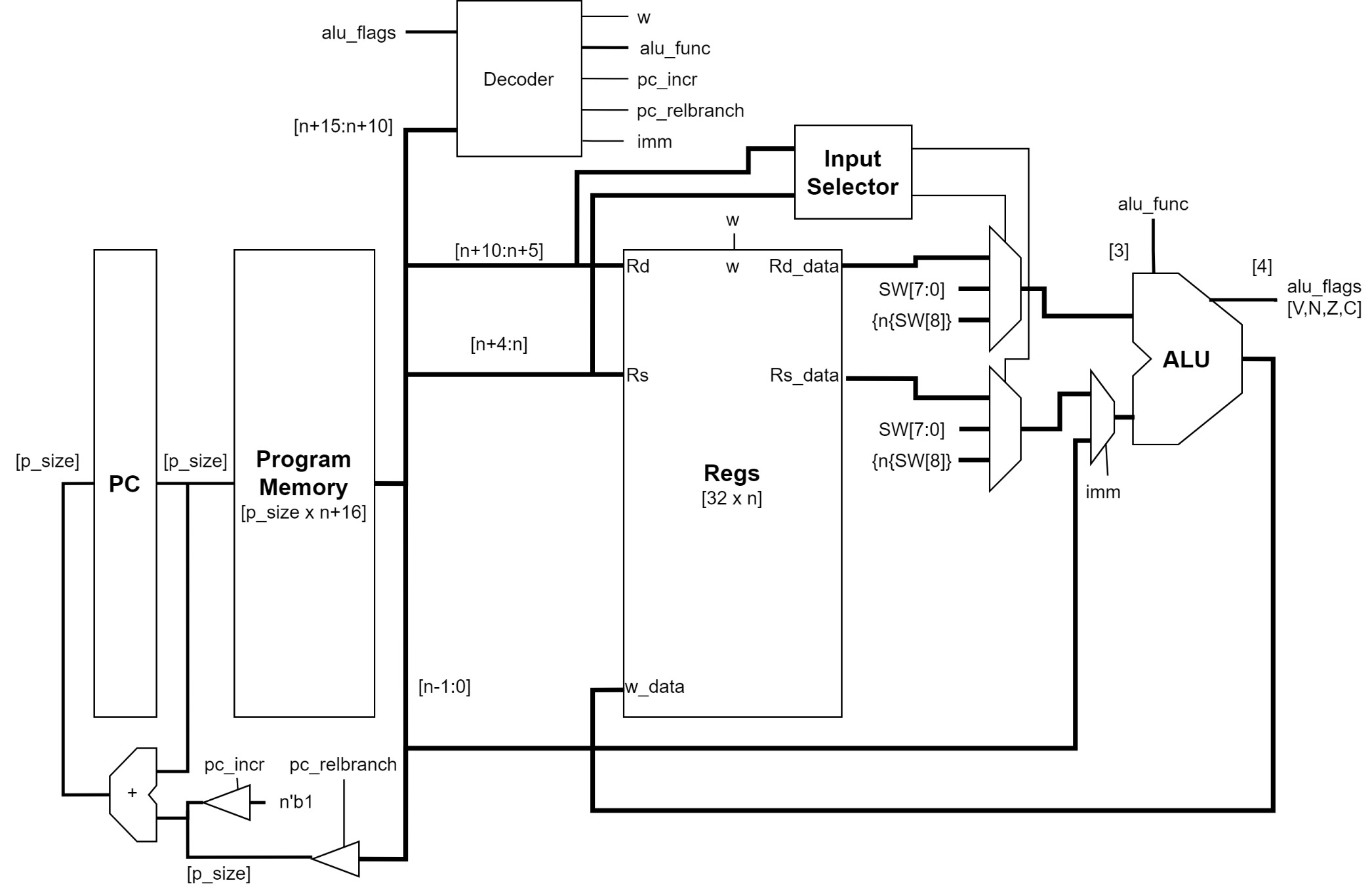
1. **3. Details of hardware blocks**



Figure : The system block diagram for the final design

Figure : The software implementing the spec algorithm

Each module was based on the example modules given in [2]. A self-checking testbench was written for each module.

**1 . 3. 1. PC**

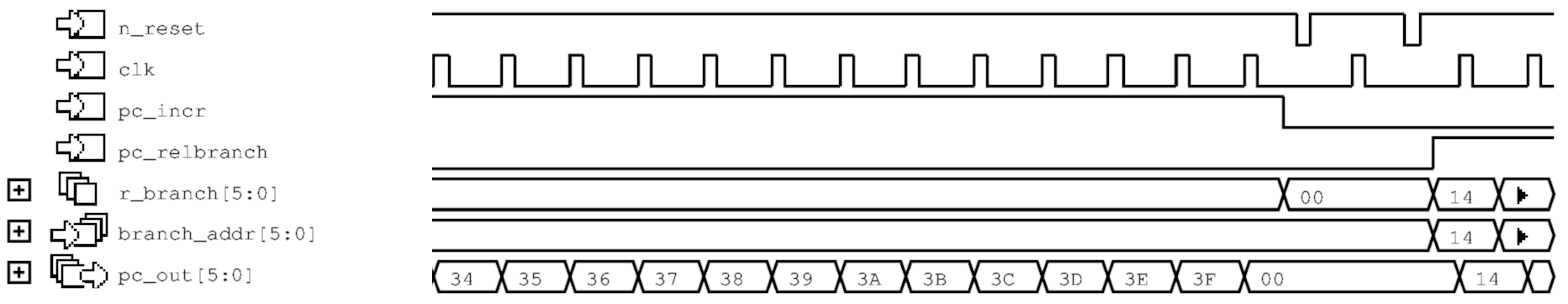
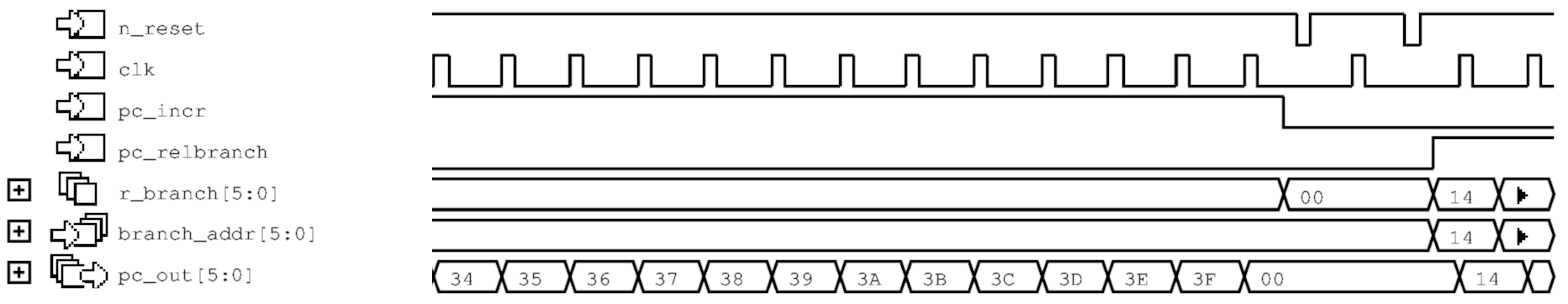
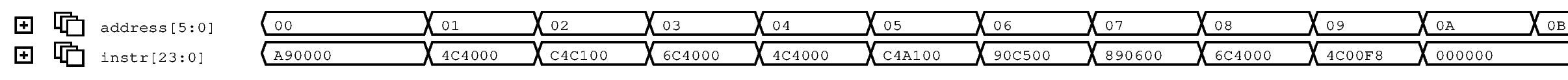
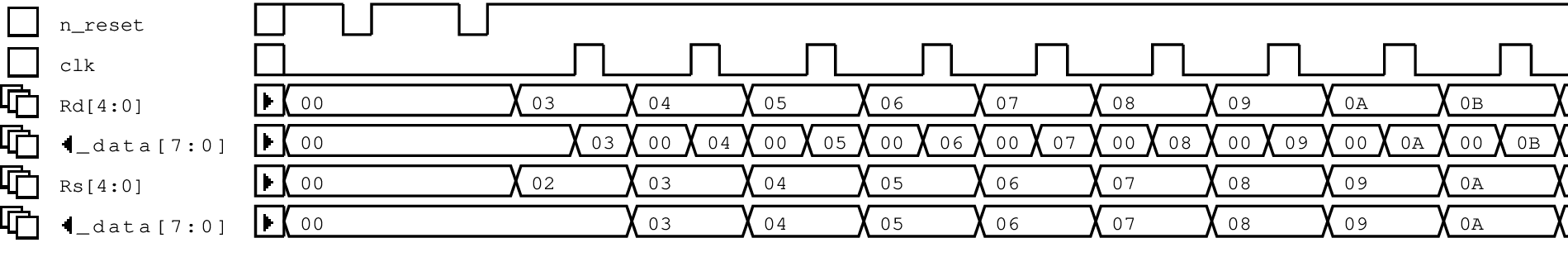


Figure : Figure Showing the end of the PC testbench. It increments the PC until it overflows then successfully performs a relative branch.

Absolute branching was removed to save resources.

Figure 4 shows the end of the testbench output

**

Figure : The beginning of the Program Memory testbench

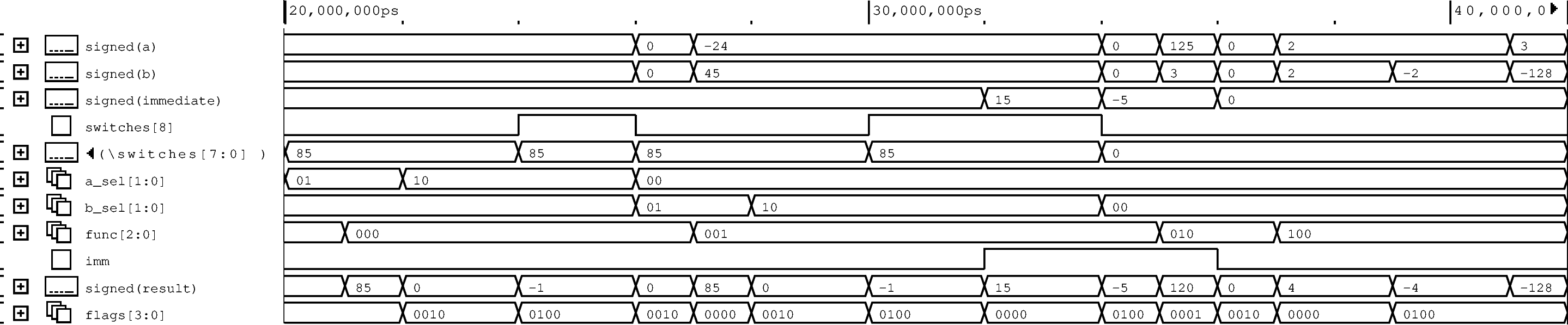
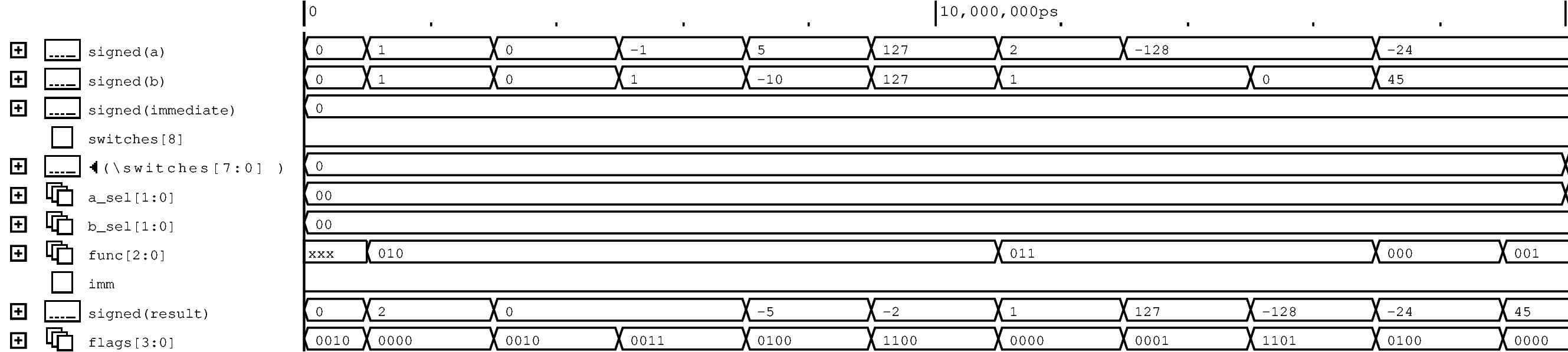


Figure : The second half of the ALU testbench

Figure : The first half of the ALU testbench



Figure : The beginning of the Register File testbench

**1 . 3. 2. Program Memory**

Figure 5 shows the beginning of the testbench output. It compares each Program Memory address with the corresponding line in prog.hex. In Figure 5, prog.hex contains the software in Figure 3.

**1 . 3. 3. Registers**

An asynchronous reset was added to aid verification.

The beginning of the testbench output is given in Figure 6. Each register from R3 has its number written to it (e.g.R3=3,R4=4,…). Rs observes the previous register to check none are overwritten.

**1 . 3. 4. ALU**

The full output for the self-checking ALU testbench can be found in Figure 7 and Figure 8. They have been annotated isolating each test and showing the success criteria.

**1 . 3. 5. Decoder**

The decoder and top-level CPU module were developed and verified simultaneously so it does not have a dedicated testbench. Instructions were added and verified individually using the basic\_cpu\_stim testbench.

1. **4. FPGA implementation**

**1 . 4. 1. Individual Module Synthesis**

Quartus projects were created for each module listed in Section 1. 3. Errors and warnings were fixed. Quartus RTL-level diagrams were checked. For example, it was checked that the multiplier and adders in the ALU had been correctly inferred and assigned to dedicated hardware.

**1 . 4. 2. Integrated Design Synthesis**

Thanks to the individual module testing, there were no errors in the first synthesis trial of the full design. A few warnings about incorrectly defined variable widths in the top-level module had to be fixed, however.

**1 . 4. 3. Synthesised System Testing**

To determine that that FPGA implementation was working correctly the tests performed in the ./rtl/picoMIPS4test\_stim.sv testbench were repeated. The correct outputs were observed, and the behaviour matched the simulated system.

1. **5. Conclusion**

Each element of the spec has been achieved. The processor can successfully run the algorithm given in [1] in simulation and when synthesised on a DE1. All required instructions have been implemented.

The programmed processor used the following resources on the FPGA:

|  |  |
| --- | --- |
| **ALMs** | 85 |
| **Registers** | 54 |
| **DSP Blocks** | 1 |

This project provided experience in verifying large SystemVerilog projects and in producing development tools and software for deeply embedded systems.

One extension to this project would be to set the immediate input to overwrite the Rd input on the ALU, rather than Rs. This would negate the need for the MOV instruction for reading from the inputs.

1. **6. References**

|  |
| --- |
|  |

|  |  |
| --- | --- |
| [1] | T. Kazmierski, “ELEC6234 Embedded Processors Notes,” 2021. |
| [2] | T. Kazmierski, “ELEC6234 Assignment 1 Brief,” 2021. |