

Single Cycle RISC-V Processor

Hao Wang

ECE505

10/25/2024

Introduction

A single cycle RISC-V processor was implemented in SystemVerilog. It supports a small subset of the 32-bit integer instructions. Test benches were designed for each major module of the processor and the entire processor was tested by running four distinct test programs. Additionally, the processor was synthesized and implemented for a Basys3 FPGA board.

Modules

The top module for the processor consists of a clock, reset, halt, and RAM debug ports. The ram debug port is necessary to prevent the synthesizer from optimizing away the entire design while giving it the freedom to optimize parts of the design. The module can also be parameterized to set the machine code for the rom. The module instantiates the following sub-modules, clock generator, ALU, ALU control, control unit, immediate generator, register file, rom, and ram. Additionally, it instantiates several multiplexers, and a clocked process to set the PC address. The source files for these modules are contained in the src directory. The appendix contains a block diagram of the top module.

The clock generator generates a 25MHz clock signal from a 100MHz clock signal and additionally provides a locked signal. The external reset is connected to the reset port. The reset for the processor is asserted when the external reset is asserted and or when the locked signal is not asserted.

The ALU module performs arithmetic operations. It accepts two 32-bit inputs and an opcode. It outputs the operation result, and a zero signal when the result is equal to zero. Supported operations are ADD, SLL, OR, XOR, AND, MUL, SUB, and EQ.

The ALU control module computes the ALU opcode from the instruction function bits. It accepts the function-7 and function-3 bits from the current instruction, and an opcode as inputs. The opcode selects between decoding between B-type, R-type, and I-type instructions. Pure addition is also selectable.

The control unit module decodes the current instruction opcode and outputs control signals which consists of the alu control opcode, isBranch, memRead, memToReg, memWrite, aluUseImm, regWrite, isJal, isJalr, and halt.

The immediate generator accepts the current instruction and decodes and outputs the full sign extended immediate value.

The register file is a memory module with two read ports, and one write port, with operations occurring on the positive edge of the clock. It is parameterized to hold 32 words, where each word is 32 bits. Address zero of the memory always reads as the value zero and is immutable.

The rom module is a read only memory module that is initialized with the program memory. It is parameterized with the memory file to preload, a word size of 32 bits, and a word count of 32 words. Operations occur on the positive edge of the clock.

The ram module is a single port read and write memory module. It is parameterized with a word size of 32 bits and a word count of 256 words. Operations occur on the negative edge of the clock.

Test Benches

The test benches for sub modules verify basic functionality, while the top test bench verifies the operation of the processor by running test programs and comparing the state of the ram and register file to precomputed states obtained from an online emulator. The source files for the test benches are contained in the src/tb directory. And the states for the top test bench are contained in the src/tb/TopTestVectors_pkg.sv file. Program machine code files are contained in the src/tb/programs directory.

The third program computes the factorial of a number. The assembly and machine code are listed below.

```
00c00513 // addi a0, x0, 12
00c000ef // jal ra, fact
00a02023 // sw a0, 0(x0)
0000007f // HALT
ff810113 // fact: addi sp, sp, -8
00112223 // sw ra, 4(sp)
00a12023 // sw a0, 0(sp)
fff50513 // addi a0, a0, -1
00051863 // bne a0, x0, else
00100513 // addi a0, x0, 1
00810113 // addi sp, sp, 8
00008067 // jalr x0, 0(ra)
fe1ff0ef // else: jal ra, fact
00050293 // addi t0, a0, 0
00012503 // lw a0, 0(sp)
00412083 // lw ra, 4(sp)
00810113 // addi sp, sp, 8
02550533 // mul a0, a0, t0
00008067 // jalr x0, 0(ra)
```

A screenshot of the register file and ram are shown below. Address zero in the register file corresponds to the X1 not X0
































>  [0][31:0]	00000008
>  [1][31:0]	000000fc
>  [2][31:0]	00000000
>  [3][31:0]	00000000
>  [4][31:0]	02611500
>  [5][31:0]	00000000
>  [6][31:0]	00000000
>  [7][31:0]	00000000
>  [8][31:0]	00000000
>  [9][31:0]	1c8cfc00
>  [10][31:0]	00000000
>  [11][31:0]	00000000
>  [12][31:0]	00000000
>  [13][31:0]	00000000
>  [14][31:0]	00000000
>  [15][31:0]	00000000
>  [16][31:0]	00000000
>  [17][31:0]	00000000
>  [18][31:0]	00000000
>  [19][31:0]	00000000
>  [20][31:0]	00000000
>  [21][31:0]	00000000
>  [22][31:0]	00000000
>  [23][31:0]	00000000
>  [24][31:0]	00000000
>  [25][31:0]	00000000
>  [26][31:0]	00000000
>  [27][31:0]	00000000
>  [28][31:0]	00000000
>  [29][31:0]	00000000
>  [30][31:0]	00000000

Figure 1. Program 3, register file.

































































		>  [27][31:0]	00000000
		>  [28][31:0]	00000000
		>  [29][31:0]	00000000
		>  [30][31:0]	00000000
		>  [31][31:0]	00000000
		>  [32][31:0]	00000000
		>  [33][31:0]	00000000
		>  [34][31:0]	00000000
		>  [35][31:0]	00000000
		>  [36][31:0]	00000000
		>  [37][31:0]	00000000
		>  [38][31:0]	00000000
		>  [39][31:0]	00000001
		>  [40][31:0]	00000034
		>  [41][31:0]	00000002
		>  [42][31:0]	00000034
		>  [43][31:0]	00000003
		>  [44][31:0]	00000034
		>  [45][31:0]	00000004
		>  [46][31:0]	00000034
		>  [47][31:0]	00000005
		>  [48][31:0]	00000034
		>  [49][31:0]	00000006
		>  [50][31:0]	00000034
		>  [51][31:0]	00000007
		>  [52][31:0]	00000034
		>  [53][31:0]	00000008
		>  [54][31:0]	00000034
		>  [55][31:0]	00000009
		>  [56][31:0]	00000034
		>  [57][31:0]	0000000a
		>  [58][31:0]	00000034
		>  [59][31:0]	0000000b
		>  [60][31:0]	00000034
		>  [61][31:0]	0000000c
		>  [62][31:0]	00000008
		>  [63][31:0]	00000000
>  [0][31:0]	1c8cfc00		
>  [1][31:0]	00000000		
>  [2][31:0]	00000000		
>  [3][31:0]	00000000		
>  [4][31:0]	00000000		
>  [5][31:0]	00000000		
>  [6][31:0]	00000000		
>  [7][31:0]	00000000		
>  [8][31:0]	00000000		
>  [9][31:0]	00000000		
>  [10][31:0]	00000000		
>  [11][31:0]	00000000		
>  [12][31:0]	00000000		
>  [13][31:0]	00000000		
>  [14][31:0]	00000000		
>  [15][31:0]	00000000		
>  [16][31:0]	00000000		
>  [17][31:0]	00000000		
>  [18][31:0]	00000000		
>  [19][31:0]	00000000		
>  [20][31:0]	00000000		
>  [21][31:0]	00000000		
>  [22][31:0]	00000000		
>  [23][31:0]	00000000		
>  [24][31:0]	00000000		
>  [25][31:0]	00000000		
>  [26][31:0]	00000000		

Figure 2. Program 3, ram.

The fourth program computes the sum of an array where the size of the array is at ram address 0, and the 32-bit elements of the array start at address 4. The elements 2, 6, 5, 1, and 7, are loaded as the test vector. X5 stores the sum of the elements, X10 is used as an index variable, X11 is used as an address variable, x12 is used to store the total number of elements, while x6 is used to store the current element. The following is the assembly and machine code for the program.

```
00000513 // addi x10, x0, 0
00400593 // addi x11, x0, 4
00000293 // addi x5, x0, 0
00002603 // lw x12, 0(x0)
00c50c63 // beq x10, x12, 24
0005a303 // lw x6, 0(x11)
006282b3 // add x5, x5, x6
00150513 // addi x10, x10, 1
00458593 // addi x11, x11, 4
fedff06f // jal x0, -20
0000007f // HALT
```

The register file and ram states after the program finishes are shown below. Again, the addresses in the register file are offset by 1.
































>  [0][31:0]	00000000
>  [1][31:0]	00000000
>  [2][31:0]	00000000
>  [3][31:0]	00000000
>  [4][31:0]	00000015
>  [5][31:0]	00000007
>  [6][31:0]	00000000
>  [7][31:0]	00000000
>  [8][31:0]	00000000
>  [9][31:0]	00000005
>  [10][31:0]	00000018
>  [11][31:0]	00000005
>  [12][31:0]	00000000
>  [13][31:0]	00000000
>  [14][31:0]	00000000
>  [15][31:0]	00000000
>  [16][31:0]	00000000
>  [17][31:0]	00000000
>  [18][31:0]	00000000
>  [19][31:0]	00000000
>  [20][31:0]	00000000
>  [21][31:0]	00000000
>  [22][31:0]	00000000
>  [23][31:0]	00000000
>  [24][31:0]	00000000
>  [25][31:0]	00000000
>  [26][31:0]	00000000
>  [27][31:0]	00000000
>  [28][31:0]	00000000
>  [29][31:0]	00000000
>  [30][31:0]	00000000

Figure 3. Program 4, register file.



















>  [0][31:0]	00000005
>  [1][31:0]	00000002
>  [2][31:0]	00000006
>  [3][31:0]	00000005
>  [4][31:0]	00000001
>  [5][31:0]	00000007
>  [6][31:0]	00000000
>  [7][31:0]	00000000
>  [8][31:0]	00000000
>  [9][31:0]	00000000
>  [10][31:0]	00000000
>  [11][31:0]	00000000
>  [12][31:0]	00000000
>  [13][31:0]	00000000
>  [14][31:0]	00000000
>  [15][31:0]	00000000
>  [16][31:0]	00000000
>  [17][31:0]	00000000

Figure 4. Program 4, ram

Synthesis

The processor was synthesized for the Basys3 target and achieved a maximum clock of 25MHz. The clock summary and timing analysis summary are shown below.

Name	Waveform	Period (ns)	Frequency (MHz)
▼ rawClk	{0.000 5.000}	10.000	100.000
clkfbout_ClockGenerator	{0.000 5.000}	10.000	100.000
sysClk_ClockGenerator	{0.000 20.000}	40.000	25.000

Figure 5. Clock summary from Vivado. The system clock has a frequency of 25MHz.

Setup	Hold	Pulse Width
Worst Negative Slack (WNS): 0.446 ns	Worst Hold Slack (WHS): 0.263 ns	Worst Pulse Width Slack (WPWS): 3.000 ns
Total Negative Slack (TNS): 0.000 ns	Total Hold Slack (THS): 0.000 ns	Total Pulse Width Negative Slack (TPWS): 0.000 ns
Number of Failing Endpoints: 0	Number of Failing Endpoints: 0	Number of Failing Endpoints: 0
Total Number of Endpoints: 691	Total Number of Endpoints: 691	Total Number of Endpoints: 130
All user specified timing constraints are met.		

Figure 6. Timing analysis summary from Vivado. No timing violations are detected.

The resource usage post synthesis is shown below.

Resource	Estimation	Available	Utilization %
LUT	717	20800	3.45
LUTRAM	48	9600	0.50
FF	32	41600	0.08
BRAM	1	50	2.00
DSP	3	90	3.33
IO	21	106	19.81

Table 1. Post synthesis resource utilization.

Additionally, implementation and bitstream generation was run. The resource usage post implementation is shown below.

Resource	Utilization	Available	Utilization %
LUT	656	20800	3.15
LUTRAM	44	9600	0.46
FF	32	41600	0.08
BRAM	1	50	2.00
DSP	3	90	3.33
IO	22	106	20.75
BUFG	2	32	6.25
MMCM	1	5	20.00

Appendix. Top Module Block Diagram

