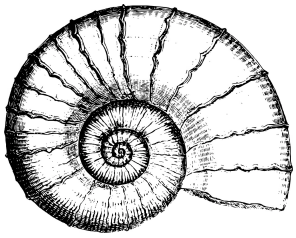
The Shell CPU

Component Implementation and Testing



Jonathan Rice Shelley II

October 21, 2020

Version 1.0

1 Contents

2 Component Implementation and Testing Overview

This document details the implementation and testing of each unique component in the Shell CPU. Two additional signals "inr" and "outvalue" have been added to the CPU. The two additional signals were added to view the contents of the register file. The VHDL for each module is to long to be listed and can be found with other documentation here.

3 Control Unit Verification

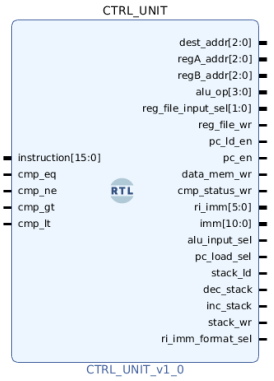


Figure 1: CPU Control Unit

The control unit is tested against all 23 instructions in the Shell ISA. The order of the instructions tested can be seen in the table below. The control unit output to the instructions is shown in Figure 2.

Instruction test order
hlt
add r1, r2, r3
sub r1, r2, r3
and r2, r2, r3
or r1, r2, r3
xor r1, r2, r3
nand r1, r2, r3
lw r1, r2
sw r1, r2
asr r1, r2
asl r1, r2
cmp r1, r2
jalr r1, r2
push r1
pop r1
lsp r1
addi r1, 5
subi r1, 5
lui r1, 5
beq 5
bne 5
bgt 5
blt 5

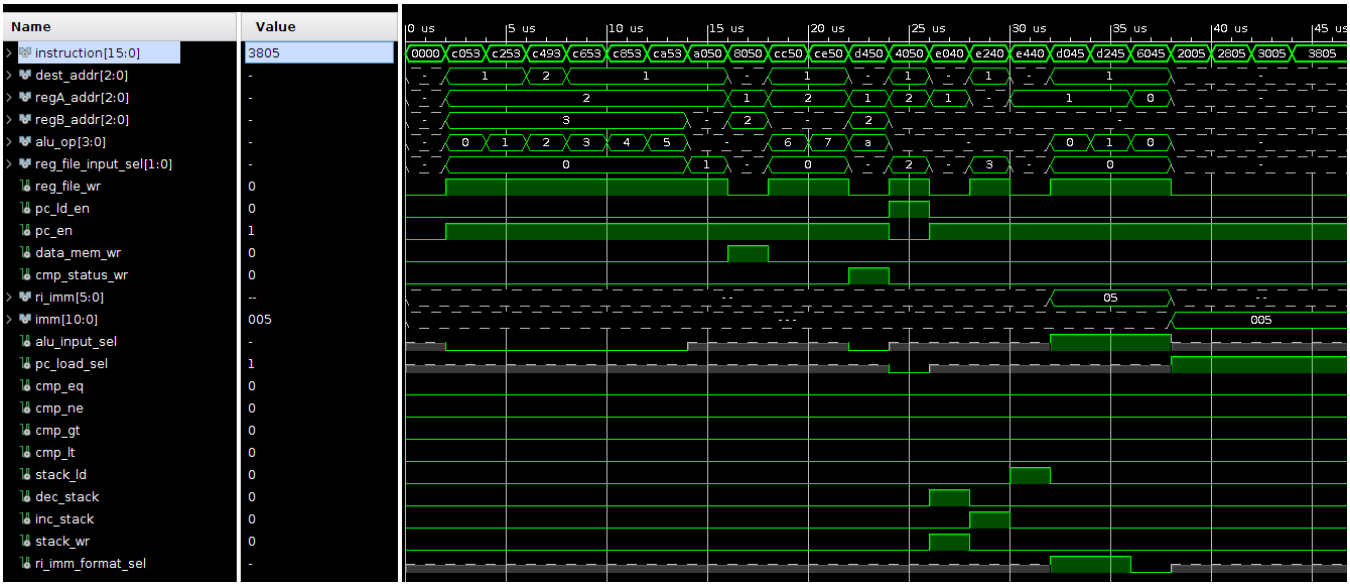


Figure 2: CPU Control Testbench

The outputs of the simulated control unit have been verified with the expected values listed in the Shell datapath documentation.

4 Component Testing

4.1 Arithmetic Logic Unit

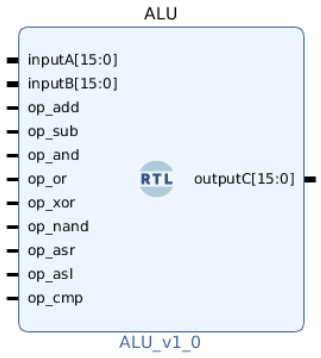


Figure 3: Arithmetic Logic Unit

Component Verification

The arithmetic logic unit or ALU performs arithmetic operations on data in the CPU. The block is entirely combinational logic and controlled by its "op" signals. A testbench is created to verify the functionality of the ALU. Figure 4 shows the ALU undertest performing each operation.

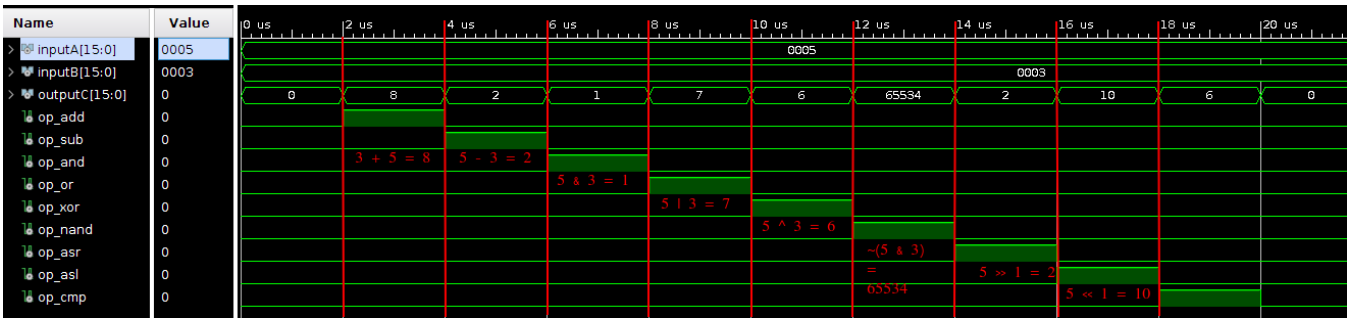


Figure 4: Arithmetic Logic Unit Testbench

4.2 Simple Adder

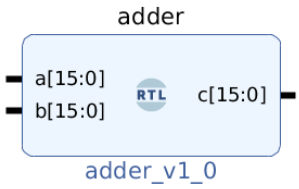


Figure 5: Simple Adder Unit

Component Verification

The simple adder block is used to sum values together within the CPU datapath. The block is used once to calculate PC + 1. The block is used a second time to find pc + imm for B-Type instructions. The testbench results for the simple adder can be seen in Figure 6. From the testbench it can be concluded that the simple adder properly adds the two decimal numbers.

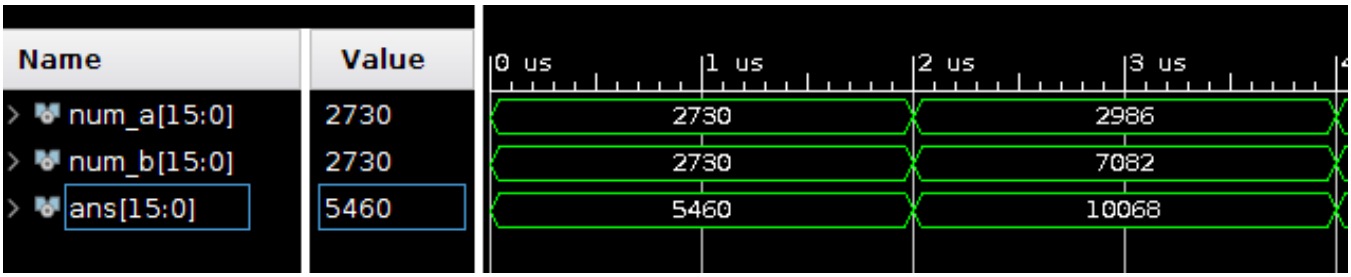


Figure 6: Simple Adder Unit Testbench

4.3 Arithmetic Logic Unit Controller

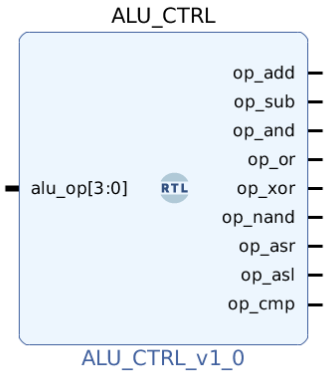


Figure 7: Arithmetic Logic Unit Controller

Component Verification

The arithmetic logic unit controller is used to interpret the ALU op codes from the CPU controller. The outputs of the arithmetic logic unit controller directly control what operation the ALU performs. The ALU control unit can be seen correctly decoding alu_op in Figure 8.

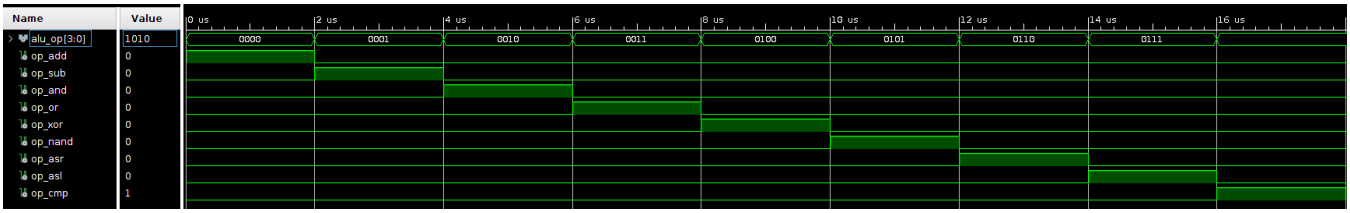


Figure 8: Arithmetic Logic Unit Controller Testbench Results

4.4 Two to One Multiplexer

Figure 9: Two to One Multiplexer

Component Verification

The two to one multiplexer is used throughout the datapath to mux 16-bit buses. The functionality of the mux is verified as seen in Figure 10.

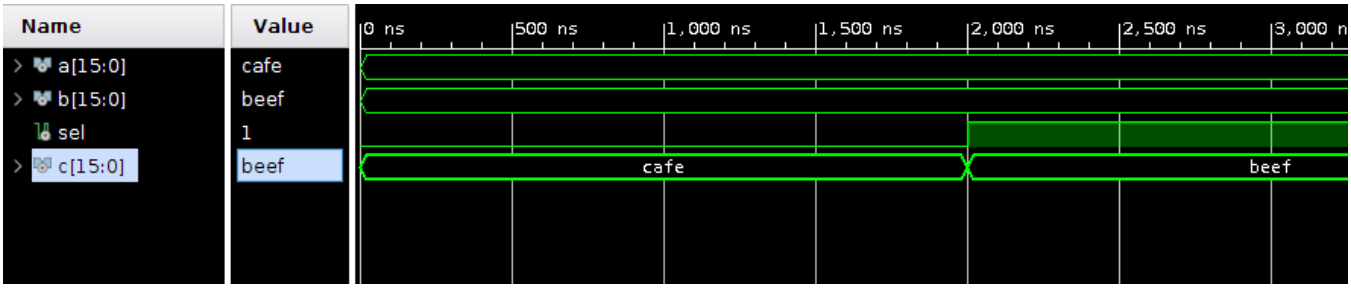


Figure 10: Two to One Multiplexer Testbench

4.5 Comparator Status Register

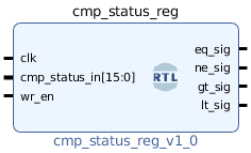


Figure 11: Comparator Status Register

Component Verification

The comparator status register stores the results of a comparison. The testbench for this component can be seen in Figure 12. The first clock cycle should not update the value as wr_en is low. The second clock cycle writes 0xFFFF to the register. The third clock cycle resets the register. The contents of the register are reflected by the signals eq_sig, ne_sig, gt_sig, and lt_sig.

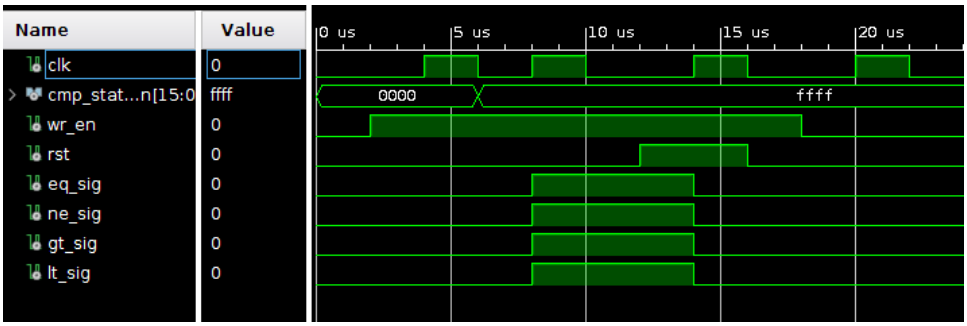


Figure 12: Comparator Status Register Testbench

4.6 11-bit Immediate Sign Extend

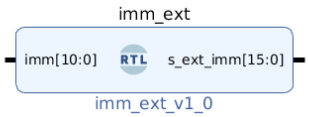


Figure 13: 11-bit Immediate Sign Extend

Component Verification

Used to sign extend 11-bit immediates from B-type instructions. Verification of this component can be seen in Figure 14.

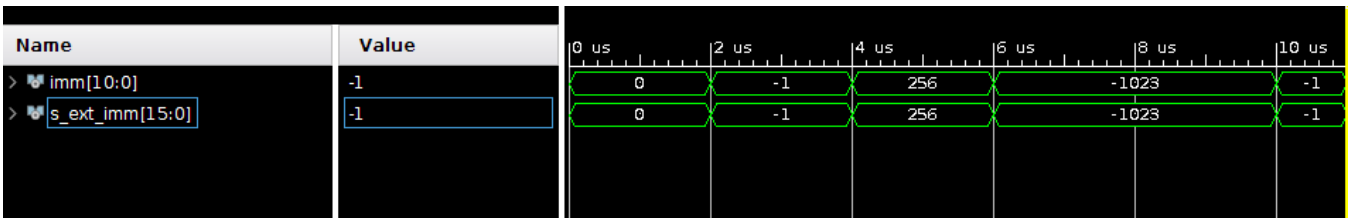


Figure 14: 11-bit Immediate Sign Extend Testbench

4.7 Program Counter Register

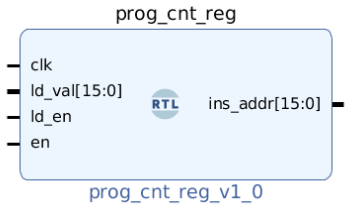


Figure 15: Program Counter Register

Component Verification

Program counter used to keep the address of the next instruction in memory. Verification of the program counter can be seen in Figure 16.

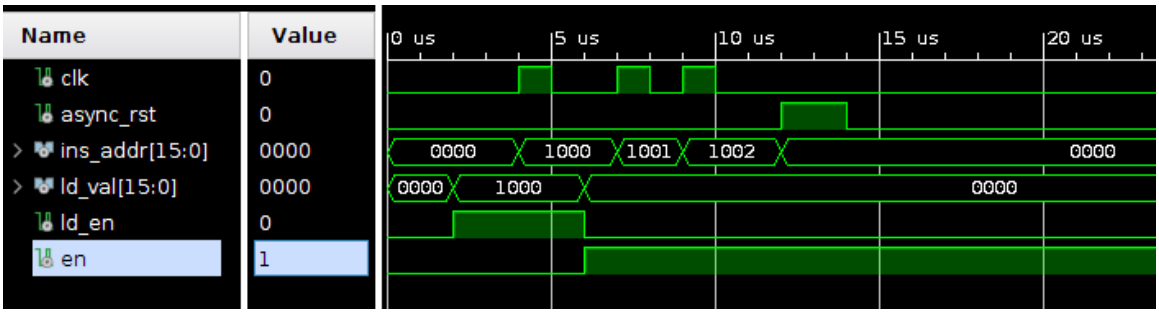


Figure 16: Program Counter Register Testbench

4.8 General Purpose Register File

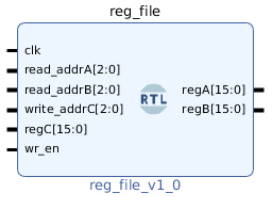


Figure 17: General Purpose Register File

Component Verification

GPR (General Purpose Register) file is used to store temporary values during computation. The verification of the register file can be seen in Figure 18. The testbench shows multiple writes to the GPR then clocks a synchronous reset.

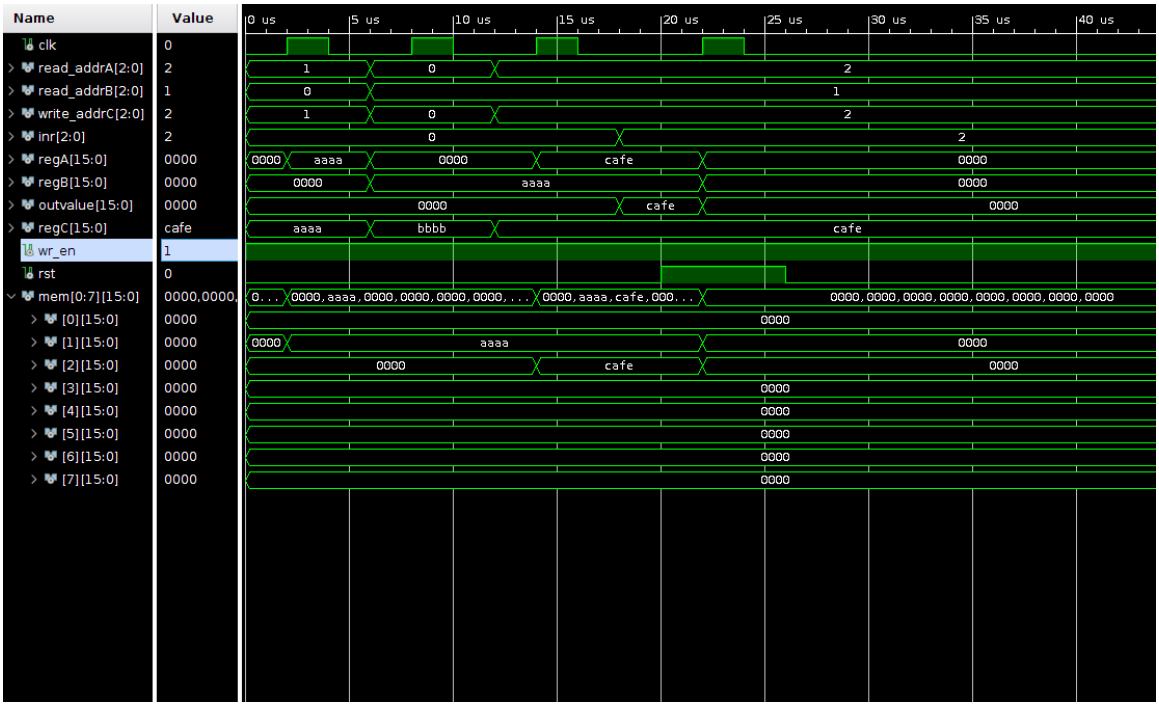


Figure 18: General Purpose Register File

4.9 Register File Input Multiplexer

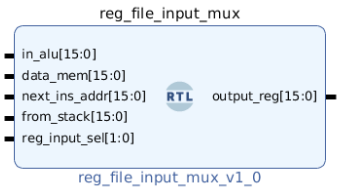


Figure 19: Register File Input Multiplexer

Component Verification

Used to multiplex the data input to the GPR file. The verification of this component can be seen in Figure 20.

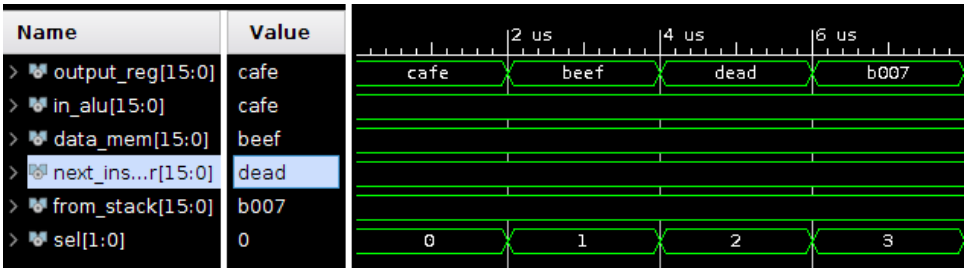


Figure 20: Register File Input Multiplexer

4.10 RI-Type Zero Extend Immediate & RI-Type Immediate Upper 6-bit Concatenation

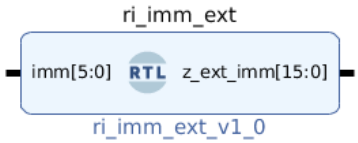


Figure 21: RI-Type Zero Extend Immediate

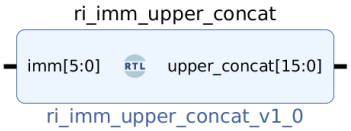


Figure 22: RI-Type Immediate Upper 6-bit Concatenation

Component Verification

The immediate zero extend module and immediate upper 6-bit concatenation module are verified below in Figure 23.

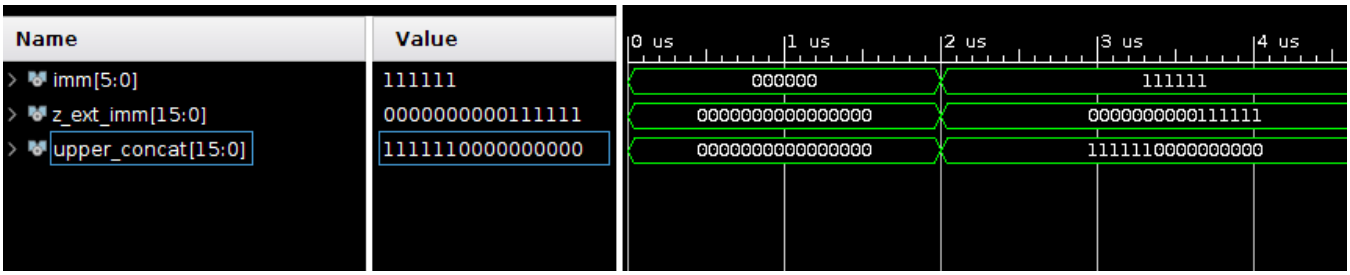


Figure 23: RI-Type Zero Extend Immediate

4.11 Stack Pointer Register

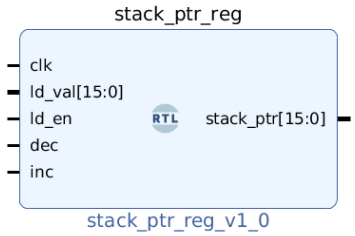


Figure 24: Stack Pointer Register

Component Verification

Stack pointer register maintains stack pointer. Verification of the stack pointer module can be seen in Figure 25.

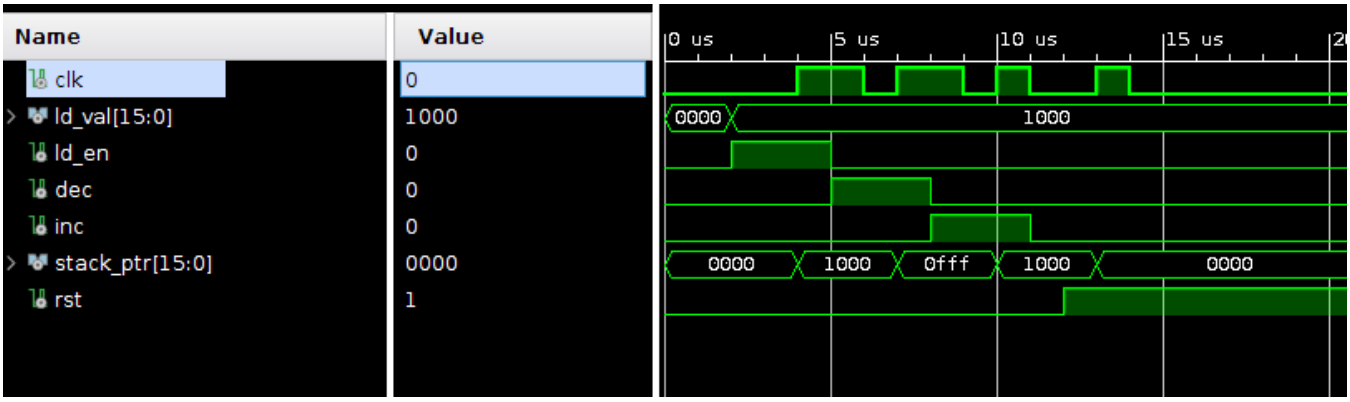


Figure 25: Stack Pointer Register