Risc Design

Sayan Samanta

Fall 2024

1 Introduction

The implemented two cycle 16-bit RISC processor has 128 word addressable memory and it has 16 unique instructions. The instructions can be categorized into two categories: 1. Single Word 2. Double Word and three formats: 1. Reg type instructions - Arithmetic or Logical 2. Jump or Flow control instructions - JMP, Conditional Jumps or CALL/RET 3. Mem type instructions - LW/SW. The ISA has three addressing modes 1. Register direct - Operand in the Registers 2. Immediate - Operand in the instruction 3. Register indirect - Register holds the address, where as the third addressing mode only used in Mem type instructions. The design does not have dedicated stack pointer or frame pointer.

General Purpose Resisters It has 32 general purpose registers (R0-R31) and out of them register-0 (R0) and register-1 (R1) are read only register containing a constant zero and one respectively. However, they can be used in as an operand for any instruction.

Special Purpose Registers Before we move to instructions, it is important to note some special registers. The **program counter** (PC) is not accessible to the user and always starts from value 0. It is the responsibility of the user to load the program from memory location - 0. Also, there is a **status register** that has four flags - zero, negative/less than, overflow and greater than equal; that get updated by every ALU operation. Again the status register is not accessible to the user for read or write but they are used for conditional branching operation. Finally, we have **Return Address (RA)** register is used when we make function call using CALL instruction. RA saves the return address which is $PC + 1$ and same is used when making a return to the main program using RET as PC loads the return address from RA register.

2 Instruction Description

Bit Fields and its sizes: The table 1 and table 2 contains the instruction along with its opcode and format. We need 4 bit opcode for 16 instructions and 5 bit addresses to access the 32 general purpose registers. Again we have 128 word 16-bit memory, so we need 7-bit addresses to access the 128 words with a linear addressing i.e. Program counter increments by 1 unless it is updated by flow control instructions.

HALT: The HALT instruction is a special type of instruction that is used to stop the PC from incrementing to the next instruction address and figure 1 shows the machine code format of it.

Instruction	Op-Code in Hex	Im Bit	Halt	Instruction Format	Operation
HALT	0	0	1	HALT	Stops the CPU
ADD	1	0/1	0	(ADD Rd, Rs)/(ADD Rd, #Value)	Rd = Rd + Rs/#Value
SUB	2	0/1	0	(SUB Rd, Rs)/(SUB Rd, #Value)	Rd = Rd - Rs/#Value
AND	3	0/1	0	(AND Rd, Rs)/(AND Rd, #Value)	Rd = Rd & Rs/#Value
OR	4	0/1	0	(OR Rd, Rs)/(OR Rd, #Value)	Rd = $Rd \| Rs$ / #Value
NOT	5	0	0	(NOT Rd, Rs)	Rd = (Rs)
CMP	6	0	0	(CMP Rd, Rs)/(CMP Rd, #Value)	Rd - Rs, Rd not updated
LW	7	0/1	0	(LW Rd, Rs)/(LW Rd, #Value)	Rd←M[Rs] / Rd = #value
SW	8	0	0	(SW Rd, Rs)	Rd→M[Rs]

Table 1: Halt, Register and Mem Instructions

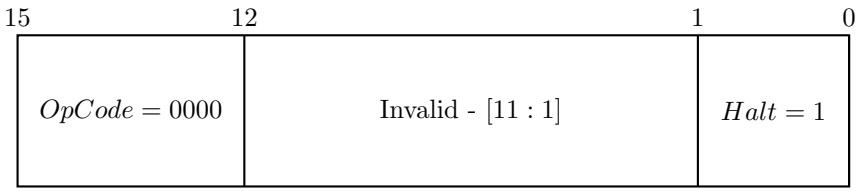

Figure 1: Halt instruction Format

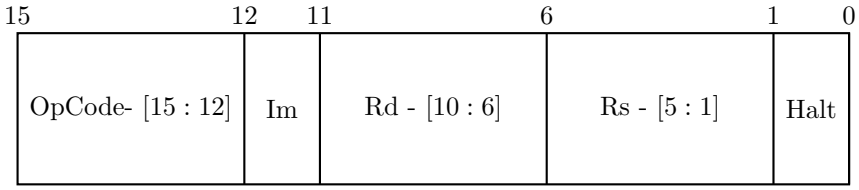

Figure 2: Reg Type Instruction Format

Reg Type: The register type instruction machine code format is shown in figure 2. The immediate addressing mode is enabled by setting $Im = 1$ and then Rs field is considered a 5 bit 2's complement signed integer. The immediate value is sign extended to 16 bit and used in ALU operation. The assembly instruction format for immediate instructions need to have a $'\#'$ added, like ADD R4,#5, in order to use that as an identifier for the compiler; where as ADD R4,R3 is an register addressed instructions.

Mem Type: The single word LW/SW instructions are the instructions that access the memory and uses register indirect addressing modes, i.e., registers are used as a pointer to memory and follows the same instruction format as in figure 2 but the immediate addressing mode should be disabled by setting $Im = 0$.

The only double word instruction is needed to load a 16-bit operand to a register and the format is shown in figure 3 and 4 and the immediate addressing mode should be enabled by setting $Im = 1$ to indicate that it is a load word immediate. The user must use $'\#'$, e.g. LW R3,#64, in order to load 64 into R3.

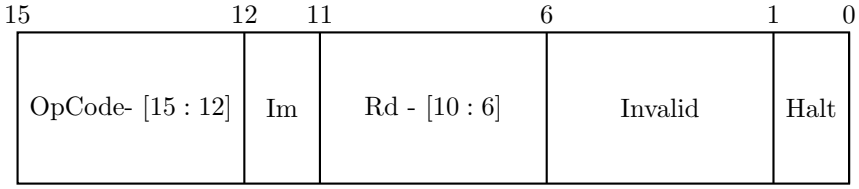

Figure 3: Word-1-Load Word Immediate

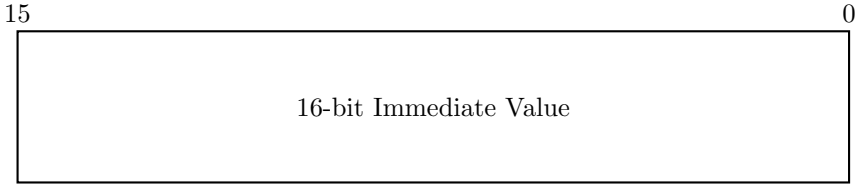

Figure 4: Word-2-Load Word Immediate

Flow Control Instructions: The machine code format of the flow control instructions are shown in 5 except the RET or return instruction which is presented in 6. As it can be noticed in 2, different types of jump instructions and call instruction embeds the 7-bit jump address within itself. As our address space is 128 words, 7 bits are sufficient to access any address which ranges from 0 $\rightarrow$127.

Function call is implemented using CALL instruction where the return address, i.e. the address of the next instruction, is saved in RA register automatically and **function return** is implemented by RET instruction where program counter is mapped to the return address stored in RA.

Greater than equal or less than constructs are implemented using a combination of CMP instruction which updates the status register and then JGE or JL instructions jump to the address based on whether the condition is true or false. Similarly, the **equal to or not equal to** constructs are implemented using a combination of CMP instruction which updates the status register and then JZ or JNZ instructions.

Instruction	Op-Code in Hex	Im Bit	Halt	Instruction Format	Operation
JMP	9	0	0	JMP #addr	Jump to #addr
JZ	A	0	0	JZ #addr	Jump to #addr if zero flag is set
JNZ	B	0	0	JNZ #addr	Jump to #addr if zero flag is reset
JGE	C	0	0	JGE #addr	Jump to #addr if greater than equal flag is set
JL	D	0	0	JL #addr	Jump to #addr if less than or negative flag is set
CALL	E	0	0	CALL #addr	Function call, saves $PC + 1$ in RA register
RET	F	0	0	RET	Return from function, PC Retrieves return address from RA

Table 2: Flow Control Instructions

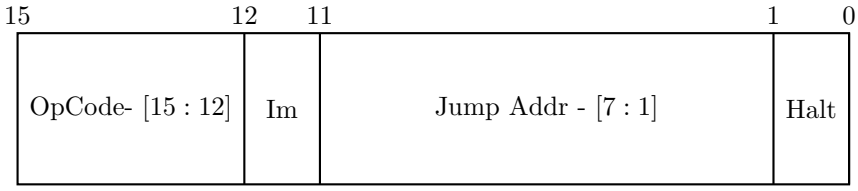

Figure 5: Jump Type instruction Format

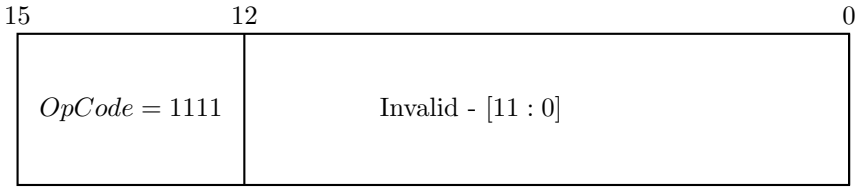

Figure 6: RET instruction Format

3 Programming Constructs

The programming constructs of different types can be implemented as following.

3.1 Construct-1

Basic operations: If we want to perform simple **arithmetic operations**, it can be performed in this way. Let say, a and b are stored at memory location 64 and 65 and c and d will be stored in 66 and 67. Desired operation:

$c = a + b$

$d = c - a$

Equivalent assembly implementation along with machine codes in Hex is given below:

LW R3,#64 →R3 = & a →78C0 0040

LW R4, R3 →R4 = a →7106

ADD R3,#1 →R3 = & b →18C2

LW R5, R3 →R5 = b →7146

ADD R5, R4 →R5 = $a + b$ →1148

ADD R3,#1 →R3 = & c →18C2

SW R5, R3 →C = $a + b$ →8146

LW R6, R3 →R6 = c →7186

SUB R6, R4 →R6 = $c - a$ →2188

ADD R3,#1 →R3 = & d →18C2

SW R6, R3 →d = $c - a$ →8186

HALT →Stop →0001

3.2 Construct-2

Simple **logical operations** can be performed also. Let us store a, b, c, d in the same memory location as before with an added variable e at location 68. Desired operation:

$c = a|b$

$d = c\&a$

$e = !a$

Equivalent assembly implementation is given below where R3 holds the address of a:

```
1  LW    R3, #64
2  LW    R4, R3
3  ADD   R3, #1
4  LW    R5, R3
5  OR    R5, R4
6  ADD   R3, #1
7  SW    R5, R3
8  LW    R6, R3
9  AND   R6, R4
10 ADD   R3, #1
11 SW    R6, R3
12 NOT   R2, R4
13 ADD   R3, #1
14 SW    R2, R3
15 HALT
```

3.3 Construct-3

More **complex and iterative tasks** can be performed in the following way. Let say we want to perform division by subtraction of two variable a and b with a function called **'div'** which takes the values as parameters and returns the result and the user need not worry about whether a is greater than b or otherwise as it will always divide the larger number by smaller one. In a high level language it will be implemented as below:

```
main{
a=3
b=12
c = div(a,b)
}
int div (a,b) {
 r=0;
 if (a=>b)
 {
  while (a!=0)
  {
    a=a-b;
    r++;
  }
 }
 else
 {
  while (b!=0)
  {
    b=b-a;
    r++;
  }
 }
 return r;
}
```

Equivalent assembly has been implemented in following way with consideration that a and b is present at memory location 64 and 65 and c is at 66.

```
LW R3,#64
LW R4, R3
ADD R3, #1
LW R5, R3
CALL DIV
ADD R3, #1
SW R6, R3
HALT

DIV:
AND R6, R0
CMP R5, R4
JGE L1
JL L2

L1:
CMP R5,R0
JZ DONE
JNZ SUB1
```

```
20
21 DONE:
22 RET
23
24 SUB1:
25 SUB R5, R4
26 ADD R6, #1
27 JMP L1
28
29 L2:
30 CMP R4, R0
31 JZ DONE
32 JNZ SUB2
33
34 SUB2:
35 SUB R4,R5
36 ADD R6,#1
37 JMP L2
```

4 Design and Testing

The user interface of the processor is shown in figure 8 and internal schematic is presented in figure 10. The port and its direction with purpose are listed in table 3. The 'access port' must be asserted by the user in order to access the memory while programming. The program must start from address '0' which is the default starting value of program counter. Once the data or program written into the memory, the user should assert the 'start port' to indicate processor to start execution. The processor executes the instructions when 'start' is de-asserted and continues until it gets the HALT instruction when it asserts the 'done' signal to indicate end of execution. The processor takes initial two clock cycles to setup its operations and then executes each instruction in two clock cycles, however the writeback happens at the posedge of the next clock cycle when data is needed. As the SRAM macro is a synchronuous one, i.e. read and write possible at only posedge of the clock; only load word with immediate value is a double word instruction and it takes five clock cycles to execute and load word with register addressing mode takes 3 cycles to execute. The SystemVerilog implementation is present in Listing 4 in the appendix section and the top module name is my_risc.

Testing: In order to test the ISA, a testbench with programming construct-1 has been developed, present in listing 5 in the appendix section and the output corresponding to that is presented in Listing 1. Desired operation:

$$c = a + b$$

$$d = c - a$$

It has $a = 5$ and $b = 7$, the output c and d has expected value of 12 and 7. The variables a to d are store in location 64 to 67 respectively. The register content after the execution can seen be in figure 7. The datapath and control block functions can be seen in figure 11 and 9 respectively.

Port Name	Direction	Width(in bits)	Purpose
Iclk	input	1	Active clock
Ireset	input	1	Active high reset
Iaccess	input	1	Active high. Asserted to access the memory
Iwrb	input	1	Active Low. Asserted to write into the memory and deasserted to read from the memory along with 'access pin'
Istart	input	1	Active high. Asserted to start the execution
Iaddr	input	7	Address lines
Idata_in	input	16	Input data
O_done	output	1	Active high. Asserted to indicate end of execution
Odata_out	output	16	Output data

Table 3: Port Description

Figure 7: Construct-1 Test Bench O/P Registers

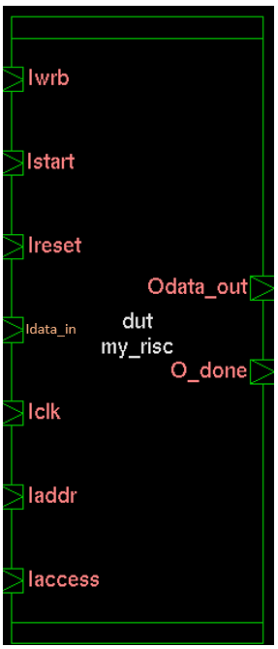

Figure 8: ISA User Interface

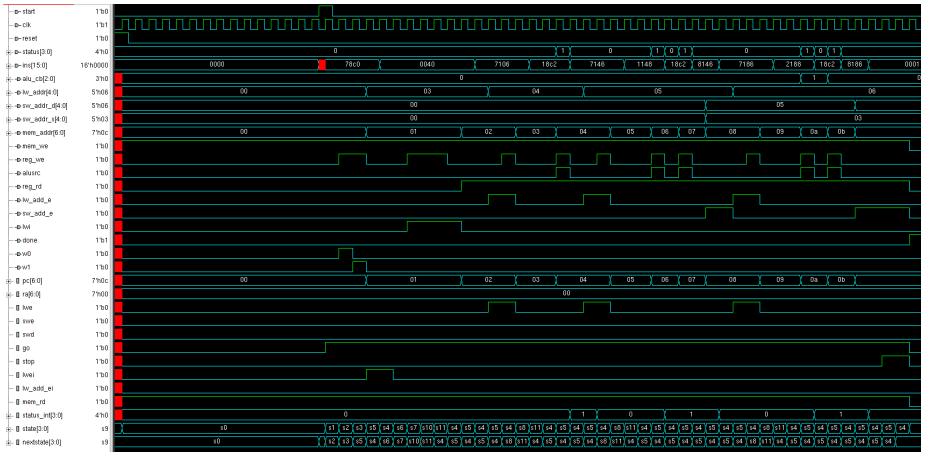

Figure 9: Construct-1 Test Bench O/P Control Block

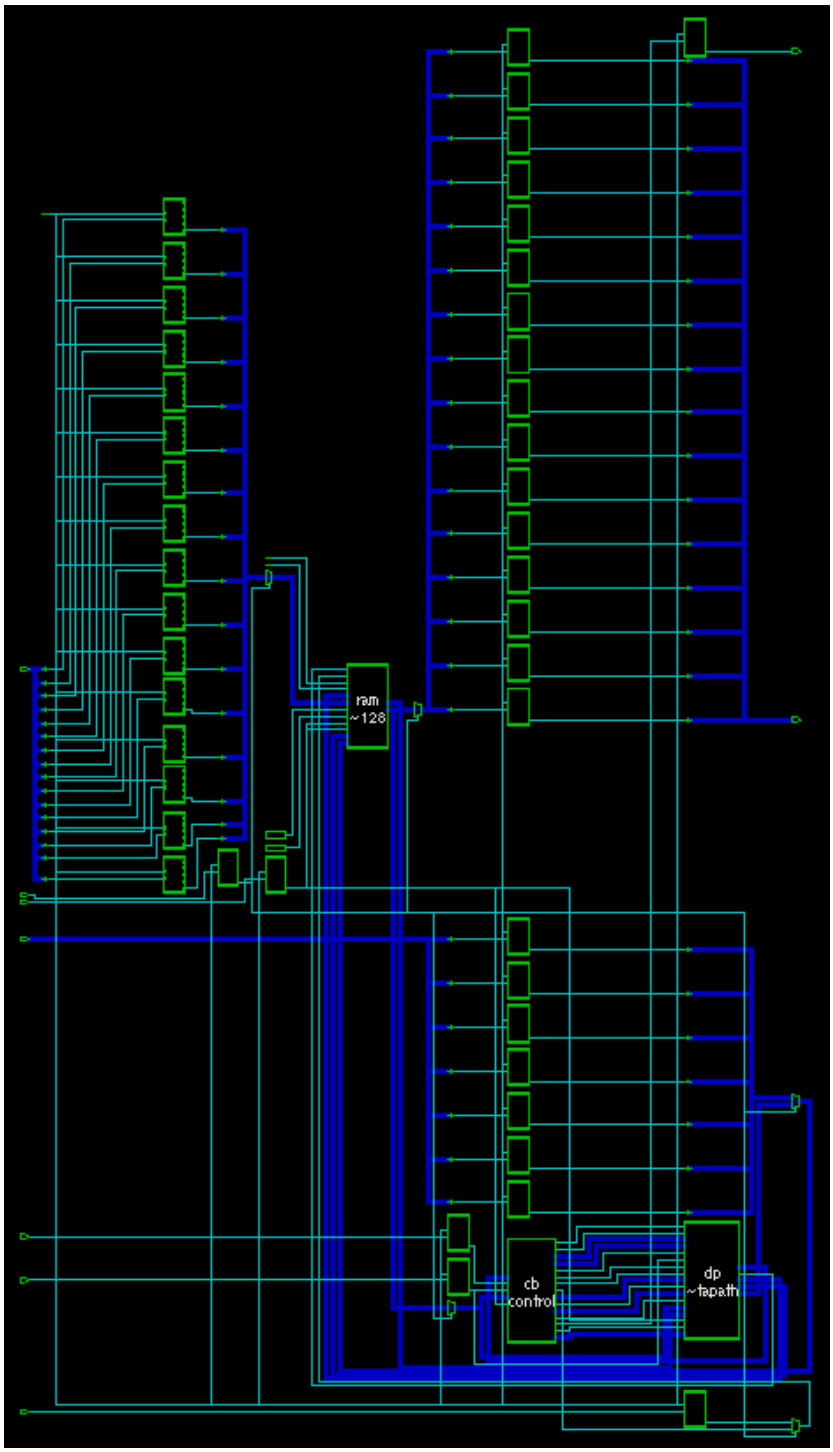

Figure 10: ISA Schematic Diagram

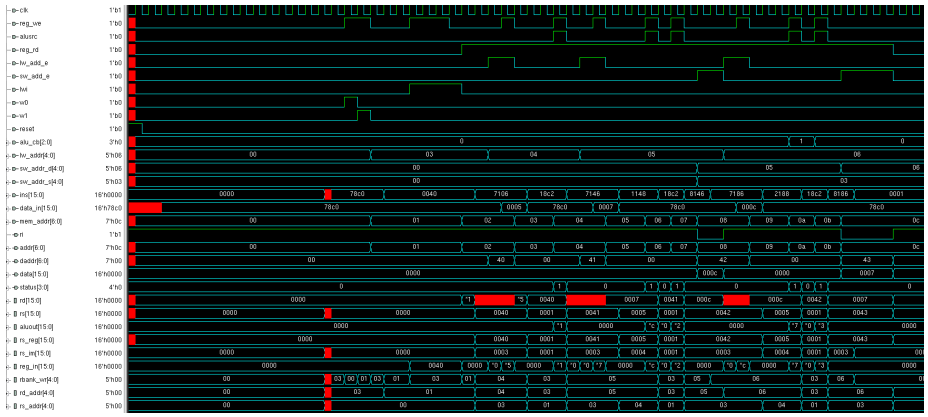

Figure 11: Construct-1 Test Bench O/P Data Path

```
Info: [VCS_SAVE_RESTORE_INFO] ASLR (Address Space Layout
    Randomization) is detected on the machine. To enable
    $save functionality, ASLR will be switched off and simv
    re-executed.
Please use '-no_save' simv switch to avoid re-execution or
    '-suppress=ASLR_DETECTED_INFO' to suppress this message.
Chronologic VCS simulator copyright 1991-2023
Contains Synopsys proprietary information.
Compiler version V-2023.12-SP2_Full64; Runtime version V
    -2023.12-SP2_Full64;  Dec  6 16:14 2024
time= 1360 ns, done=0
time= 1440 ns, done=0
time= 1520 ns, done=0
time= 1600 ns, done=0
time= 1680 ns, done=0
time= 1760 ns, done=0
time= 1840 ns, done=0
time= 1920 ns, done=0
time= 2000 ns, done=0
time= 2080 ns, done=0
time= 2160 ns, done=0
time= 2240 ns, done=0
time= 2320 ns, done=0
time= 2400 ns, done=0
time= 2480 ns, done=0
time= 2560 ns, done=0
time= 2640 ns, done=0
time= 2720 ns, done=0
time= 2800 ns, done=0
time= 2880 ns, done=0
time= 2960 ns, done=0
time= 3040 ns, done=0
```

```
time= 3120 ns, done=0
time= 3200 ns, done=0
time= 3280 ns, done=0
time= 3360 ns, done=0
time= 3440 ns, done=0
time= 3520 ns, done=0
time= 3600 ns, done=0
time= 3680 ns, done=0
time= 3760 ns, done=0
time= 3840 ns, done=0
time= 3920 ns, done=0
time= 4000 ns, done=0
time= 4080 ns, done=0
time= 4160 ns, done=0
time= 4240 ns, done=0
time= 4320 ns, done=0
time= 4400 ns, done=0
time= 4480 ns, done=0
time= 4560 ns, done=0
time= 4640 ns, done=0
time= 4720 ns, done=1
time= 4800 ns, done=1
time= 4880 ns, done=1
time= 4960 ns, done=1
time= 5040 ns, A=                5
time= 5120 ns, B=                7
Checking C=A+B
time= 5200 ns, C=               12
PASS
Checking D=C-A
time= 5280 ns, D=                7
PASS
$finish called from file "../../src_syn_no_always_comb/
    test_bench1_128word.sv", line 190.
$finish at simulation time             52800000
           V C S   S i m u l a t i o n   R e p o r t
Time: 5280000000 fs
CPU Time:      0.270 seconds;       Data structure size:
    0.1Mb
Fri Dec  6 16:14:08 2024
```

Listing 1: Construct-1 Output

Further testing has been done for construct-2 and the testbench for that is present in listing 6 in the appendix section and the output corresponding to that is presented in Listing 2. Here, the value of a is 00FF and b is FF00 in Hex. The variables a, b, c, d and e are store in consecutive memory location starting from 64. Desired operation:

$c = a|b$

$d = c\&a$

$e = !a$

The register content after operation is shown in figure 12 respectively.

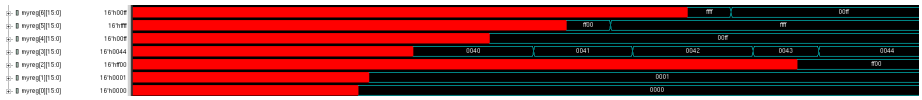

Figure 12: Construct-2 Test Bench O/P Registers

```
Info: [VCS_SAVE_RESTORE_INFO] ASLR (Address Space Layout
    Randomization) is detected on the machine. To enable
    $save functionality, ASLR will be switched off and simv
    re-executed.
Please use '-no_save' simv switch to avoid re-execution or
    '-suppress=ASLR_DETECTED_INFO' to suppress this message.
Chronologic VCS simulator copyright 1991-2023
Contains Synopsys proprietary information.
Compiler version V-2023.12-SP2_Full64; Runtime version V
    -2023.12-SP2_Full64;  Dec  6 16:14 2024
time= 1600 ns, O_done=0
time= 1680 ns, O_done=0
time= 1760 ns, O_done=0
time= 1840 ns, O_done=0
time= 1920 ns, O_done=0
time= 2000 ns, O_done=0
time= 2080 ns, O_done=0
time= 2160 ns, O_done=0
time= 2240 ns, O_done=0
time= 2320 ns, O_done=0
time= 2400 ns, O_done=0
time= 2480 ns, O_done=0
time= 2560 ns, O_done=0
time= 2640 ns, O_done=0
time= 2720 ns, O_done=0
time= 2800 ns, O_done=0
time= 2880 ns, O_done=0
time= 2960 ns, O_done=0
time= 3040 ns, O_done=0
time= 3120 ns, O_done=0
time= 3200 ns, O_done=0
time= 3280 ns, O_done=0
time= 3360 ns, O_done=0
time= 3440 ns, O_done=0
time= 3520 ns, O_done=0
```

```
time= 3600 ns, O_done=0
time= 3680 ns, O_done=0
time= 3760 ns, O_done=0
time= 3840 ns, O_done=0
time= 3920 ns, O_done=0
time= 4000 ns, O_done=0
time= 4080 ns, O_done=0
time= 4160 ns, O_done=0
time= 4240 ns, O_done=0
time= 4320 ns, O_done=0
time= 4400 ns, O_done=0
time= 4480 ns, O_done=0
time= 4560 ns, O_done=0
time= 4640 ns, O_done=0
time= 4720 ns, O_done=0
time= 4800 ns, O_done=0
time= 4880 ns, O_done=0
time= 4960 ns, O_done=0
time= 5040 ns, O_done=0
time= 5120 ns, O_done=0
time= 5200 ns, O_done=0
time= 5280 ns, O_done=0
time= 5360 ns, O_done=0
time= 5440 ns, O_done=1
time= 5520 ns, O_done=1
time= 5600 ns, O_done=1
time= 5680 ns, A=00ff
time= 5760 ns, B=ff00
Checking C=A|B
time= 5840 ns, C=ffff
PASS
Checking D=C&A
time= 5920 ns, D=00ff
PASS
Checking E=!A
time= 6000 ns, E=ff00
PASS
$finish called from file "../../src_syn_no_always_comb/
    test_bench2_128words.sv", line 219.
$finish at simulation time             60000000
           V C S   S i m u l a t i o n   R e p o r t
Time: 6000000000 fs
CPU Time:      0.290 seconds;       Data structure size:
    0.1Mb
Fri Dec  6 16:14:17 2024
```

Listing 2: Construct-2 Output

Finally testing has been done for construct-3 and the testbench for which is present in listing 7 in the appendix section and the output corresponding that testbench is present in Listing 3. Here, the value of a is 3 and b is 12 and are store in consecutive memory location starting from 64. Desired operation is division by subtraction:

```
main{
a=3
b=12
c = div(a,b)
}
int div (a,b) {
 r=0;
 if (a>b)
 {
  while (a!=0)
  {
    a=a-b;
    r++;
  }
 }
 else
 {
  while (b!=0)
  {
    b=b-a;
    r++;
  }
 }
 return r;
}
```

The register content after the execution is shown in figure 13. The value of c is stored at address 66. As this program has multiple flow control instruction, the change of PC value can be observed in figure 14.

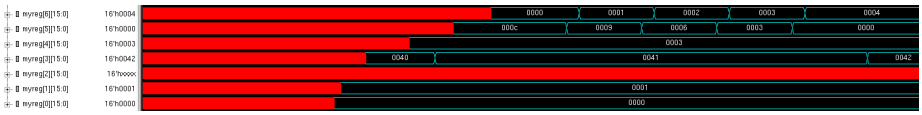

Figure 13: Construct-3 Test Bench O/P Registers

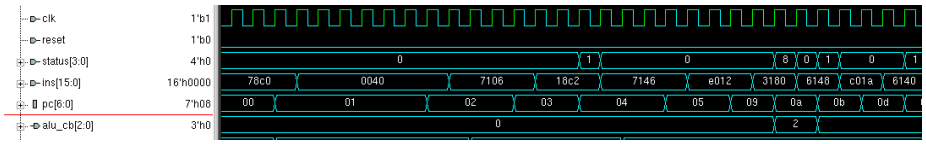

Figure 14: Construct-3 Test Bench O/P Control Block

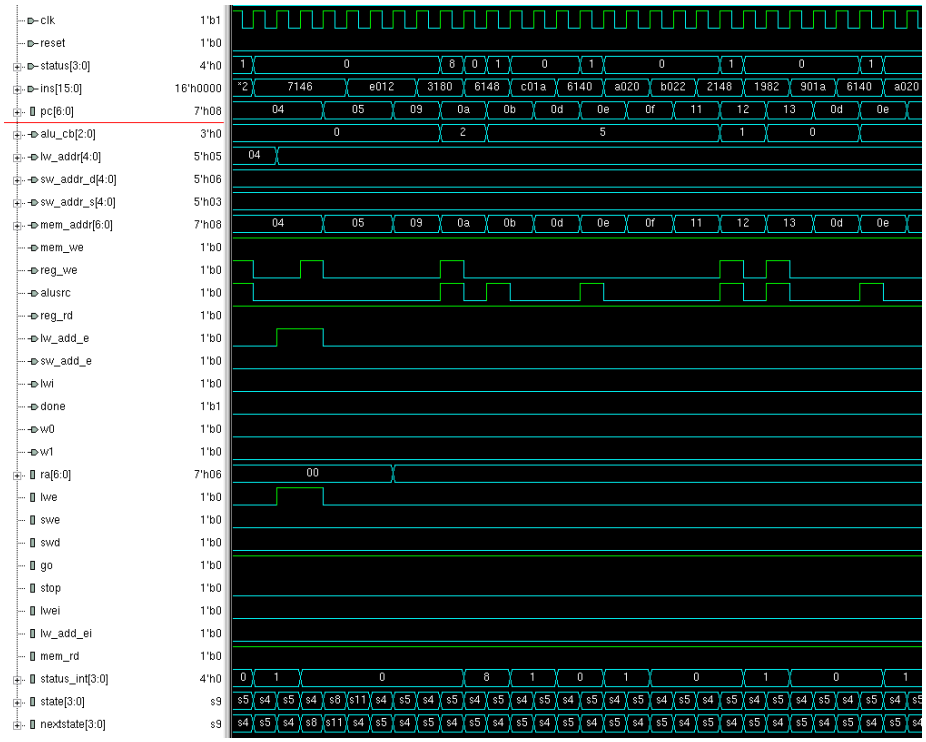

Figure 15: Construct-3 Test Bench O/P Control Block(all signals)

```
Info: [VCS_SAVE_RESTORE_INFO] ASLR (Address Space Layout
    Randomization) is detected on the machine. To enable
    $save functionality, ASLR will be switched off and simv
    re-executed.
Please use '-no_save' simv switch to avoid re-execution or
    '-suppress=ASLR_DETECTED_INFO' to suppress this message.
Chronologic VCS simulator copyright 1991-2023
Contains Synopsys proprietary information.
Compiler version V-2023.12-SP2_Full64; Runtime version V
    -2023.12-SP2_Full64;  Dec  6 16:14 2024
time= 2320 ns, O_done=0
time= 2400 ns, O_done=0
```

```
time= 2480 ns, O_done=0
time= 2560 ns, O_done=0
time= 2640 ns, O_done=0
time= 2720 ns, O_done=0
time= 2800 ns, O_done=0
time= 2880 ns, O_done=0
time= 2960 ns, O_done=0
time= 3040 ns, O_done=0
time= 3120 ns, O_done=0
time= 3200 ns, O_done=0
time= 3280 ns, O_done=0
time= 3360 ns, O_done=0
time= 3440 ns, O_done=0
time= 3520 ns, O_done=0
time= 3600 ns, O_done=0
time= 3680 ns, O_done=0
time= 3760 ns, O_done=0
time= 3840 ns, O_done=0
time= 3920 ns, O_done=0
time= 4000 ns, O_done=0
time= 4080 ns, O_done=0
time= 4160 ns, O_done=0
time= 4240 ns, O_done=0
time= 4320 ns, O_done=0
time= 4400 ns, O_done=0
time= 4480 ns, O_done=0
time= 4560 ns, O_done=0
time= 4640 ns, O_done=0
time= 4720 ns, O_done=0
time= 4800 ns, O_done=0
time= 4880 ns, O_done=0
time= 4960 ns, O_done=0
time= 5040 ns, O_done=0
time= 5120 ns, O_done=0
time= 5200 ns, O_done=0
time= 5280 ns, O_done=0
time= 5360 ns, O_done=0
time= 5440 ns, O_done=0
time= 5520 ns, O_done=0
time= 5600 ns, O_done=0
time= 5680 ns, O_done=0
time= 5760 ns, O_done=0
time= 5840 ns, O_done=0
time= 5920 ns, O_done=0
time= 6000 ns, O_done=0
time= 6080 ns, O_done=0
time= 6160 ns, O_done=0
time= 6240 ns, O_done=0
time= 6320 ns, O_done=0
time= 6400 ns, O_done=0
```

```
time= 6480 ns, O_done=0
time= 6560 ns, O_done=0
time= 6640 ns, O_done=0
time= 6720 ns, O_done=0
time= 6800 ns, O_done=0
time= 6880 ns, O_done=0
time= 6960 ns, O_done=0
time= 7040 ns, O_done=0
time= 7120 ns, O_done=0
time= 7200 ns, O_done=0
time= 7280 ns, O_done=0
time= 7360 ns, O_done=0
time= 7440 ns, O_done=0
time= 7520 ns, O_done=0
time= 7600 ns, O_done=0
time= 7680 ns, O_done=0
time= 7760 ns, O_done=0
time= 7840 ns, O_done=0
time= 7920 ns, O_done=0
time= 8000 ns, O_done=0
time= 8080 ns, O_done=0
time= 8160 ns, O_done=0
time= 8240 ns, O_done=0
time= 8320 ns, O_done=0
time= 8400 ns, O_done=0
time= 8480 ns, O_done=0
time= 8560 ns, O_done=0
time= 8640 ns, O_done=0
time= 8720 ns, O_done=0
time= 8800 ns, O_done=0
time= 8880 ns, O_done=0
time= 8960 ns, O_done=0
time= 9040 ns, O_done=0
time= 9120 ns, O_done=0
time= 9200 ns, O_done=0
time= 9280 ns, O_done=0
time= 9360 ns, O_done=0
time= 9440 ns, O_done=0
time= 9520 ns, O_done=0
time= 9600 ns, O_done=0
time= 9680 ns, O_done=1
time= 9760 ns, O_done=1
time= 9840 ns, O_done=1
time= 9920 ns, O_done=1
time=10000 ns, A=                3
time=10080 ns, B=               12
Checking C=B/A
time=10160 ns, C=                4
PASS
$finish called from file "../../src_syn_no_always_comb/
```

```
    test_bench3_128words.sv", line 318.
$finish at simulation time             101600000
           V C S   S i m u l a t i o n   R e p o r t
Time: 10160000000 fs
CPU Time:      0.270 seconds;       Data structure size:
    0.1Mb
Fri Dec  6 16:14:24 2024
```

Listing 3: Construct-3 Output

5 SRAM and IO pads

In the design, SRAM16x128 macro has been used as the memory which is a true dual port synchronous SRAM i.e. both read and write operations are done in posedge of the clk. This memory also has dual clocks for each of its read and write ports. Table 4 shows the interfacing ports on the macro.

Port Name	Direction	Width(in bits)	Purpose
CSB1/CSB2	input	1	Active Low Chip select for two ports
CE1/CE2	input	1	Independent clocks for two ports
OEB1/OEB2	input	1	Active Low. Deasserted to enable the output when reading
WEB1/WEB2	input	1	Active Low. Asserted to write into the memory and deasserted to read from the memory
A1/A2	input	7	Address lines
I1/I2	input	16	Input data
O1/O2	output	16	Output data

Table 4: SRAM Port Description

Input buffer and output buffers were also used as a macro from the library. The block diagram of the input buffer ISH1025, which is used in this design, is shown in Figure 16 and the truth table for the same is presented in Figure 17.

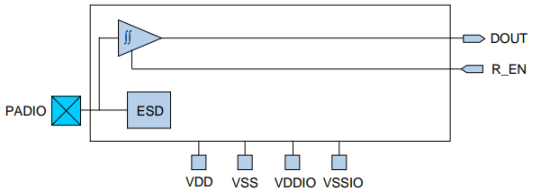

Figure 16: Input Buffer

Input		Output
PADIO	R_EN	DOUT
1	1	1
0	1	0
Z	0	0
0	0	0
1	0	0

Figure 17: Input Buffer Truth Table

D2I1025 has been used as output buffers and the block diagram and truth table for the same can be found in Figure 18 and 19.

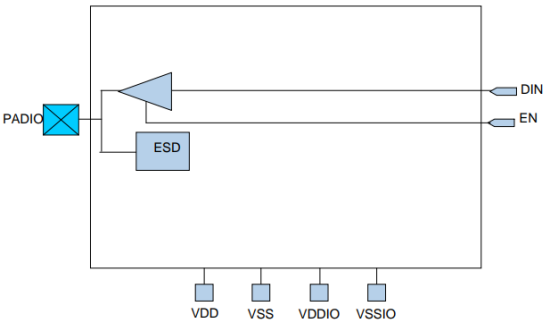

Figure 18: Output Buffer

Input		Output
EN	DIN	PADIO
1	1	1
1	0	0
0	X	z
X	X	z

Figure 19: Output Buffer Truth Table

6 Synthesis and Place-and-Route

As the cells cells were referred from multiple libraries the link and target libraries for the corresponding libraries had to be included in the dc command like following to link the design to corresponding macro cells:

```
1    set link_library [list /apps/designlib/SAED90_EDK/
     SAED_EDK90nm/Digital_Standard_cell_Library/synopsys/
     models/saed90nm_max.db /apps/designlib/SAED90_EDK/
     SAED_EDK90nm_IO/IO_Standard_Cell_Library/synopsys/models/
     saed90nm_io_max_fc.db /apps/designlib/SAED90_EDK/
     SAED_EDK90nm/Memories/synopsys/models/SRAM16x128_max.db /
     apps/designlib/SAED90_EDK/SAED_EDK90nm/
     Digital_Standard_cell_Library/synopsys/models/
     saed90nm_typ.db /apps/designlib/SAED90_EDK/
     SAED_EDK90nm_IO/IO_Standard_Cell_Library/synopsys/models/
     saed90nm_io_typ_fc.db /apps/designlib/SAED90_EDK/
     SAED_EDK90nm/Memories/synopsys/models/SRAM16x128_typ.db /
     apps/designlib/SAED90_EDK/SAED_EDK90nm/
     Digital_Standard_cell_Library/synopsys/models/
     saed90nm_min.db /apps/designlib/SAED90_EDK/
     SAED_EDK90nm_IO/IO_Standard_Cell_Library/synopsys/models/
     saed90nm_io_min_fc.db /apps/designlib/SAED90_EDK/
     SAED_EDK90nm/Memories/synopsys/models/SRAM16x128_min.db]
2    set target_library [list /apps/designlib/SAED90_EDK/
     SAED_EDK90nm/Digital_Standard_cell_Library/synopsys/
     models/saed90nm_max.db /apps/designlib/SAED90_EDK/
     SAED_EDK90nm_IO/IO_Standard_Cell_Library/synopsys/models/
     saed90nm_io_max_fc.db /apps/designlib/SAED90_EDK/
     SAED_EDK90nm/Memories/synopsys/models/SRAM16x128_max.db]
```

Few of the important reports from synthesis are the following.

```
1    Area
2   ---------------------------------
3   Combinational Area:   395332.678167
4   Noncombinational Area:
5                         240592.302227
6   Buf/Inv Area:           2814.566473
7   Total Buffer Area:           796.26
8   Total Inverter Area:        2018.30
9   Macro/Black Box Area:  43223.316406
10  Net Area:               4598.426272
11  ---------------------------------
12  Cell Area:            679148.296801
13  Design Area:          683746.723073
14
15  Cell Count
16  ---------------------------------
17  Hierarchical Cell Count:          8
18  Hierarchical Port Count:        418
```

```
  Leaf Cell Count:                2855
  Buf/Inv Cell Count:              509
  Buf Cell Count:                  144
  Inv Cell Count:                  365
  CT Buf/Inv Cell Count:             0
  Combinational Cell Count:       2289
  Sequential Cell Count:           566
  Macro Count:                       0
  ----------------------------------

  Timing Path Group 'ideal_clock1'
  ----------------------------------
  Levels of Logic:               15.00
  Critical Path Length:         134.72
  Critical Path Slack:           63.10
  Critical Path Clk Period:     200.00
  Total Negative Slack:           0.00
  No. of Violating Paths:         0.00
  Worst Hold Violation:           0.00
  Total Hold Violation:           0.00
  No. of Hold Violations:         0.00
  ----------------------------------

  Constraint                                              Cost
    -------------------------------------------------------
    min_capacitance                                       0.00 (
  MET)
    max_transition                                        0.00 (
  MET)
    max_capacitance                                    4780.13 (
  VIOLATED)
    max_delay/setup                                       0.00 (
  MET)
    sequential_clock_pulse_width                          0.00 (
  MET)
    critical_range                                        0.00 (
  MET)
```

Even though the target clock was given 200, worst path gave a slack of 63, which means it can run at clock speed of 135. But there were max cap violations in 17 nets.

Once synthesized netlist was verified, placement and routing was done in IC Compiler. Few commands which are bit different when macros are used directly from multiple library are listed below:

```
    set_app_var link_library "* /apps/designlib/SAED90_EDK/
  SAED_EDK90nm/Digital_Standard_cell_Library/synopsys/
  models/saed90nm_max.db /apps/designlib/SAED90_EDK/
  SAED_EDK90nm_IO/IO_Standard_Cell_Library/synopsys/models/
```

```
    saed90nm_io_max_fc.db /apps/designlib/SAED90_EDK/
    SAED_EDK90nm/Memories/synopsys/models/SRAM16x128_max.db /
    apps/designlib/SAED90_EDK/SAED_EDK90nm/
    Digital_Standard_cell_Library/synopsys/models/
    saed90nm_typ.db /apps/designlib/SAED90_EDK/
    SAED_EDK90nm_IO/IO_Standard_Cell_Library/synopsys/models/
    saed90nm_io_typ_fc.db /apps/designlib/SAED90_EDK/
    SAED_EDK90nm/Memories/synopsys/models/SRAM16x128_typ.db /
    apps/designlib/SAED90_EDK/SAED_EDK90nm/
    Digital_Standard_cell_Library/synopsys/models/
    saed90nm_min.db /apps/designlib/SAED90_EDK/
    SAED_EDK90nm_IO/IO_Standard_Cell_Library/synopsys/models/
    saed90nm_io_min_fc.db /apps/designlib/SAED90_EDK/
    SAED_EDK90nm/Memories/synopsys/models/SRAM16x128_min.db"
2    set_app_var target_library " /apps/designlib/SAED90_EDK/
    SAED_EDK90nm/Digital_Standard_cell_Library/synopsys/
    models/saed90nm_typ.db /apps/designlib/SAED90_EDK/
    SAED_EDK90nm_IO/IO_Standard_Cell_Library/synopsys/models/
    saed90nm_io_typ_fc.db /apps/designlib/SAED90_EDK/
    SAED_EDK90nm/Memories/synopsys/models/SRAM16x128_typ.db"
3   create_mw_lib -technology "/apps/designlib/SAED90_EDK/
    SAED_EDK90nm/Technology_Kit/milkyway/saed90nm_icc_1p9m.tf
    " -mw_reference_library "/apps/designlib/SAED90_EDK/
    SAED_EDK90nm/Digital_Standard_cell_Library/process/astro/
    fram/saed90nm /apps/designlib/SAED90_EDK/SAED_EDK90nm_IO/
    IO_Standard_Cell_Library/process/astro/saed_io_fc_fr /
    apps/designlib/SAED90_EDK/SAED_EDK90nm/Memories/process/
    astro/saed_sram_fr" "my_risc_LIB"
```

It is important to list all the libraries in the target and link library list and also when creating the design lib, we must provide the FRAM libraries in the reference for the macro cells to be linked. The IO pads were automatically placed by the ICC while SRAM macro needed some special commands to make the placement work (ram is the instance name of SRAM macro in the design):

```
1 set physopt_hard_keepout_distance 10
2 set placer_soft_keepout_chanel_width 25
3 set mw_logic0_net "VSS"
4 set mw_logic1_net "VDD"
5 set_undoable_attribute [get_cells -all ram] origin {1200
    1200}
6 set_dont_touch_placement [get_cells -all ram]
```

Also, as this is very big design with a very big area requirement, core_utilization had to be really low to reduce the DRCs and shorts.

```
1    create_floorplan -core_utilization 0.03 -start_first_row
     -left_io2core "50" -bottom_io2core "50" -right_io2core "
    50" -top_io2core "50"
```

Let us look at some important report of this.

```
Timing Path Group 'ideal_clock1'
  -----------------------------------
  Levels of Logic:              32.00
  Critical Path Length:         12.43
  Critical Path Slack:         187.48
  Critical Path Clk Period:    200.00
  Total Negative Slack:          0.00
  No. of Violating Paths:        0.00
  Worst Hold Violation:          0.00
  Total Hold Violation:          0.00
  No. of Hold Violations:        0.00
  -----------------------------------


  Cell Count
  -----------------------------------
  Hierarchical Cell Count:          8
  Hierarchical Port Count:        458
  Leaf Cell Count:               2643
  Buf/Inv Cell Count:             300
  Buf Cell Count:                 140
  Inv Cell Count:                 160
  CT Buf/Inv Cell Count:           64
  Combinational Cell Count:      2077
  Sequential Cell Count:          566
  Macro Count:                      1
  -----------------------------------


  Area
  -----------------------------------
  Combinational Area:   396747.334123
  Noncombinational Area:
                        240592.302227
  Buf/Inv Area:           3910.348842
  Total Buffer Area:          2946.36
  Total Inverter Area:         963.99
  Macro/Black Box Area:  43223.316406
  Net Area:               7828.432591
  Net XLength        :      436080.62
  Net YLength        :      416116.12
  -----------------------------------
  Cell Area:            680562.952756
  Design Area:          688391.385348
  Net Length        :       852196.75


  Design Rules
  -----------------------------------
```

```
  Total Number of Nets:           2755
  Nets With Violations:             73
  Max Trans Violations:             21
  Max Cap Violations:               72
```

The verilog generated by this step also passes the verification with same test benches and the final view is shown in Figure 20.

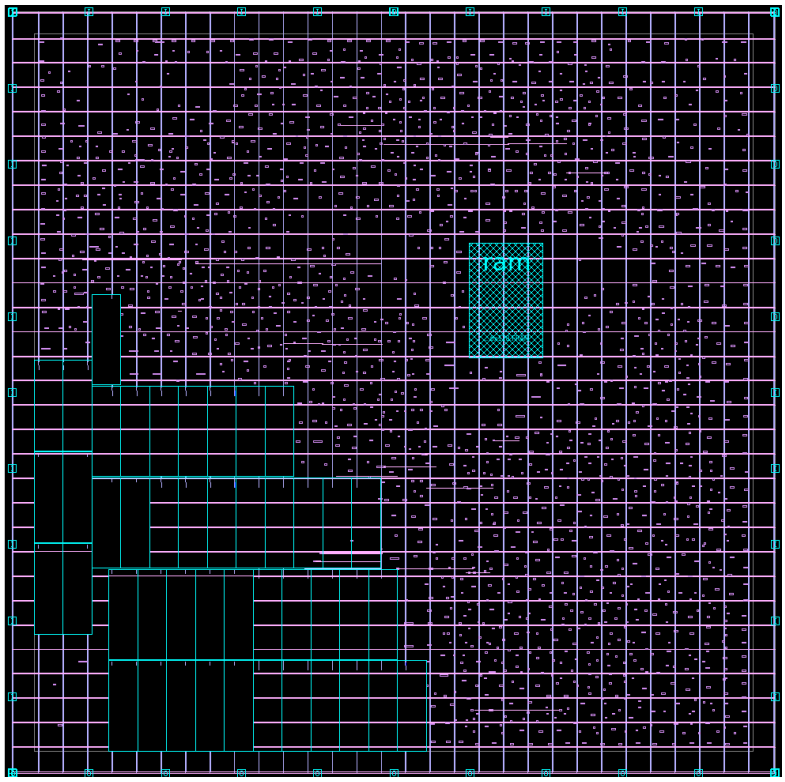

Figure 20: ICC Output

Powe analysis was done in power-time and the power consumption for Construct-3 or testbench in Listing 7 is following:

```
Report : Averaged Power


Power-specific unit information :
    Voltage Units = 1 V
    Capacitance Units = 0.001 pf
    Time Units = 1 ns
    Dynamic Power Units = 1 W
    Leakage Power Units = 1 W
```

```
10
11                                        Int      Switch   Leak
        Total
12 Hierarchy                              Power    Power
    Power    Power    %
13 ---------------------------------------------------------------

14 my_risc                                   0.409 3.76e-05 1.97
    e-03    0.411 100.0
15   cb (control)                         3.45e-06 1.90e-06 1.09
     e-05 1.62e-05   0.0
16     r142 (control_DW01_inc_0)          7.42e-08 6.72e-08 4.60
     e-07 6.01e-07   0.0
17   dp (datapath)                        1.46e-05 2.56e-05 1.12
     e-04 1.53e-04   0.0
18     my_reg (regbank)                   1.39e-05 2.40e-05 1.02
     e-04 1.39e-04   0.0
19     sign (signExt)                        0.000    0.000
     0.000    0.000   N/A
20     alu (myAlu)                        6.08e-07 9.37e-07 7.86
     e-06 9.41e-06   0.0
21       add_46 (myAlu_DW01_add_0)        9.01e-08 3.63e-08 1.43
     e-06 1.55e-06   0.0
22       r78 (myAlu_DW01_sub_0)           2.67e-07 2.46e-07 1.74
     e-06 2.26e-06   0.0
23
24 Report : Time Based Power
25
26
27 Power-specific unit information :
28     Voltage Units = 1 V
29     Capacitance Units = 0.001 pf
30     Time Units = 1 ns
31     Dynamic Power Units = 1 W
32     Leakage Power Units = 1 W
33
34
35
36                                        Int      Switch   Leak
        Total
37 Hierarchy                              Power    Power
    Power    Power    %
38 ---------------------------------------------------------------

39 my_risc                                   0.410 1.07e-05 1.97
    e-03    0.412 100.0
40   cb (control)                         4.37e-07 3.95e-07 9.79
     e-06 1.06e-05   0.0
41     r142 (control_DW01_inc_0)          2.57e-08    0.000 4.61
     e-07 4.86e-07   0.0
```

```
  dp (datapath)                          1.87e-08 1.00e-07 1.06
    e-04 1.07e-04   0.0
    my_reg (regbank)                     1.07e-09    0.000 9.74
    e-05 9.74e-05   0.0
    sign (signExt)                          0.000    0.000
    0.000    0.000   N/A
    alu (myAlu)                          1.47e-08 9.76e-08 6.23
    e-06 6.34e-06   0.0
      add_46 (myAlu_DW01_add_0)             0.000    0.000 1.42
    e-06 1.42e-06   0.0
      r78 (myAlu_DW01_sub_0)                0.000    0.000 1.43
    e-06 1.43e-06   0.0


                                         Peak     Peak
      Glitch   X-tran
Hierarchy                                Power    Time
      Power    Power
-------------------------------------------------------------------

my_risc                                  1.40e+04
    1760.0000-1760.0001

          0.000 7.13e-08
  cb (control)                           4.55e-04
    7080.5740-7080.5741

          0.000 6.13e-10
    r142 (control_DW01_inc_0)            2.40e-05
    8840.6570-8840.6571

          0.000    0.000
  dp (datapath)                          1.80e-04
    4441.2853-4441.2854

          0.000 7.92e-09
    my_reg (regbank)                     1.43e-04
    9640.7220-9640.7221

          0.000    0.000
    sign (signExt)                          0.000
    0.0000-0.0001     0.000    0.000
    alu (myAlu)                          7.81e-05
    8200.9770-8200.9771

          0.000 7.79e-09
      add_46 (myAlu_DW01_add_0)          1.42e-06
    0.0000-0.0001     0.000    0.000
      r78 (myAlu_DW01_sub_0)             1.43e-06
    0.0000-0.0001     0.000    0.000
```

While I though that SRAM will consume the highest power, but it did not.

7 Conclusion

Even though the initial implementation was done in single cycle, I had to change the design to a two-cycle implementation to use the SRAM macro. It also increased the machine cycle for load word word instruction. The max cap violation during synthesis and place and route was caused by the IO pads which are really large. Also, even with increasing the area, ICC could not fix the shorts and DRCs. But the overall design passes the verification at each stage and every instruction simulated has worked properly.

8 Appendix - SystemVerilog Codes

```
module regbank (
input logic clk, regwe, regrd,
input logic [4:0] rd_addr, rs_addr, wr_addr,
input logic [15:0] write_data,
output logic [15:0] rd, rs
);
    logic [15:0] myreg[31:0];
    always_ff @(posedge clk)
    begin
        if (regwe)
        begin
            if (wr_addr==0) myreg[wr_addr] <= 0;
            else if (wr_addr==1) myreg[wr_addr] <= 16'h0001;
            else myreg[wr_addr] <= write_data;
        end
    end
    assign rs = (regrd)? myreg[rs_addr]:0;
    assign rd = (regrd)? myreg[rd_addr]:0;
endmodule

module signExt (input logic [4:0] data_in,
        output logic [15:0] data_out);
    assign data_out = {{12{data_in[4]}},data_in[3:0]};
endmodule


module myAlu (input logic [15:0] A, B,
        input logic alusrc, reset,
                input logic [2:0] alu_cb,
                output logic [15:0] ALUOUT,
        output logic [3:0] status);
```

```
32      logic [15:0] comp;
33          always_comb
34          begin
35          if (reset)
36          begin
37              status = 4'd0;
38              ALUOUT = 16'd0;
39              comp = 16'd0;
40          end
41          else if (alusrc)
42          begin
43              case (alu_cb)
44                          3'b000: begin
45                  comp = 16'd0;
46                  ALUOUT = $signed(A) + $signed(B);
47                  if (ALUOUT == 16'd0)
48                      status[3] = 1'b1;
49                  else
50                      status[3] = 1'b0;
51                  if (ALUOUT[15] == 1'b1)
52                      status[2] = 1'b1;
53                  else
54                      status[2] = 1'b0;
55                  if ((A[15]==1'b0) && (B[15]==1'b0) && (
      ALUOUT[15] == 1'b1))
56                      status[1] = 1'b1;
57                  else
58                      status[1] = 1'b0;
59                  if ((A[15]==1'b1) && (B[15]==1'b1) && (
      ALUOUT[15] == 1'b0))
60                      status[1] = 1'b1;
61                  else
62                      status[1] = 1'b0;
63                  if (A>=B)
64                      status[0] = 1'b1;
65                  else
66                      status[0] = 1'b0;
67                  end
68                          3'b001: begin
69                  comp = 16'd0;
70                  ALUOUT = $signed(A) - $signed(B);
71                  if (A == B)
72                      status[3] = 1'b1;
73                  else
74                      status[3] = 1'b0;
75                  if (ALUOUT[15] == 1'b1)
76                      status[2] = 1'b1;
77                  else
78                      status[2] = 1'b0;
```

```
79                  if ((A[15]==1'b0) && (B[15]==1'b1) && (
    ALUOUT[15] == 1'b1))
80                      status[1] = 1'b1;
81                  else
82                      status[1] = 1'b0;
83                  if ((A[15]==1'b1) && (B[15]==1'b0) && (
    ALUOUT[15] == 1'b0))
84                      status[1] = 1'b1;
85                  else
86                      status[1] = 1'b0;
87                  if (A>=B)
88                      status[0] = 1'b1;
89                  else
90                      status[0] = 1'b0;
91                  end
92              3'b010: begin
93                  comp = 16'd0;
94                  ALUOUT = A&B;
95                  status[1] = 1'b0;
96                  if (ALUOUT == 16'd0)
97                      status[3] = 1'b1;
98                  else
99                      status[3] = 1'b0;
100                 if (ALUOUT[15] == 1'b1)
101                     status[2] = 1'b1;
102                 else
103                     status[2] = 1'b0;
104                 if (A>=B)
105                     status[0] = 1'b1;
106                 else
107                     status[0] = 1'b0;
108                 end
109             3'b011: begin
110                 comp = 16'd0;
111                 ALUOUT = A|B;
112                 status[1] = 1'b0;
113                 if (ALUOUT == 16'd0)
114                     status[3] = 1'b1;
115                 else
116                     status[3] = 1'b0;
117                 if (ALUOUT[15] == 1'b1)
118                     status[2] = 1'b1;
119                 else
120                     status[2] = 1'b0;
121                 if (A>=B)
122                     status[0] = 1'b1;
123                 else
124                     status[0] = 1'b0;
125                 end
126             3'b100: begin
```

```
127                 comp = 16'd0;
128                 ALUOUT = ~B;
129                 status[1] = 1'b0;
130                 status[0] = 1'b0;
131                 if (ALUOUT == 16'd0)
132                     status[3] = 1'b1;
133                 else
134                     status[3] = 1'b0;
135                 if (ALUOUT[15] == 1'b1)
136                     status[2] = 1'b1;
137                 else
138                     status[2] = 1'b0;
139                 end
140             3'b101: begin
141                 ALUOUT = 16'd0;
142                 comp = $signed(A) - $signed(B);
143                 if (A == B)
144                     status[3] = 1'b1;
145                 else
146                     status[3] = 1'b0;
147                 if (comp[15] == 1'b1)
148                     status[2] = 1'b1;
149                 else
150                     status[2] = 1'b0;
151                 if ((A[15]==1'b0) && (B[15]==1'b1) && (comp
    [15] == 1'b1))
152                     status[1] = 1'b1;
153                 else
154                     status[1] = 1'b0;
155                 if ((A[15]==1'b1) && (B[15]==1'b0) && (comp
    [15] == 1'b0))
156                     status[1] = 1'b1;
157                 else
158                     status[1] = 1'b0;
159                 if (A>=B)
160                     status[0] = 1'b1;
161                 else
162                     status[0] = 1'b0;
163                 end
164             default: begin
165                 ALUOUT = 16'd0;
166                 comp = 16'd0;
167                 status = 4'd0;
168                 end
169             endcase
170         end
171         else
172         begin
173             ALUOUT = 16'd0;
174             comp = 16'd0;
```

```
175             status = 4'd0;
176                 end
177     end
178 endmodule
179
180 module datapath(
181 input logic clk, reg_we,alusrc,reg_rd,lw_add_e,sw_add_e, lwi
        , w0, w1, reset,
182 input logic [2:0] alu_cb,
183 input logic [4:0] lw_addr, sw_addr_d, sw_addr_s,
184 input wire [15:0] ins,
185 input  wire [15:0] data_in,
186 input logic [6:0] mem_addr,
187 output logic ri,
188 output logic [6:0] addr,
189 output logic [6:0] daddr,
190 output logic [15:0] data,
191 output logic [3:0] status
192 );
193 logic [15:0] rd, rs, aluout, rs_reg, rs_im, reg_in;
194 //wire  [15:0] ins;
195 //wire  [15:0] data_in;
196 logic [4:0] rbank_wr,rd_addr,rs_addr;
197 regbank my_reg(.clk(clk), .regwe(reg_we), .regrd(reg_rd), .
        rd_addr(rd_addr), .rs_addr(rs_addr), .wr_addr(rbank_wr),
        .write_data(reg_in), .rd(rd), .rs(rs_reg) );
198 myAlu alu(.A(rd), .B(rs), .alusrc(alusrc), .reset(reset), .
        alu_cb(alu_cb), .ALUOUT(aluout), .status(status[3:0]));
199 signExt sign(.data_in(ins[5:1]), .data_out(rs_im));
200 assign rs = (ins[11])? rs_im : rs_reg;
201 assign addr = mem_addr;
202 //assign ri = lw_add_e;
203 assign daddr = (lw_add_e||sw_add_e) ? (rs_reg[6:0]):7'b0;
204
205 always_comb
206 begin
207     if (lwi)
208     begin
209         reg_in = ins[15:0];
210     end
211     else if (lw_add_e)
212     begin
213         reg_in = data_in[15:0];
214     end
215     else
216     begin
217         reg_in = aluout;
218     end
219     if (w0) rbank_wr = 0;
220     else if (w1) rbank_wr = 5'b00001;
```

```
221     else if (lwi) rbank_wr = lw_addr;
222     else rbank_wr = ins[10:6];
223     if (sw_add_e)
224     begin
225         rd_addr = sw_addr_d;
226         rs_addr = sw_addr_s;
227         data = rd[15:0];
228         ri = 1'b0;
229     end
230     else
231     begin
232         rd_addr = ins[10:6];
233         rs_addr = ins[5:1];
234         data = 16'b0;
235         ri = 1'b1;
236     end
237 end
238 endmodule
239
240 module control(
241 input logic start, clk, reset,
242 input logic [3:0] status,
243 input wire [15:0] ins,
244 output logic [2:0] alu_cb,
245 output logic [4:0] lw_addr, sw_addr_d, sw_addr_s,
246 output logic [6:0] mem_addr,
247 output logic mem_we,reg_we,alusrc,reg_rd,lw_add_e,sw_add_e,
    lwi,done,w0,w1);
248     wire [15:0] ins;
249     logic [6:0] pc,ra;
250     logic lwe,swe,swd,go,stop, lwei,lw_add_ei,mem_rd;
251     logic [3:0] status_int;
252
253
254     typedef enum logic [3:0] {s0, s1, s2, s3, s4, s5, s6, s7
    ,s8,s9,s10,s11} statetype;
255         statetype state, nextstate;
256     always_ff @(posedge clk)
257     begin
258         if (reset)  state <= s0;
259         else if (start) state <= s1;
260         else if (go) state <= nextstate;
261
262     end
263
264     always_comb
265         begin
266                 case (state)
267                         s0: begin
268                 if(start) nextstate = s1;
```

```
269                 else nextstate = s0;
270                 end
271                         s1: nextstate = s2;
272             s2: nextstate = s3;
273             s3: nextstate = s5;
274                         s4: begin
275                 if(lwei) nextstate = s6;
276                 else if(lwe) nextstate = s8;
277                 else if (stop) nextstate = s9;
278                 else nextstate = s5;
279                 end
280             s5: nextstate = s4;
281             s6: nextstate = s7;
282             s7: nextstate = s10;
283             s8: nextstate = s11;
284                         s9: begin
285                 if (start) nextstate = s1;
286                 else nextstate = s9;
287                 end
288             s10: nextstate = s11;
289             s11: nextstate = s4;
290             default: nextstate = s0;
291                 endcase
292         end
293     always_ff @(posedge clk)
294     begin
295     if (nextstate == s0)
296     begin
297         go <= 1'b0;
298         done <= 1'b0;
299         stop <= 1'b0;
300         alusrc <= 1'b0;
301         reg_we <= 1'b0;
302         lwe <= 1'b0;
303         swe <= 1'b0;
304         mem_rd <= 1'b1;
305         reg_rd <= 1'b0;
306         w0 <= 1'b0;
307         w1 <= 1'b0;
308         lw_add_e <= 1'b0;
309         sw_add_e <= 1'b0;
310         lwi <= 1'b0;
311         lwei <= 1'b0;
312         swd <= 1'b0;
313         alu_cb <= 3'b0;
314         lw_addr <= 4'b0;
315         sw_addr_d <= 4'b0;
316         sw_addr_s <= 4'b0;
317         lw_add_ei <= 1'b0;
318         pc <= 7'd0;
```

```
319             ra <= 7'd0;
320             status_int <= 4'b0;
321         end
322         else if (nextstate == s1)
323         begin
324             go <= 1'b1;
325             done <= 1'b0;
326 
327         end
328         else if (nextstate == s2)
329         begin
330             w0 <= 1'b1;
331             reg_we <= 1'b1;
332         end
333         else if (nextstate == s3)
334         begin
335             w0 <= 1'b0;
336             w1 <= 1'b1;
337             mem_rd <= 1'b1;
338             reg_we <= 1'b1;
339         end
340         else if (nextstate == s4)
341         begin
342             reg_we <= 1'b0;
343             mem_rd <= 1'b1;
344             alusrc <= 1'b0;
345             status_int <= status;
346             if (lwe)
347                 begin
348                 reg_we <= 1'b1;
349                 end
350         end
351         else if (nextstate == s5)
352         begin
353         w1 <= 1'b0;
354         case (ins[15:12])
355         4'd0: begin
356             alusrc <= 1'b0;
357             reg_we <= 1'b0;
358             lwe <= 1'b0;
359             swe <= 1'b0;
360             mem_rd <= 1'b1;
361             reg_rd <= 1'b1;
362             lwi <= 1'b0;
363             if (ins[0])
364             begin
365                 stop <= 1'b1;
366                 pc <= pc;
367             end
368             else
```

```
369             pc <= pc + 1;
370          end
371     4'd1: begin
372          alu_cb <= 3'b0;
373          alusrc <= 1'b1;
374          reg_we <= 1'b1;
375          lwi <= 1'b0;
376          lwe <= 1'b0;
377          swe <= 1'b0;
378          lw_add_e <= 1'b0;
379          sw_add_e <= 1'b0;
380          pc <= pc + 1;
381          end
382     4'd2: begin
383          alu_cb <= 3'b001;
384          alusrc <= 1'b1;
385          reg_we <= 1'b1;
386          lwi <= 1'b0;
387          lwe <= 1'b0;
388          swe <= 1'b0;
389          lw_add_e <= 1'b0;
390          sw_add_e <= 1'b0;
391          pc <= pc + 1;
392          end
393     4'd3: begin
394          alu_cb <= 3'b010;
395          alusrc <= 1'b1;
396          reg_we <= 1'b1;
397          lwi <= 1'b0;
398          lwe <= 1'b0;
399          swe <= 1'b0;
400          lw_add_e <= 1'b0;
401          sw_add_e <= 1'b0;
402          pc <= pc + 1;
403          end
404     4'd4: begin
405          alu_cb <= 3'b011;
406          alusrc <= 1'b1;
407          reg_we <= 1'b1;
408          lwi <= 1'b0;
409          lwe <= 1'b0;
410          swe <= 1'b0;
411          lw_add_e <= 1'b0;
412          sw_add_e <= 1'b0;
413          pc <= pc + 1;
414          end
415     4'd5: begin
416          alu_cb <= 3'b100;
417          alusrc <= 1'b1;
418          reg_we <= 1'b1;
```

```
419         lwi <= 1'b0;
420         lwe <= 1'b0;
421         swe <= 1'b0;
422         lw_add_e <= 1'b0;
423         sw_add_e <= 1'b0;
424         pc <= pc + 1;
425          end
426     4'd6: begin
427         alu_cb <= 3'b101;
428         alusrc <= 1'b1;
429         reg_we <= 1'b0;
430         lwi <= 1'b0;
431         lwe <= 1'b0;
432         swe <= 1'b0;
433         lw_add_e <= 1'b0;
434         sw_add_e <= 1'b0;
435         pc <= pc + 1;
436          end
437     4'd7:begin
438         alusrc <= 1'b0;
439         reg_we <= 1'b0;
440         sw_add_e <= 1'b0;
441         swe <= 1'b0;
442         lw_addr <= ins[10:6];
443         if (ins[11])
444         begin
445         lwei <= 1'b1;
446         pc <= pc+1;
447         end
448         else
449         begin
450         lw_add_e <= 1'b1;
451         lwei <= 1'b0;
452         lwe <= 1'b1;
453         pc <= pc;
454         end
455        end
456     4'd8:begin
457         sw_addr_d <= ins[10:6];
458         sw_addr_s <= ins[5:1];
459         alusrc <= 1'b0;
460         reg_we <= 1'b0;
461         lw_add_e <= 1'b0;
462         sw_add_e <= 1'b1;
463         lwe <= 1'b0;
464         lwi <= 1'b0;
465         pc <= pc + 1;
466        end
467     4'd9: begin
468         alusrc <= 1'b0;
```

```
469         reg_we <= 1'b0;
470         lwe <= 1'b0;
471         swe <= 1'b0;
472         lwi <= 1'b0;
473         lw_add_e <= 1'b0;
474         sw_add_e <= 1'b0;
475         pc <= ins[7:1];
476          end
477     4'd10: begin
478         alusrc <= 1'b0;
479         reg_we <= 1'b0;
480         lwi <= 1'b0;
481         lwe <= 1'b0;
482         swe <= 1'b0;
483         lw_add_e <= 1'b0;
484         sw_add_e <= 1'b0;
485         if (status_int[3])
486         pc <= ins[7:1];
487         else pc <= pc + 1;
488          end
489     4'd11: begin
490         alusrc <= 1'b0;
491         reg_we <= 1'b0;
492         lwi <= 1'b0;
493         lwe <= 1'b0;
494         swe <= 1'b0;
495         lw_add_e <= 1'b0;
496         sw_add_e <= 1'b0;
497         if (status_int[3] == 1'b0)
498         pc <= ins[7:1];
499         else pc <= pc + 1;
500          end
501     4'd12: begin
502         alusrc <= 1'b0;
503         reg_we <= 1'b0;
504         lwi <= 1'b0;
505         lwe <= 1'b0;
506         swe <= 1'b0;
507         lw_add_e <= 1'b0;
508         sw_add_e <= 1'b0;
509         if (status_int[0])
510         pc <= ins[7:1];
511         else pc <= pc + 1;
512          end
513     4'd13: begin
514         alusrc <= 1'b0;
515         reg_we <= 1'b0;
516         lwi <= 1'b0;
517         lwe <= 1'b0;
518         swe <= 1'b0;
```

```
519         lw_add_e <= 1'b0;
520         sw_add_e <= 1'b0;
521         if (status_int[2])
522         pc <= ins[7:1];
523         else pc <= pc + 1;
524          end
525     4'd14: begin
526         alusrc <= 1'b0;
527         reg_we <= 1'b0;
528         lwi <= 1'b0;
529         lwe <= 1'b0;
530         swe <= 1'b0;
531         lw_add_e <= 1'b0;
532         sw_add_e <= 1'b0;
533         ra <= pc+1;
534         pc <= ins[7:1];
535          end
536     4'd15: begin
537         alusrc <= 1'b0;
538         reg_we <= 1'b0;
539         lwi <= 1'b0;
540         lwe <= 1'b0;
541         swe <= 1'b0;
542         lw_add_e <= 1'b0;
543         sw_add_e <= 1'b0;
544         pc <= ra;
545          end
546     endcase
547     end
548     else if (nextstate == s6)
549     begin
550         lwei <= 1'b0;
551     end
552     else if (nextstate == s7)
553     begin
554         lwi <= 1'b1;
555         reg_we <= 1'b1;
556     end
557     else if (nextstate == s8)
558     begin
559         lwe <= 1'b0;
560         reg_we <= 1'b0;
561         lw_add_e <= 1'b0;
562         pc <= pc + 1;
563     end
564     else if (nextstate == s9)
565     begin
566         go <= 1'b0;
567         lw_add_e <= 1'b0;
568         sw_add_e <= 1'b0;
```

```
569         swd <= 1'b0;
570         alusrc <= 1'b0;
571         reg_we <= 1'b0;
572         lwe <= 1'b0;
573         swe <= 1'b0;
574         mem_rd <= 1'b0;
575         reg_rd <= 1'b0;
576         stop <= 1'b0;
577         done <= 1'b1;
578     end
579     end
580 
581     assign mem_we=(mem_rd)?1'b1:1'b0;
582     assign mem_addr = pc;
583 endmodule
584 
585 
586 module my_risc(
587 input logic Iclk, Ireset, Istart, Iwrb, Iaccess,
588 input logic [6:0] Iaddr,
589 input logic [15:0] Idata_in,
590 output logic [15:0] Odata_out,
591 output logic O_done
592 );
593 
594 logic clk, reset, start, wrb, access;
595 logic [6:0] addr;
596 logic [15:0] data_in;
597 logic [15:0] data_out;
598 logic done,pad_dr;
599 
600 logic we_in, mem_we,reg_we,alusrc,reg_rd,lw_add_e,sw_add_e,
        lwi,w0,w1,ri,OEB1,OEB2;
601 logic [3:0] status;
602 logic [6:0] addr_in,c_mem_addr,d_addr,daddr_in;
603 logic [15:0] mdata_in, ins, rdata_in, mdata_out, rdata_out;
604 //wire [15:0] mdata_out, rdata_out;
605 logic [2:0] alu_cb;
606 logic [4:0] lw_addr,sw_addr_d, sw_addr_s;
607 assign mdata_in = (access)?data_in : 0;
608 assign addr_in= (access)?addr : d_addr;
609 //assign addr_in= (access)?addr : 0;
610 assign ins= (access)?0 : mdata_out;
611 assign data_out = (access)? mdata_out : 0;
612 assign we_in = (access)? wrb:mem_we;
613 assign OEB1 = 1'b0;
614 assign OEB2 = 1'b0;
615 assign pad_dr = 1'b1;
616 //assign mdata_out = (OEB1) ? 16'bz : mdata_out1;
617 //assign rdata_out = (OEB2) ? 16'bz : rdata_out1;
```

```
618
619 ISH1025 pad0 ( .PADIO(Iclk), .R_EN(pad_dr), .DOUT(clk) );
620 ISH1025 pad1 ( .PADIO(Ireset), .R_EN(pad_dr), .DOUT(reset) )
        ;
621 ISH1025 pad2 ( .PADIO(Istart), .R_EN(pad_dr), .DOUT(start) )
        ;
622 ISH1025 pad3 ( .PADIO(Iwrb), .R_EN(pad_dr), .DOUT(wrb) );
623 ISH1025 pad4 ( .PADIO(Iaccess), .R_EN(pad_dr), .DOUT(access)
         );
624
625 ISH1025 pad5 ( .PADIO(Iaddr[0]), .R_EN(pad_dr), .DOUT(addr
        [0]) );
626 ISH1025 pad6 ( .PADIO(Iaddr[1]), .R_EN(pad_dr), .DOUT(addr
        [1]) );
627 ISH1025 pad7 ( .PADIO(Iaddr[2]), .R_EN(pad_dr), .DOUT(addr
        [2]) );
628 ISH1025 pad8 ( .PADIO(Iaddr[3]), .R_EN(pad_dr), .DOUT(addr
        [3]) );
629 ISH1025 pad9 ( .PADIO(Iaddr[4]), .R_EN(pad_dr), .DOUT(addr
        [4]) );
630 ISH1025 pad10 ( .PADIO(Iaddr[5]), .R_EN(pad_dr), .DOUT(addr
        [5]) );
631 ISH1025 pad11 ( .PADIO(Iaddr[6]), .R_EN(pad_dr), .DOUT(addr
        [6]) );
632
633 ISH1025 pad12 ( .PADIO(Idata_in[0]), .R_EN(pad_dr), .DOUT(
        data_in[0]) );
634 ISH1025 pad13 ( .PADIO(Idata_in[1]), .R_EN(pad_dr), .DOUT(
        data_in[1]) );
635 ISH1025 pad14 ( .PADIO(Idata_in[2]), .R_EN(pad_dr), .DOUT(
        data_in[2]) );
636 ISH1025 pad15 ( .PADIO(Idata_in[3]), .R_EN(pad_dr), .DOUT(
        data_in[3]) );
637 ISH1025 pad16 ( .PADIO(Idata_in[4]), .R_EN(pad_dr), .DOUT(
        data_in[4]) );
638 ISH1025 pad17 ( .PADIO(Idata_in[5]), .R_EN(pad_dr), .DOUT(
        data_in[5]) );
639 ISH1025 pad18 ( .PADIO(Idata_in[6]), .R_EN(pad_dr), .DOUT(
        data_in[6]) );
640 ISH1025 pad19 ( .PADIO(Idata_in[7]), .R_EN(pad_dr), .DOUT(
        data_in[7]) );
641 ISH1025 pad20 ( .PADIO(Idata_in[8]), .R_EN(pad_dr), .DOUT(
        data_in[8]) );
642 ISH1025 pad21 ( .PADIO(Idata_in[9]), .R_EN(pad_dr), .DOUT(
        data_in[9]) );
643 ISH1025 pad22 ( .PADIO(Idata_in[10]), .R_EN(pad_dr), .DOUT(
        data_in[10]) );
644 ISH1025 pad23 ( .PADIO(Idata_in[11]), .R_EN(pad_dr), .DOUT(
        data_in[11]) );
```

```
645 ISH1025 pad24 ( .PADIO(Idata_in[12]), .R_EN(pad_dr), .DOUT(
        data_in[12]) );
646 ISH1025 pad25 ( .PADIO(Idata_in[13]), .R_EN(pad_dr), .DOUT(
        data_in[13]) );
647 ISH1025 pad26 ( .PADIO(Idata_in[14]), .R_EN(pad_dr), .DOUT(
        data_in[14]) );
648 ISH1025 pad27 ( .PADIO(Idata_in[15]), .R_EN(pad_dr), .DOUT(
        data_in[15]) );
649
650 D2I1025 p_0 ( .DIN(data_out[0]), .EN(pad_dr), .PADIO(
        Odata_out[0]) );
651 D2I1025 p_1 ( .DIN(data_out[1]), .EN(pad_dr), .PADIO(
        Odata_out[1]) );
652 D2I1025 p_2 ( .DIN(data_out[2]), .EN(pad_dr), .PADIO(
        Odata_out[2]) );
653 D2I1025 p_3 ( .DIN(data_out[3]), .EN(pad_dr), .PADIO(
        Odata_out[3]) );
654 D2I1025 p_4 ( .DIN(data_out[4]), .EN(pad_dr), .PADIO(
        Odata_out[4]) );
655 D2I1025 p_5 ( .DIN(data_out[5]), .EN(pad_dr), .PADIO(
        Odata_out[5]) );
656 D2I1025 p_6 ( .DIN(data_out[6]), .EN(pad_dr), .PADIO(
        Odata_out[6]) );
657 D2I1025 p_7 ( .DIN(data_out[7]), .EN(pad_dr), .PADIO(
        Odata_out[7]) );
658 D2I1025 p_8 ( .DIN(data_out[8]), .EN(pad_dr), .PADIO(
        Odata_out[8]) );
659 D2I1025 p_9 ( .DIN(data_out[9]), .EN(pad_dr), .PADIO(
        Odata_out[9]) );
660 D2I1025 p_10 ( .DIN(data_out[10]), .EN(pad_dr), .PADIO(
        Odata_out[10]) );
661 D2I1025 p_11 ( .DIN(data_out[11]), .EN(pad_dr), .PADIO(
        Odata_out[11]) );
662 D2I1025 p_12 ( .DIN(data_out[12]), .EN(pad_dr), .PADIO(
        Odata_out[12]) );
663 D2I1025 p_13 ( .DIN(data_out[13]), .EN(pad_dr), .PADIO(
        Odata_out[13]) );
664 D2I1025 p_14 ( .DIN(data_out[14]), .EN(pad_dr), .PADIO(
        Odata_out[14]) );
665 D2I1025 p_15 ( .DIN(data_out[15]), .EN(pad_dr), .PADIO(
        Odata_out[15]) );
666
667 D2I1025 p_16 ( .DIN(done), .EN(pad_dr), .PADIO(O_done) );
668
669 //SRAM16x1024 ram(.A1(addr_in), .A2((daddr_in)) , .CE1(clk),
        .CE2(clk), .WEB1(we_in), .WEB2(ri), .OEB1(OEB1), .OEB2(
        OEB2), .I1(mdata_in), .I2(rdata_in), .O1(mdata_out), .O2(
        rdata_out));
670 SRAM16x128 ram(.A1(addr_in), .A2((daddr_in)) , .CE1(clk), .
        CE2(clk), .WEB1(we_in), .WEB2(ri), .OEB1(OEB1), .OEB2(
```

```
    OEB2), .CSB1(1'b0), .CSB2(1'b0), .I1(mdata_in), .I2(
    rdata_in), .O1(mdata_out), .O2(rdata_out));
671 control cb(.start(start), .clk(clk), .reset(reset), .status(
    status), .ins(ins), .alu_cb(alu_cb), .mem_addr(c_mem_addr
    ), .mem_we(mem_we),.reg_we(reg_we),.alusrc(alusrc),.
    reg_rd(reg_rd),.lw_add_e(lw_add_e),.sw_add_e(sw_add_e),.
    lwi(lwi),.done(done), .w0(w0), .w1(w1), .lw_addr(lw_addr)
    , .sw_addr_d(sw_addr_d), .sw_addr_s(sw_addr_s));
672 datapath dp(.clk(clk), .reg_we(reg_we), .alusrc(alusrc), .
    reg_rd(reg_rd), .lw_add_e(lw_add_e), .sw_add_e(sw_add_e),
     .lwi(lwi), .alu_cb(alu_cb), .mem_addr(c_mem_addr), .ins(
    ins), .addr(d_addr), .data(rdata_in), .status(status), .
    w0(w0), .w1(w1), .reset(reset), .lw_addr(lw_addr), .
    data_in(rdata_out), .daddr(daddr_in), .ri(ri), .sw_addr_d
    (sw_addr_d), .sw_addr_s(sw_addr_s));
673 endmodule
```

Listing 4: SystemVerilog Implementation

```
1  `timescale 1ns/10ps
2
3  module test_bench1 ();
4  logic Iclk, Ireset, Istart, Iwrb, Iaccess;
5  logic [6:0] Iaddr;
6  logic [15:0] Idata_in;
7  logic [15:0] Odata_out;
8  logic O_done;
9  my_risc dut(Iclk, Ireset, Istart, Iwrb, Iaccess, Iaddr,
       Idata_in, Odata_out, O_done);
10  always
11          begin
12                  Iclk=0; #40; Iclk=1; #40;
13          end
14          initial
15          begin
16                  Ireset=1;
17                  Istart = 0;
18          Iaccess=1;
19          Iwrb = 0;
20          Iaddr = 64;
21          Idata_in = 5;
22                  #80;
23          Ireset=0;
24          Iaddr = 65;
25          Idata_in = 7;
26          #80;
27          Iaddr = 0;
28          Idata_in = 16'h78C0; //LW R3,#64
29          #80;
30          Iaddr = 1;
```

```
31          Idata_in = 16'h0040;
32          #80;
33          Iaddr = 2;
34          Idata_in = 16'h7106; //LW R4, R3
35          #80;
36          Iaddr = 3;
37          Idata_in = 16'h18C2; // ADD R3, #1
38          #80;
39          Iaddr = 4;
40          Idata_in = 16'h7146; //LW R5, R3
41          #80;
42          Iaddr = 5;
43          Idata_in = 16'h1148; // ADD R5, R4
44          #80;
45          Iaddr = 6;
46          Idata_in = 16'h18C2; // ADD R3, #1
47          #80;
48          Iaddr = 7;
49          Idata_in = 16'h8146; // SW R5, R3
50          #80;
51          Iaddr = 8;
52          Idata_in = 16'h7186; // LW R6, R3
53          #80;
54          Iaddr = 9;
55          Idata_in = 16'h2188; // SUB R6, R4
56          #80;
57          Iaddr = 10;
58          Idata_in = 16'h18C2; // ADD R3, #1
59          #80;
60          Iaddr = 11;
61          Idata_in = 16'h8186; // SW R6, R3
62          #80;
63          Iaddr = 12;
64          Idata_in = 16'h0001; // HALT
65          #80;
66          Iaccess = 0;
67          Istart = 1;
68          Iwrb = 1;
69          #80;
70          Istart = 0;
71          #80;
72          $display("time=%5d ns, done=%b", $time, O_done);
73          #80;
74          $display("time=%5d ns, done=%b", $time, O_done);
75          #80;
76          $display("time=%5d ns, done=%b", $time, O_done);
77          #80;
78          $display("time=%5d ns, done=%b", $time, O_done);
79          #80;
80          $display("time=%5d ns, done=%b", $time, O_done);
```

```
81          #80;
82          $display("time=%5d ns, done=%b", $time, O_done);
83          #80;
84          $display("time=%5d ns, done=%b", $time, O_done);
85          #80;
86          $display("time=%5d ns, done=%b", $time, O_done);
87          #80;
88          $display("time=%5d ns, done=%b", $time, O_done);
89          #80;
90          $display("time=%5d ns, done=%b", $time, O_done);
91          #80;
92          $display("time=%5d ns, done=%b", $time, O_done);
93          #80;
94          $display("time=%5d ns, done=%b", $time, O_done);
95          #80;
96          $display("time=%5d ns, done=%b", $time, O_done);
97          #80;
98          $display("time=%5d ns, done=%b", $time, O_done);
99          #80;
100         $display("time=%5d ns, done=%b", $time, O_done);
101         #80;
102         $display("time=%5d ns, done=%b", $time, O_done);
103         #80;
104         $display("time=%5d ns, done=%b", $time, O_done);
105         #80;
106         $display("time=%5d ns, done=%b", $time, O_done);
107         #80;
108         $display("time=%5d ns, done=%b", $time, O_done);
109         #80;
110         $display("time=%5d ns, done=%b", $time, O_done);
111         #80;
112         $display("time=%5d ns, done=%b", $time, O_done);
113         #80;
114         $display("time=%5d ns, done=%b", $time, O_done);
115         #80;
116         $display("time=%5d ns, done=%b", $time, O_done);
117         #80;
118         $display("time=%5d ns, done=%b", $time, O_done);
119         #80;
120         $display("time=%5d ns, done=%b", $time, O_done);
121         #80;
122         $display("time=%5d ns, done=%b", $time, O_done);
123         #80;
124         $display("time=%5d ns, done=%b", $time, O_done);
125         #80;
126         $display("time=%5d ns, done=%b", $time, O_done);
127         #80;
128         $display("time=%5d ns, done=%b", $time, O_done);
129         #80;
130         $display("time=%5d ns, done=%b", $time, O_done);
```

```
131             #80;
132             $display("time=%5d ns, done=%b", $time, O_done);
133             #80;
134             $display("time=%5d ns, done=%b", $time, O_done);
135             #80;
136             $display("time=%5d ns, done=%b", $time, O_done);
137             #80;
138             $display("time=%5d ns, done=%b", $time, O_done);
139             #80;
140             $display("time=%5d ns, done=%b", $time, O_done);
141             #80;
142             $display("time=%5d ns, done=%b", $time, O_done);
143             #80;
144             $display("time=%5d ns, done=%b", $time, O_done);
145             #80;
146             $display("time=%5d ns, done=%b", $time, O_done);
147             #80;
148             $display("time=%5d ns, done=%b", $time, O_done);
149             #80;
150             $display("time=%5d ns, done=%b", $time, O_done);
151             #80;
152             $display("time=%5d ns, done=%b", $time, O_done);
153             #80;
154             $display("time=%5d ns, done=%b", $time, O_done);
155             #80;
156             $display("time=%5d ns, done=%b", $time, O_done);
157             #80;
158             $display("time=%5d ns, done=%b", $time, O_done);
159             #80;
160             $display("time=%5d ns, done=%b", $time, O_done);
161             #80;
162             $display("time=%5d ns, done=%b", $time, O_done);
163             Iaccess = 1;
164             Iaddr = 64;
165             #80;
166             $display("time=%5d ns, A=%16d", $time, Odata_out);
167             Iaddr = 65;
168             #80;
169             $display("time=%5d ns, B=%16d", $time, Odata_out);
170             Iaddr = 66;
171             #80;
172             $display("Checking C=A+B");
173             $display("time=%5d ns, C=%16d", $time, Odata_out);
174             Iaddr = 67;
175             if (Odata_out != 12) begin
176              $display("Error");
177              end
178              else begin
179              $display("PASS");
180              end
```

```
181         #80;
182         $display("Checking D=C-A");
183         $display("time=%5d ns, D=%16d", $time, Odata_out);
184         if (Odata_out != 7) begin
185          $display("Error");
186          end
187          else begin
188          $display("PASS");
189          end
190         $finish;
191     end
192     initial
193         begin
194                 $dumpfile ("my_risc.dump");
195                 $dumpvars (0, test_bench1);
196         end // initial begin
197
198 endmodule
199
```

Listing 5: Test Bench: Basic Arithmetic

```
1  `timescale 1ns/10ps
2
3  module test_bench2 ();
4  logic Iclk, Ireset, Istart, Iwrb, Iaccess;
5  logic [6:0] Iaddr;
6  logic [15:0] Idata_in;
7  logic [15:0] Odata_out;
8  logic O_done;
9  my_risc dut(Iclk, Ireset, Istart, Iwrb, Iaccess, Iaddr,
       Idata_in, Odata_out, O_done);
10  always
11          begin
12                  Iclk=0; #40; Iclk=1; #40;
13          end
14          initial
15          begin
16                  Ireset=1;
17                  Istart = 0;
18          Iaccess=1;
19          Iwrb = 0;
20          Iaddr = 64;
21          Idata_in = 16'h00FF;
22                  #80;
23          Ireset=0;
24          Iaddr = 65;
25          Idata_in = 16'hFF00;
26          #80;
27          Iaddr = 0;
```

```
28          Idata_in = 16'h78C0; //LW R3,#64
29          #80;
30          Iaddr = 1;
31          Idata_in = 16'h0040;
32          #80;
33          Iaddr = 2;
34          Idata_in = 16'h7106; //LW R4, R3
35          #80;
36          Iaddr = 3;
37          Idata_in = 16'h18C2; // ADD R3, #1
38          #80;
39          Iaddr = 4;
40          Idata_in = 16'h7146; //LW R5, R3
41          #80;
42          Iaddr = 5;
43          Idata_in = 16'h4148; // OR R5, R4
44          #80;
45          Iaddr = 6;
46          Idata_in = 16'h18C2; // ADD R3, #1
47          #80;
48          Iaddr = 7;
49          Idata_in = 16'h8146; // SW R5, R3
50          #80;
51          Iaddr = 8;
52          Idata_in = 16'h7186; // LW R6, R3
53          #80;
54          Iaddr = 9;
55          Idata_in = 16'h3188; // AND R6, R4
56          #80;
57          Iaddr = 10;
58          Idata_in = 16'h18C2; // ADD R3, #1
59          #80;
60          Iaddr = 11;
61          Idata_in = 16'h8186; // SW R6, R3
62          #80;
63          Iaddr = 12;
64          Idata_in = 16'h5088; // NOT R2, R4
65          #80;
66          Iaddr = 13;
67          Idata_in = 16'h18C2; // ADD R3, #1
68          #80;
69          Iaddr = 14;
70          Idata_in = 16'h8086; // SW R2, R3
71          #80;
72          Iaddr = 15;
73          Idata_in = 16'h0001; // HALT
74          #80;
75          Iaccess = 0;
76          Istart = 1;
77          Iwrb = 1;
```

```
78          #80;
79          Istart = 0;
80          #80;
81          $display("time=%5d ns, O_done=%b", $time, O_done);
82          #80;
83          $display("time=%5d ns, O_done=%b", $time, O_done);
84          #80;
85          $display("time=%5d ns, O_done=%b", $time, O_done);
86          #80;
87          $display("time=%5d ns, O_done=%b", $time, O_done);
88          #80;
89          $display("time=%5d ns, O_done=%b", $time, O_done);
90          #80;
91          $display("time=%5d ns, O_done=%b", $time, O_done);
92          #80;
93          $display("time=%5d ns, O_done=%b", $time, O_done);
94          #80;
95          $display("time=%5d ns, O_done=%b", $time, O_done);
96          #80;
97          $display("time=%5d ns, O_done=%b", $time, O_done);
98          #80;
99          $display("time=%5d ns, O_done=%b", $time, O_done);
100         #80;
101         $display("time=%5d ns, O_done=%b", $time, O_done);
102         #80;
103         $display("time=%5d ns, O_done=%b", $time, O_done);
104         #80;
105         $display("time=%5d ns, O_done=%b", $time, O_done);
106         #80;
107         $display("time=%5d ns, O_done=%b", $time, O_done);
108         #80;
109         $display("time=%5d ns, O_done=%b", $time, O_done);
110         #80;
111         $display("time=%5d ns, O_done=%b", $time, O_done);
112         #80;
113         $display("time=%5d ns, O_done=%b", $time, O_done);
114         #80;
115         $display("time=%5d ns, O_done=%b", $time, O_done);
116         #80;
117         $display("time=%5d ns, O_done=%b", $time, O_done);
118         #80;
119         $display("time=%5d ns, O_done=%b", $time, O_done);
120         #80;
121         $display("time=%5d ns, O_done=%b", $time, O_done);
122         #80;
123         $display("time=%5d ns, O_done=%b", $time, O_done);
124         #80;
125         $display("time=%5d ns, O_done=%b", $time, O_done);
126         #80;
127         $display("time=%5d ns, O_done=%b", $time, O_done);
```

```
128         #80;
129         $display("time=%5d ns, O_done=%b", $time, O_done);
130         #80;
131         $display("time=%5d ns, O_done=%b", $time, O_done);
132         #80;
133         $display("time=%5d ns, O_done=%b", $time, O_done);
134         #80;
135         $display("time=%5d ns, O_done=%b", $time, O_done);
136         #80;
137         $display("time=%5d ns, O_done=%b", $time, O_done);
138         #80;
139         $display("time=%5d ns, O_done=%b", $time, O_done);
140         #80;
141         $display("time=%5d ns, O_done=%b", $time, O_done);
142         #80;
143         $display("time=%5d ns, O_done=%b", $time, O_done);
144         #80;
145         $display("time=%5d ns, O_done=%b", $time, O_done);
146         #80;
147         $display("time=%5d ns, O_done=%b", $time, O_done);
148         #80;
149         $display("time=%5d ns, O_done=%b", $time, O_done);
150         #80;
151         $display("time=%5d ns, O_done=%b", $time, O_done);
152         #80;
153         $display("time=%5d ns, O_done=%b", $time, O_done);
154         #80;
155         $display("time=%5d ns, O_done=%b", $time, O_done);
156         #80;
157         $display("time=%5d ns, O_done=%b", $time, O_done);
158         #80;
159         $display("time=%5d ns, O_done=%b", $time, O_done);
160         #80;
161         $display("time=%5d ns, O_done=%b", $time, O_done);
162         #80;
163         $display("time=%5d ns, O_done=%b", $time, O_done);
164         #80;
165         $display("time=%5d ns, O_done=%b", $time, O_done);
166         #80;
167         $display("time=%5d ns, O_done=%b", $time, O_done);
168         #80;
169         $display("time=%5d ns, O_done=%b", $time, O_done);
170         #80;
171         $display("time=%5d ns, O_done=%b", $time, O_done);
172         #80;
173         $display("time=%5d ns, O_done=%b", $time, O_done);
174         #80;
175         $display("time=%5d ns, O_done=%b", $time, O_done);
176         #80;
177         $display("time=%5d ns, O_done=%b", $time, O_done);
```

```
178         #80;
179         $display("time=%5d ns, O_done=%b", $time, O_done);
180         #80;
181         $display("time=%5d ns, O_done=%b", $time, O_done);
182         Iaccess = 1;
183         Iaddr = 64;
184         #80;
185         $display("time=%5d ns, A=%4h", $time, Odata_out);
186         Iaddr = 65;
187         #80;
188         $display("time=%5d ns, B=%4h", $time, Odata_out);
189         Iaddr = 66;
190         #80;
191         $display("Checking C=A|B");
192         $display("time=%5d ns, C=%4h", $time, Odata_out);
193         Iaddr = 67;
194         if (Odata_out != 16'hffff) begin
195          $display("Error");
196          end
197          else begin
198          $display("PASS");
199          end
200         #80;
201         $display("Checking D=C&A");
202         $display("time=%5d ns, D=%4h", $time, Odata_out);
203         Iaddr = 68;
204         if (Odata_out != 16'h00ff) begin
205          $display("Error");
206          end
207          else begin
208          $display("PASS");
209          end
210         #80;
211         $display("Checking E=!A");
212         $display("time=%5d ns, E=%4h", $time, Odata_out);
213         if (Odata_out != 16'hff00) begin
214          $display("Error");
215          end
216          else begin
217          $display("PASS");
218          end
219         $finish;
220     end
221     initial
222         begin
223                 $dumpfile ("my_risc.dump");
224                 $dumpvars (0, test_bench2);
225         end // initial begin
226
227 endmodule
```

```
228
```

Listing 6: Test Bench: Basic Logical

```
1  `timescale 1ns/10ps
2
3  module test_bench3 ();
4  logic Iclk, Ireset, Istart, Iwrb, Iaccess;
5  logic [6:0] Iaddr;
6  logic [15:0] Idata_in;
7  logic [15:0] Odata_out;
8  logic O_done;
9  my_risc dut(Iclk, Ireset, Istart, Iwrb, Iaccess, Iaddr,
       Idata_in, Odata_out, O_done);
10  always
11          begin
12                  Iclk=0; #40; Iclk=1; #40;
13          end
14          initial
15          begin
16                  Ireset=1;
17                  Istart = 0;
18         Iaccess=1;
19         Iwrb = 0;
20         Iaddr = 64;
21         Idata_in = 3;
22                  #80;
23         Ireset=0;
24         Iaddr = 65;
25         Idata_in = 12;
26         #80;
27         Iaddr = 0;
28         Idata_in = 16'h78C0; //LW R3,#64
29         #80;
30         Iaddr = 1;
31         Idata_in = 16'h0040;
32         #80;
33         Iaddr = 2;
34         Idata_in = 16'h7106; //LW R4, R3
35         #80;
36         Iaddr = 3;
37         Idata_in = 16'h18C2; // ADD R3, #1
38         #80;
39         Iaddr = 4;
40         Idata_in = 16'h7146; //LW R5, R3
41         #80;
42         Iaddr = 5;
43         Idata_in = 16'hE012; // CALL DIV(#9)
44         #80;
45         Iaddr = 6;
```

```
46          Idata_in = 16'h18C2; // ADD R3, #1
47          #80;
48          Iaddr = 7;
49          Idata_in = 16'h8186; // SW R6, R3
50          #80;
51          Iaddr = 8;
52          Idata_in = 16'h0001; // HALT
53          #80;
54          Iaddr = 9;
55          Idata_in = 16'h3180; // DIV: AND R6, R0
56          #80;
57          Iaddr = 10;
58          Idata_in = 16'h6148; // CMP R5, R4
59          #80;
60          Iaddr = 11;
61          Idata_in = 16'hC01A; // JGE L1(#13)
62          #80;
63          Iaddr = 12;
64          Idata_in = 16'hD028; // JL L2(#20)
65          #80;
66          Iaddr = 13;
67          Idata_in = 16'h6140; // L1: CMP R5,R0
68          #80;
69          Iaddr = 14;
70          Idata_in = 16'hA020; // JZ DONE(#16)
71          #80;
72          Iaddr = 15;
73          Idata_in = 16'hB022; // JNZ SUB1(#17)
74          #80;
75          Iaddr = 16;
76          Idata_in = 16'hF000; // DONE: RET
77          #80;
78          Iaddr = 17;
79          Idata_in = 16'h2148; // SUB1: SUB R5, R4
80          #80;
81          Iaddr = 18;
82          Idata_in = 16'h1982; // ADD R6, #1
83          #80;
84          Iaddr = 19;
85          Idata_in = 16'h901A; // JMP L1(#13)
86          #80;
87          Iaddr = 20;
88          Idata_in = 16'h6100; // L2: CMP R4, R0
89          #80;
90          Iaddr = 21;
91          Idata_in = 16'hA020; // JZ DONE(#16)
92          #80;
93          Iaddr = 22;
94          Idata_in = 16'hB02E; // JNZ SUB2(#22)
95          #80;
```

```
96          Iaddr = 23;
97          Idata_in = 16'h210A; // SUB2: SUB R4,R5
98          #80;
99          Iaddr = 24;
100         Idata_in = 16'h1982; // ADD R6,#1
101         #80;
102         Iaddr = 25;
103         Idata_in = 16'h9028; // JMP L2(#20)
104         #80
105         Iaccess = 0;
106         Istart = 1;
107         Iwrb = 1;
108         #80;
109         Istart = 0;
110         $display("time=%5d ns, O_done=%b", $time, O_done);
111         #80;
112         $display("time=%5d ns, O_done=%b", $time, O_done);
113         #80;
114         $display("time=%5d ns, O_done=%b", $time, O_done);
115         #80;
116         $display("time=%5d ns, O_done=%b", $time, O_done);
117         #80;
118         $display("time=%5d ns, O_done=%b", $time, O_done);
119         #80;
120         $display("time=%5d ns, O_done=%b", $time, O_done);
121         #80;
122         $display("time=%5d ns, O_done=%b", $time, O_done);
123         #80;
124         $display("time=%5d ns, O_done=%b", $time, O_done);
125         #80;
126         $display("time=%5d ns, O_done=%b", $time, O_done);
127         #80;
128         $display("time=%5d ns, O_done=%b", $time, O_done);
129         #80;
130         $display("time=%5d ns, O_done=%b", $time, O_done);
131         #80;
132         $display("time=%5d ns, O_done=%b", $time, O_done);
133         #80;
134         $display("time=%5d ns, O_done=%b", $time, O_done);
135         #80;
136         $display("time=%5d ns, O_done=%b", $time, O_done);
137         #80;
138         $display("time=%5d ns, O_done=%b", $time, O_done);
139         #80;
140         $display("time=%5d ns, O_done=%b", $time, O_done);
141         #80;
142         $display("time=%5d ns, O_done=%b", $time, O_done);
143         #80;
144         $display("time=%5d ns, O_done=%b", $time, O_done);
145         #80;
```

```
146             $display("time=%5d ns, O_done=%b", $time, O_done);
147             #80;
148             $display("time=%5d ns, O_done=%b", $time, O_done);
149             #80;
150             $display("time=%5d ns, O_done=%b", $time, O_done);
151             #80;
152             $display("time=%5d ns, O_done=%b", $time, O_done);
153             #80;
154             $display("time=%5d ns, O_done=%b", $time, O_done);
155             #80;
156             $display("time=%5d ns, O_done=%b", $time, O_done);
157             #80;
158             $display("time=%5d ns, O_done=%b", $time, O_done);
159             #80;
160             $display("time=%5d ns, O_done=%b", $time, O_done);
161             #80;
162             $display("time=%5d ns, O_done=%b", $time, O_done);
163             #80;
164             $display("time=%5d ns, O_done=%b", $time, O_done);
165             #80;
166             $display("time=%5d ns, O_done=%b", $time, O_done);
167             #80;
168             $display("time=%5d ns, O_done=%b", $time, O_done);
169             #80;
170             $display("time=%5d ns, O_done=%b", $time, O_done);
171             #80;
172             $display("time=%5d ns, O_done=%b", $time, O_done);
173             #80;
174             $display("time=%5d ns, O_done=%b", $time, O_done);
175             #80;
176             $display("time=%5d ns, O_done=%b", $time, O_done);
177             #80;
178             $display("time=%5d ns, O_done=%b", $time, O_done);
179             #80;
180             $display("time=%5d ns, O_done=%b", $time, O_done);
181             #80;
182             $display("time=%5d ns, O_done=%b", $time, O_done);
183             #80;
184             $display("time=%5d ns, O_done=%b", $time, O_done);
185             #80;
186             $display("time=%5d ns, O_done=%b", $time, O_done);
187             #80;
188             $display("time=%5d ns, O_done=%b", $time, O_done);
189             #80;
190             $display("time=%5d ns, O_done=%b", $time, O_done);
191             #80;
192             $display("time=%5d ns, O_done=%b", $time, O_done);
193             #80;
194             $display("time=%5d ns, O_done=%b", $time, O_done);
195             #80;
```

```
196             $display("time=%5d ns, O_done=%b", $time, O_done);
197             #80;
198             $display("time=%5d ns, O_done=%b", $time, O_done);
199             #80;
200             $display("time=%5d ns, O_done=%b", $time, O_done);
201             #80;
202             $display("time=%5d ns, O_done=%b", $time, O_done);
203             #80;
204             $display("time=%5d ns, O_done=%b", $time, O_done);
205             #80;
206             $display("time=%5d ns, O_done=%b", $time, O_done);
207             #80;
208             $display("time=%5d ns, O_done=%b", $time, O_done);
209             #80;
210             $display("time=%5d ns, O_done=%b", $time, O_done);
211             #80;
212             $display("time=%5d ns, O_done=%b", $time, O_done);
213             #80;
214             $display("time=%5d ns, O_done=%b", $time, O_done);
215             #80;
216             $display("time=%5d ns, O_done=%b", $time, O_done);
217             #80;
218             $display("time=%5d ns, O_done=%b", $time, O_done);
219             #80;
220             $display("time=%5d ns, O_done=%b", $time, O_done);
221             #80;
222             $display("time=%5d ns, O_done=%b", $time, O_done);
223             #80;
224             $display("time=%5d ns, O_done=%b", $time, O_done);
225             #80;
226             $display("time=%5d ns, O_done=%b", $time, O_done);
227             #80;
228             $display("time=%5d ns, O_done=%b", $time, O_done);
229             #80;
230             $display("time=%5d ns, O_done=%b", $time, O_done);
231             #80;
232             $display("time=%5d ns, O_done=%b", $time, O_done);
233             #80;
234             $display("time=%5d ns, O_done=%b", $time, O_done);
235             #80;
236             $display("time=%5d ns, O_done=%b", $time, O_done);
237             #80;
238             $display("time=%5d ns, O_done=%b", $time, O_done);
239             #80;
240             $display("time=%5d ns, O_done=%b", $time, O_done);
241             #80;
242             $display("time=%5d ns, O_done=%b", $time, O_done);
243             #80;
244             $display("time=%5d ns, O_done=%b", $time, O_done);
245             #80;
```

```
246             $display("time=%5d ns, O_done=%b", $time, O_done);
247             #80;
248             $display("time=%5d ns, O_done=%b", $time, O_done);
249             #80;
250             $display("time=%5d ns, O_done=%b", $time, O_done);
251             #80;
252             $display("time=%5d ns, O_done=%b", $time, O_done);
253             #80;
254             $display("time=%5d ns, O_done=%b", $time, O_done);
255             #80;
256             $display("time=%5d ns, O_done=%b", $time, O_done);
257             #80;
258             $display("time=%5d ns, O_done=%b", $time, O_done);
259             #80;
260             $display("time=%5d ns, O_done=%b", $time, O_done);
261             #80;
262             $display("time=%5d ns, O_done=%b", $time, O_done);
263             #80;
264             $display("time=%5d ns, O_done=%b", $time, O_done);
265             #80;
266             $display("time=%5d ns, O_done=%b", $time, O_done);
267             #80;
268             $display("time=%5d ns, O_done=%b", $time, O_done);
269             #80;
270             $display("time=%5d ns, O_done=%b", $time, O_done);
271             #80;
272             $display("time=%5d ns, O_done=%b", $time, O_done);
273             #80;
274             $display("time=%5d ns, O_done=%b", $time, O_done);
275             #80;
276             $display("time=%5d ns, O_done=%b", $time, O_done);
277             #80;
278             $display("time=%5d ns, O_done=%b", $time, O_done);
279             #80;
280             $display("time=%5d ns, O_done=%b", $time, O_done);
281             #80;
282             $display("time=%5d ns, O_done=%b", $time, O_done);
283             #80;
284             $display("time=%5d ns, O_done=%b", $time, O_done);
285             #80;
286             $display("time=%5d ns, O_done=%b", $time, O_done);
287             #80;
288             $display("time=%5d ns, O_done=%b", $time, O_done);
289             #80;
290             $display("time=%5d ns, O_done=%b", $time, O_done);
291             #80;
292             $display("time=%5d ns, O_done=%b", $time, O_done);
293             #80;
294             $display("time=%5d ns, O_done=%b", $time, O_done);
295             #80;
```

```
296         $display("time=%5d ns, O_done=%b", $time, O_done);
297         #80;
298         $display("time=%5d ns, O_done=%b", $time, O_done);
299         #80;
300         $display("time=%5d ns, O_done=%b", $time, O_done);
301         Iaccess = 1;
302         Iaddr = 64;
303         #80;
304         $display("time=%5d ns, A=%16d", $time, Odata_out);
305         Iaddr = 65;
306         #80;
307         $display("time=%5d ns, B=%16d", $time, Odata_out);
308         Iaddr = 66;
309         #80;
310         $display("Checking C=B/A");
311         $display("time=%5d ns, C=%16d", $time, Odata_out);
312         if (Odata_out != 4) begin
313          $display("Error");
314          end
315          else begin
316          $display("PASS");
317          end
318         $finish;
319     end
320     initial
321         begin
322                 $dumpfile ("my_risc.dump");
323                 $dumpvars (0, test_bench3);
324         end // initial begin
325
326 endmodule
327
```

Listing 7: Test Bench: Division by Subtraction