Shane Cincotta

CPEG324

LAB 3

5/7/19

Lab 3: A Single Cycle Calculator in VHDL

Abstract:

The main goal of this lab was to implement my 8-Bit calculator ISA into a single cycle calculator in VHDL, as well as developing a testbench for the calculator. The 8-Bit ISA was previously designed in lab 1. This lab reinforces various skills such as, understanding data paths for a single cycle CPU, combining multiple data paths, developing schematic at the register transfer level, implementing the schematic in VHDL, as well as developing a testbench in VHDL to test the schematic.

Using these skills, I was able to successfully implement my 8-Bit ISA design into a single cycle CPU. The calculator was created by first creating the data paths for each individual component (such as add, subtract, load immediate, etc) then the data paths were combined into a complete data path. This data path was then translated into a schematic, which was then translated to VHDL code.

Division of Labor:

I do not have a partner for this class, as such I did the lab myself.

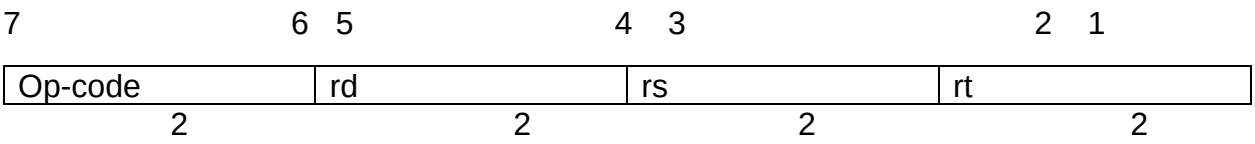
Detailed Strategy:

Before I could begin, I first had to remake my ISA design from lab 1, specifically the opcode portion. I had a conflict between the op code for my print and add commands. My new table of op codes is:

ISA Design

Op-code	Instruction
01	Add
10	Sub
11	LI
001	Compare
000	Print

Add:



Format: Add rd, rs, rt

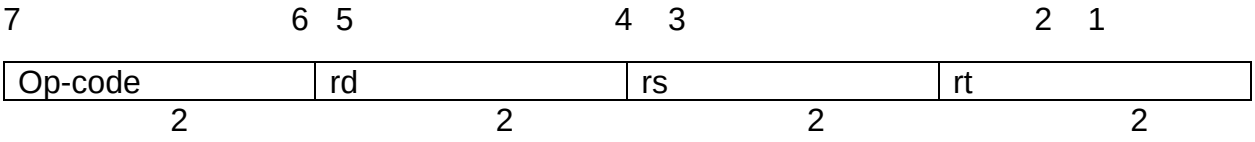
Purpose: To add 2-bit integers

Description: rd = rs + rt

The 8-bit value in register *rs* is added to the 8-bit value in register *rt* to produce a 8-bit result.

- If the addition results in an integer overflow, the destination register is not modified, an exception occurs.
- If the addition does not result in an overflow, the 8-bit result is placed into register *rd*.

Sub:



Format: Sub rd, rs, rt

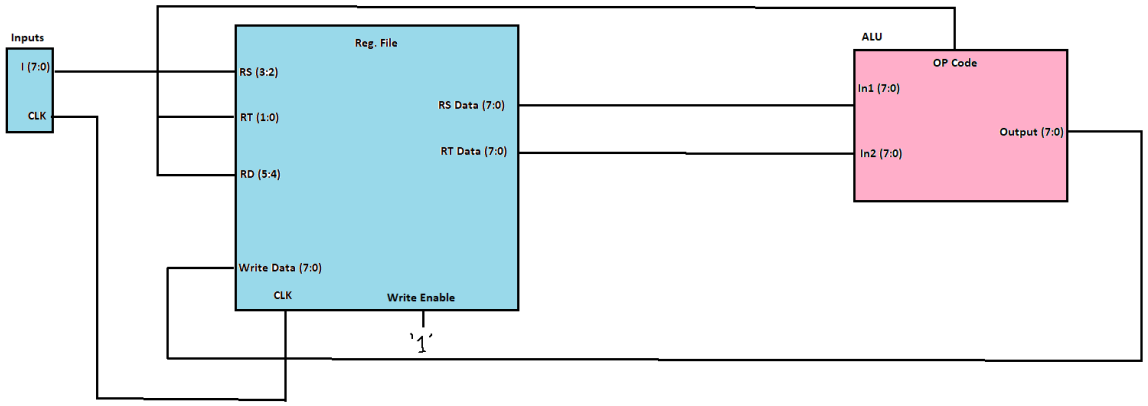
Purpose: To subtract 8-bit integers

Description: rd = rs – rt

The 8-bit value in register *rt* is subtracted from the 8-bit value in register *rs* to produce an 8-bit result.

- If the subtraction results in an integer overflow, the destination register is not modified, an exception occurs.
- If the subtraction does not result in an overflow, the 8-bit result is placed into register *rd*.

Data Path for Add/Sub Commands



Registers RS comes from bits 3/2, RT comes from bits 1/0 and RD comes from bits 5/4. The opcode is sent to the ALU, this determines which command to run (either addition or subtraction). The output of the ALU is the sent to the "Write Data" input of the reg file. The corresponding VHDL files are *"add-sub.vhdl"* (Fig 1) and *"regFile.vhdl"* (Fig 2). *Add-sub.vhdl* implements the logic of the ALU, while *regFile.vhdl* handles the logic of the register file. The ALU was made using 8 full adders (each comprised of 2 half adders).

LI:

7 6	5 4	3 0
Op-code	rd	Signed Immediate
2	2	4

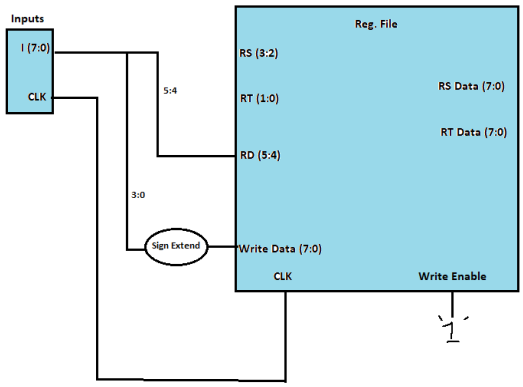
Format: LI rd, immediate

Purpose: To load an 4-bit immediate value into an 8-bit register

Description: rd = rd + immediate

- A 4-bit *immediate* value is sign extended and placed in the 8-bit register *rd*. The sign extension is performed by using *ori* which puts a constant in the least significant bits of *rd*.
- If an overflow occurs, an exception is thrown

Data Path for LI Command



Register RD comes from bits 5/4 and the immediate value comes from bits 3-0. The immediate value is sign extended; this value is then sent to the Write Data input port in the reg file. The VHDL code for the reg file is the same as used in the add/sub commands. The sign extension was preformed by using a with/select statement.

```
WD_sel <= not(I(7) and I(6));
with WD_sel select Write_Data <=
  extended_immediate when '0', --sign extended value written to RD
  ALU_OUTPUT when others; --ALU result written to RD
```

Compare:

7	5 4	4 3	2 1
Op-code	Skip (0 Skip 1, 1 skip 2)	rs	rt
3	1	2	2

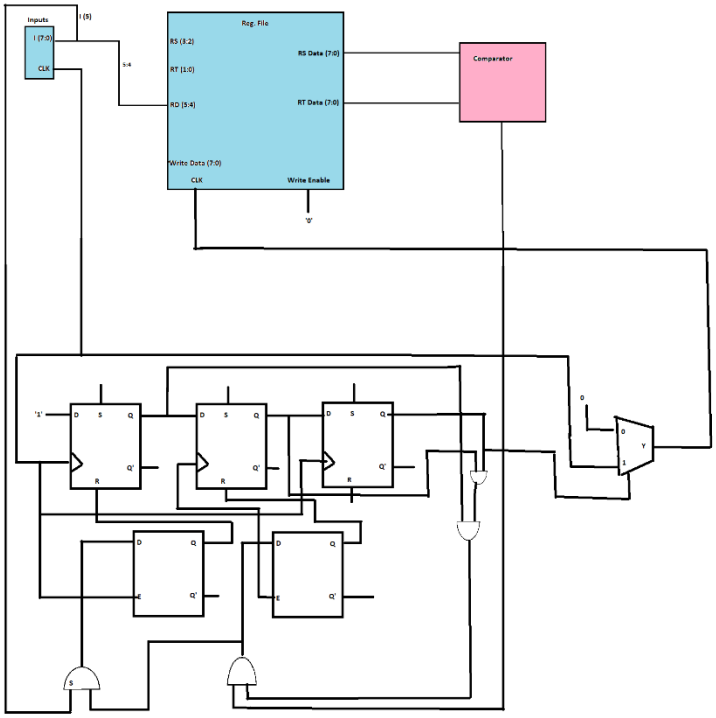
Format: compare rs, rt

Purpose: Compare two registers. If they are not equal, execute the next instruction. If equal, the choice exists of either skipping either the next 1 or the next 2 instructions. If RS != RT, no instructions are skipped. If RS=RT, 1 instruction is skipped if S=1, else 2 instructions are skipped.

Description:

Compares the result of using instruction *and* on registers *rs* and *rt*

Data Path for Compare Command



The compare operation is made possible due to the 3 flip-flops as shown in the schematic. The flip flops act as a buffer to the clock mux. If the next instruction is to be skipped, a 0 is inserted into the flip-flop. If the value of *S* is 1, then 2 zeroes are inserted to the farthest left flip-flops. This will lead to the next 2 instructions being skipped. The purpose of the D-latches is to make sure that the flip-flops are only reset when the clock signal is 1. The actual comparison was done by using the *xnor* command on the bits of the RS and RT registers

```
--checks opcode and skip bit
skip <= (not I(7)) and (not I(6)) and I(5) and skip_compare;

--Compare registers to determine if Write_Enable should skip or not
skip_compare <= (RS_DATA(7) xnor RT_DATA(7)) and
          (RS_DATA(6) xnor RT_DATA(6)) and
          (RS_DATA(5) xnor RT_DATA(5)) and
          (RS_DATA(4) xnor RT_DATA(4)) and
          (RS_DATA(3) xnor RT_DATA(3)) and
          (RS_DATA(2) xnor RT_DATA(2)) and
          (RS_DATA(1) xnor RT_DATA(1)) and
          (RS_DATA(0) xnor RT_DATA(0)
          );
```

Print:

7	5	4	3	2	0
Op-code		rs		Don't Care	
3		2		3	

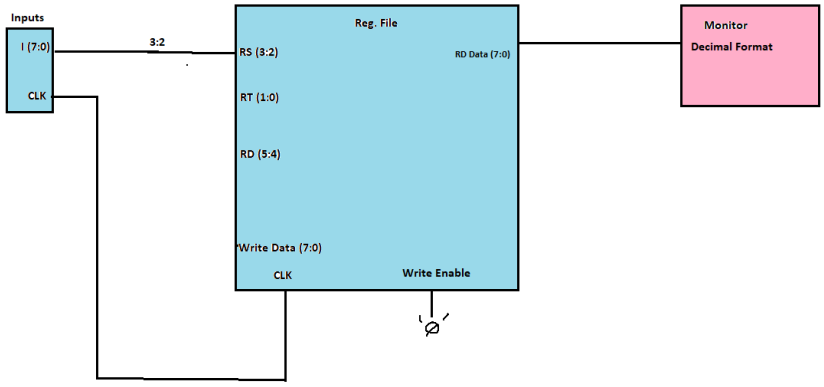
Format:print rs

Purpose: Display a registers content to the console.

Description:

Checks the 8-bit value stored in register *rs* and prints the value to the console

Data Path for Print Command



This command prints the conents of register RS, which is given by bits 3/2. The value is converted to an integer and then printed using the "report" command within *Calc.vhdl* (Fig 4).

```
  --Prints value to display
  process(filtered_clk, display) is
    variable value : integer;
    begin
      if((filtered_clk'event and filtered_clk = '1') and (display = '1')) then
        value := to_integer(signed(RS_DATA));
        report integer'image(value) severity note; --print the value
      end if;
  end process;
end architecture structural;
-- ----------------------------------------------------------------------
```

Results

Once the data path was implemented in VHDL, a testbench was made to test different combinations of instructions. The testbench includes a series of instructions for the calculator (in binary). Multiple instructions are tested in sequence with each other for error testing. What this means is that, for example, a LI instruction may be ran, and then following that an add command may be ran. This is repeated many times with different combinations of commands. This is to test how the commands interact with each other, and to see if there are any conflicts between the instructions. When I ran my testbench, I noticed that there was a conflict with my add and print instructions. When designing my ISA, I had created a conflict between the opcodes of my add/sub and print commands. I had to change the format of my opcodes in my ISA design so there was no longer a conflict.

Conclusion

Overall, I was successfully able to implement my 8-bit calculator ISA design into functional VHDL code. I began by using my original design from lab 1 (the ISA design), I continually added data paths for the various instructions. When I had all my data paths, I began translating them into VHDL code. While making my components and testing my calculator I encountered two main problems, conflicts between my opcodes and creating the skip instruction. The opcode conflict was easily fixed by recreating the ISA design from lab 1. The skip instruction was solved by using a combination of D-latches and flip-flops and using the example logic for the skip instruction as was shown in class.

Appendix I:

Included are the figures cited in the above report

Fig 1: Add-sub.vhdl

C:\Users\Shane\School Classes\CPEG 324 (System Design II)\Labs\Lab 3\add-sub.vhdl - Sublime Text (UNREGISTERED)

File Edit Selection Find View Goto Tools Project Preferences Help

add-sub.vhdl | Calc.vhdl | regFile.vhdl | clock-filter.vhdl

```
1  --Created by Shane Cincotta 5/6/19
2  --8 Bit Adder/Subtractor
3
4  library ieee;
5  use ieee.std_logic_1164.all;
6
7  -- ----------------------------------------------------------------------
8  --add-sub entity
9
10 entity add-sub is
11     port(in_1, in_2 : in std_logic_vector(7 downto 0); --in_1 and in_2 are the operands
12         mode_sel : in std_logic; --controls if we're doing addition or subtraction, 0 if addition, 1 if subtraction
13         total : out std_logic_vector(7 downto 0)); --result of doing the addition or subtraction
14 end entity add-sub;
15
16 -- ----------------------------------------------------------------------
17 --add-sub architecture
18
19 architecture structural of add-sub is
20 component 8bit-adder is
21     port(in_1, in_2 : in std_logic_vector(7 downto 0); --8-bit inputs
22          total : out std_logic_vector(7 downto 0)); --8bit output
23 end component 8bit-adder;
24
25
26 signal second_term, in_2_inverse, in_2_negative : std_logic_vector(7 downto 0); --make signals
27 constant one : std_logic_vector(7 downto 0) := "00000001"; --used for 2's compliment
28
29
30 begin
31   8bit-adder_0: 8bit-adder port map(in_1, second_term, total); --Preform the addition
32
33   8bit-adder_1: 8bit-adder port map(in_2_inverse, one, in_2_negative); -- Used for flipping sign of second term.
34
35   in_2_inverse <= not(in_2);
36
37   with mode_sel select second_term <=
38       in_2 when '0', --If mode_sel is 0, we just do regular addition with the two inputs
39       in_2_negative when others; --if mode_sel is 1, we want to set in_2 to it's negative value: a-b = a + (-b)
40 end architecture structural;
41
42 -- ----------------------------------------------------------------------
43
44
45 -- ----------------------------------------------------------------------
46 --8-Bit adder entity
47
48 entity 8bit-adder is
49     port(in_1, in_2 : in std_logic_vector(7 downto 0);
50          total : out std_logic_vector(7 downto 0));
51 end entity 8bit-adder;
52
```

C:\Users\Shane\School Classes\CPEG 324 (System Design II)\Labs\Lab 3\add-sub.vhdl - Sublime Text (UNREGISTERED)

File Edit Selection Find View Goto Tools Project Preferences Help

add-sub.vhdl | Calc.vhdl | regFile.vhdl | clock-filter.vhdl

```
52
53  -- ----------------------------------------------------------------------
54  --8-Bit adder architecture
55
56  architecture structural of 8bit-adder is
57  component full_adder is
58      port(a, b, carry_in : in std_logic;
59          total, carry_out : out std_logic);
60  end component full_adder;
61
62  signal carry_0, carry_1, carry_2, carry_3, carry_4, carry_5, carry_6 : std_logic; --carry signals for adders
63
64  --set of 8 full adders which comprise the add-sub command.  Each full adder is made up of 2 half adders.
65  --(a(in), b(in), carry_in(in), total(out), carry_out(out))
66  begin
67      fullAdder0: full_adder port map(in_1(0), in_2(0),'0', total(0), carry_0); --carry_in for the first full_adder is always 0
68      fullAdder1: full_adder port map(in_1(1), in_2(1), carry_0, total(1), carry_1);
69      fullAdder2: full_adder port map(in_1(2), in_2(2), carry_1, total(2), carry_2);
70      fullAdder3: full_adder port map(in_1(3), in_2(3), carry_2, total(3), carry_3);
71      fullAdder4: full_adder port map(in_1(4), in_2(4), carry_3, total(4), carry_4);
72      fullAdder5: full_adder port map(in_1(5), in_2(5), carry_4, total(5), carry_5);
73      fullAdder6: full_adder port map(in_1(6), in_2(6), carry_5, total(6), carry_6);
74      fullAdder7: full_adder port map(in_1(7), in_2(7), carry_6, total(7), open
75      );
76
77  end architecture structural;
78
79  -- ----------------------------------------------------------------------
80  --Full Adder Entity
81
82  entity full_adder is
83      port(a, b, carry_in : in std_logic;
84          total, carry_out : out std_logic);
85  end entity full_adder;
86
87  -- ----------------------------------------------------------------------
88  --Full Adder architecture
89
90  architecture structural of full_adder is
91  component half_adder is
92      port(a, b : in std_logic;
93          total, carry : out std_logic);
94  end component half_adder;
95
96  signal sig_1, sig_2, sig_3 : std_logic;
97  begin                   --(a(in), b(in), total(out), carry_out(out))
98      h1: half_adder port map(a, b, sig_1, sig_3);
99      h2: half_adder port map(sig_1, carry_in, total, sig_2);
100     carry_out <= sig_2 or sig_3;
101 end architecture structural;
```

Fig 2: regFile.vhdl

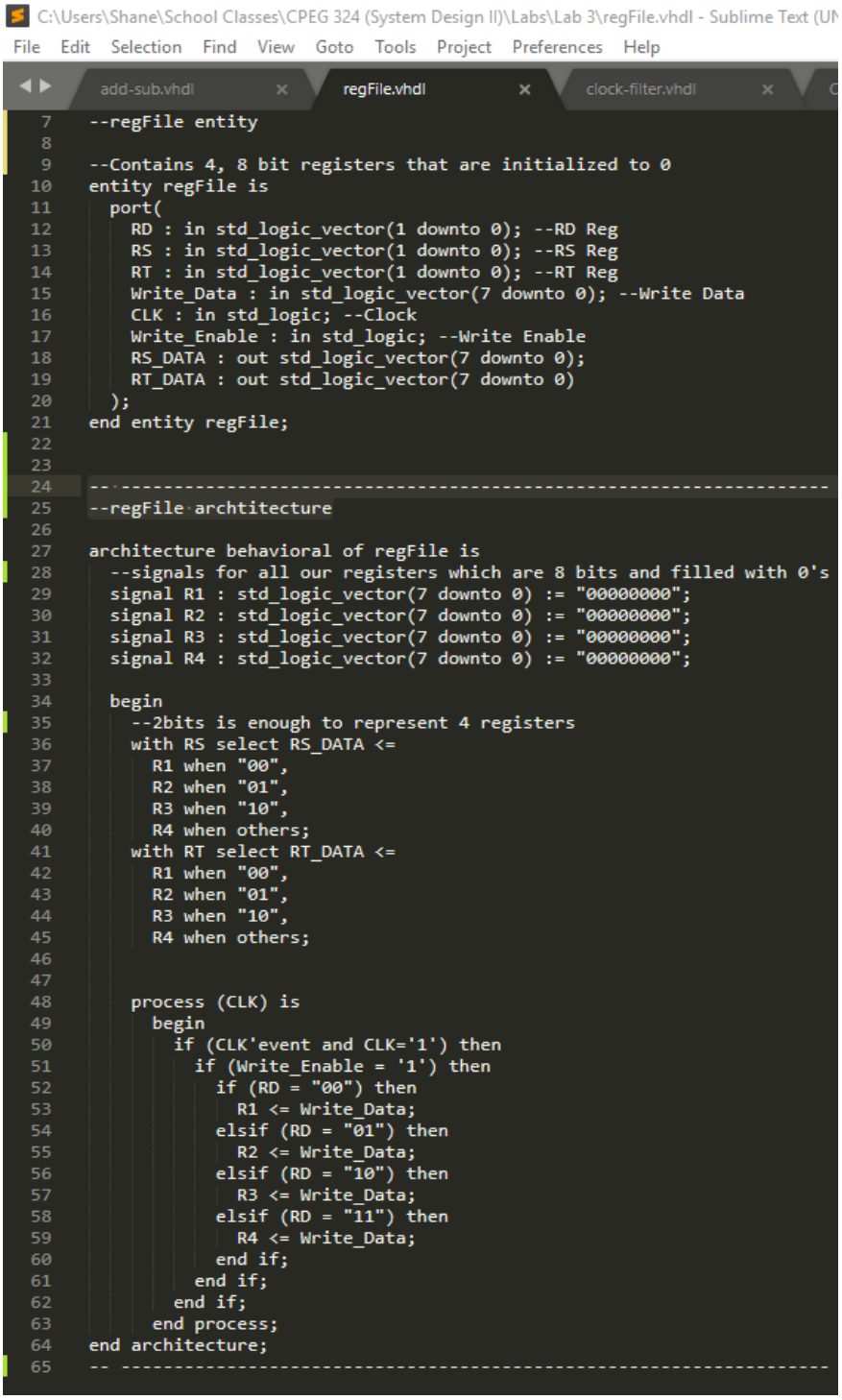
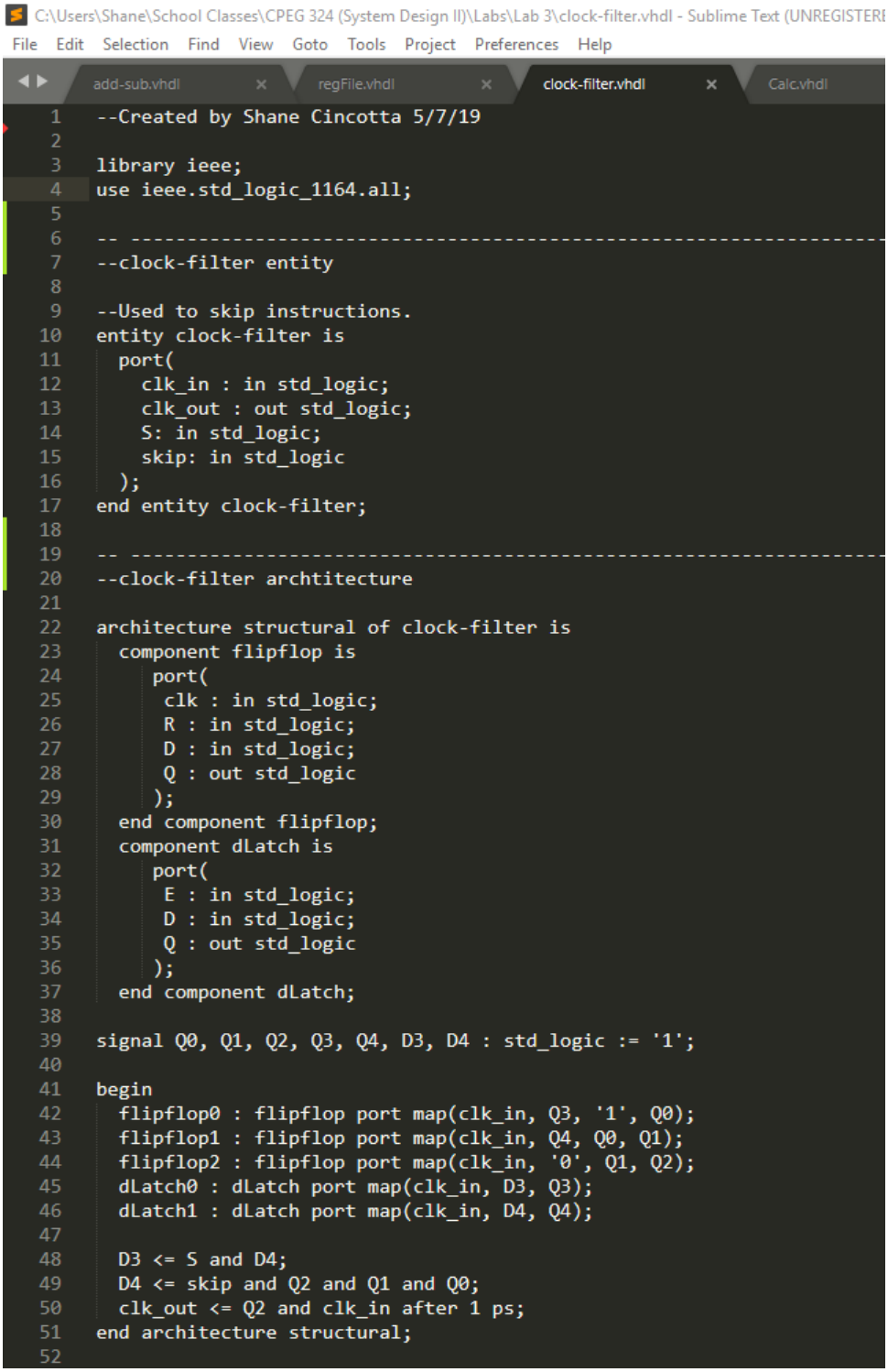


Fig 3: clock-filter.vhdl




```
1  --Created by Shane Cincotta 5/7/19
2
3  library ieee;
4  use ieee.std_logic_1164.all;
5
6  -- ------------------------------------------------------------------
7  --clock-filter entity
8
9  --Used to skip instructions.
10 entity clock-filter is
11   port(
12     clk_in : in std_logic;
13     clk_out : out std_logic;
14     S: in std_logic;
15     skip: in std_logic
16   );
17 end entity clock-filter;
18
19 -- ------------------------------------------------------------------
20 --clock-filter archtitecture
21
22 architecture structural of clock-filter is
23   component flipflop is
24      port(
25       clk : in std_logic;
26       R : in std_logic;
27       D : in std_logic;
28       Q : out std_logic
29      );
30   end component flipflop;
31   component dLatch is
32      port(
33       E : in std_logic;
34       D : in std_logic;
35       Q : out std_logic
36      );
37   end component dLatch;
38
39 signal Q0, Q1, Q2, Q3, Q4, D3, D4 : std_logic := '1';
40
41 begin
42   flipflop0 : flipflop port map(clk_in, Q3, '1', Q0);
43   flipflop1 : flipflop port map(clk_in, Q4, Q0, Q1);
44   flipflop2 : flipflop port map(clk_in, '0', Q1, Q2);
45   dLatch0 : dLatch port map(clk_in, D3, Q3);
46   dLatch1 : dLatch port map(clk_in, D4, Q4);
47
48   D3 <= S and D4;
49   D4 <= skip and Q2 and Q1 and Q0;
50   clk_out <= Q2 and clk_in after 1 ps;
51 end architecture structural;
52
```

C:\Users\Shane\School Classes\CPEG 324 (System Design II)\Labs\Lab 3\clock-filter.vhdl - Sublime Text (UNREGISTERED)

File Edit Selection Find View Goto Tools Project Preferences Help

add-sub.vhdl | regFile.vhdl | clock-filter.vhdl | Calc.vhdl

```
53  -- ----------------------------------------------------------------------
54  --Flip-flop entity
55
56  --D Flip-Flop
57  entity flipflop is
58     port(
59        clk : in std_logic;
60        R : in std_logic;
61        D : in std_logic;
62        Q : out std_logic
63     );
64  end entity flipflop;
65
66  -- ----------------------------------------------------------------------
67  --Flip-flop archtitecture
68
69  architecture behavioral of flipflop is
70    signal qt : std_logic :='1';
71  begin
72     process (clk,R) is
73     begin
74        if (R = '1') then
75          qt <= '0';
76        elsif clk'event and clk = '1' then
77            qt <= D;
78        end if;
79     end process;
80     Q<=qt;
81  end architecture behavioral;
82
83  -- ----------------------------------------------------------------------
84  --D-latch entity
85
86  --D-latch
87  entity dLatch is
88     port(
89        E : in std_logic;
90        D : in std_logic;
91        Q : out std_logic
92     );
93  end entity dLatch;
94
95  -- ----------------------------------------------------------------------
96  --D-latch archtitecture
97
98  architecture behavioral of dLatch is
99    signal t : std_logic := '0';
100 begin
101    with E select t<=
102     D when '1',
103     t when others;
104    Q<=t;
105 end architecture behavioral;
106 -- ----------------------------------------------------------------------
```

Fig 4: Calc.vhdl

C:\Users\Shane\School Classes\CPEG 324 (System Design II)\Labs\Lab 3\Calc.vhdl - Sublime Text (UNREGISTERED)

File Edit Selection Find View Goto Tools Project Preferences Help

add-sub.vhdl | regFile.vhdl | clock-filter.vhdl | Calc.vhdl

```
1   --Created By Shane Cincotta 5/7/19
2
3   library ieee;
4   use ieee.std_logic_1164.all;
5   use ieee.numeric_std.all;
6
7   -- ---------------------------------------------------------------
8   --Calc entity
9
10  entity Calc is
11    port(
12      I : in std_logic_vector(7 downto 0); --instruction input
13      clk : in std_logic
14    );
15  end entity Calc;
16
17  -- ---------------------------------------------------------------
18  --Calc architecture
19
20  architecture structural of Calc is
21    component add-sub is
22        port(
23            in_1, in_2 : in std_logic_vector(7 downto 0); --in_1 and in_2 are the operands
24            mode_sel : in std_logic; --controls if Were doing addition or subtraction, 0 if addition, 1 if subtraction
25            total : out std_logic_vector(7 downto 0)
26          );
27    end component add-sub;
28
29    component regFile is
30      port(
31        RD : in std_logic_vector(1 downto 0);
32        RS : in std_logic_vector(1 downto 0);
33        RT : in std_logic_vector(1 downto 0);
34        Write_Data : in std_logic_vector(7 downto 0);
35        CLK : in std_logic;
36        Write_Enable : in std_logic;
37        RS_DATA : out std_logic_vector(7 downto 0);
38        RT_DATA : out std_logic_vector(7 downto 0)
39      );
40    end component regFile;
41
42    component clk_filter is
43      port(
44        clk_in : in std_logic;
45        clk_out : out std_logic;
46        S: in std_logic;
47        skip: in std_logic
48      );
49    end component clk_filter;
50
51
52  signal RS, RT, RD : std_logic_vector(1 downto 0); --2 bits to represent our 4 registers
53  signal filtered_clk, Write_Enable, display, WD_sel, skip, skip_compare : std_logic;
54  signal Write_Data, RS_DATA, RT_DATA, extended_immediate, ALU_OUTPUT: std_logic_vector(7 downto 0);
55
```

C:\Users\Shane\School Classes\CPEG 324 (System Design II)\Labs\Lab 3\Calc.vhdl - Sublime Text (UNREGISTERED)

File Edit Selection Find View Goto Tools Project Preferences Help

add-sub.vhdl | regFile.vhdl | clock-filter.vhdl | Calc.vhdl

```
55
56
57  begin
58    regFile_0 : regFile port map(RS, RT, RD, Write_Data, filtered_clk, Write_Enable, RS_DATA, RT_DATA);
59    ALU: add-sub port map(RS_DATA, RT_DATA, I(7), ALU_OUTPUT);
60    clk_filter_0 : clk_filter port map(clk, filtered_clk, I(4), skip);
61
62    --Making control signals (see data path)
63    RD <= I(5 downto 4);
64    RT <= I(1 downto 0);
65
66
67    display <= not (I(7) or I(6) or I(5)); --Used for printing contents of register
68
69    with display select RS <=
70      I(3 downto 2) when '0',
71      I(4 downto 3) when others;
72
73    --Make immediate sign extended
74    extended_immediate(3 downto 0) <= I(3 downto 0);
75    with I(3) select extended_immediate(7 downto 4) <=
76      "1111" when '1',
77      "0000" when others;
78
79    WD_sel <= not(I(7) and I(6));
80    with WD_sel select Write_Data <=
81      extended_immediate when '0', --sign extended value written to RD
82      ALU_OUTPUT when others; --ALU result written to RD
83
84    --Write Data is written to RD if instruction is add, sub or LI
85    Write_Enable <= I(7) or I(6);
86
87    --checks opcode and skip bit
88    skip <= (not I(7)) and (not I(6)) and I(5) and skip_compare;
89
90    --Compare registers to determine if Write_Enable should skip or not
91    skip_compare <= (RS_DATA(7) xnor RT_DATA(7)) and
92                    (RS_DATA(6) xnor RT_DATA(6)) and
93                    (RS_DATA(5) xnor RT_DATA(5)) and
94                    (RS_DATA(4) xnor RT_DATA(4)) and
95                    (RS_DATA(3) xnor RT_DATA(3)) and
96                    (RS_DATA(2) xnor RT_DATA(2)) and
97                    (RS_DATA(1) xnor RT_DATA(1)) and
98                    (RS_DATA(0) xnor RT_DATA(0)
99                    );
100
101
102   --Prints value to display
103   process(filtered_clk, display) is
104     variable value : integer;
105     begin
106       if((filtered_clk'event and filtered_clk = '1') and (display = '1')) then
107         value := to_integer(signed(RS_DATA));
108         report integer'image(value) severity note; --print the value
109       end if;
110   end process;
111 end architecture structural;
112 -- ---------------------------------------------------------------------
113
114
```