Course: UESTC3020

Names: Jinming Ren, Yuhao Liu (Both from class 4)

**GUIDs:** ——-R, ——-L **Date:** 12 January 2024

## **DCD Coursework Report**

## 1 PART II: Validation of Combinational Design

The circuit structure of the 2-bit and 4-bit adder are shown in Figure 1 and Figure 2 in Appendix A respectively.

### 1.1 Task 7-8: VHDL Behavioural Description and Simulation of the 2-bit Adder

Listing 1 in Appendix E contains the behavioural description of the 2-bit adder. We simulate the 2-bit adder with 4 test cases: 00 + 00, 01 + 01, 10 + 10, and 11 + 11. The output waveform is shown in Figure 3.

### 1.2 Task 9-10: VHDL Structural Description and Simulation of the 4-bit Adder

We reuse the design of the 2-bit adder in Section 1.1 to build the 4-bit adder. The structural description of it is shown in Listing 2 in Appendix E. We simulate the 4-bit adder with 5 test cases: 0000 + 0000, 0001 + 0001, 1111 + 0001, 1010 + 0101, and 1111 + 1111. The output waveform is shown in Figure 4, which shows that the result is 0000 (c=0), 0010 (c=0), 0000 (c=1), 1111 (c=0), and 1110 (c=1) respectively. Obviously, they are correct.

### 2 PART III: Generalised Adder

### 2.1 Task 11: Block Diagram of Our Proposed Generalised Adder

The block diagram of our proposed generalised adder is shown in Figure 5 in Appendix B. Here is an explanation of each block in the diagram and how they are connected each other:

- **Input:** For the user to enter the number of data N and all operands.
- Data Storage Part: Store N 4-bit data entered by the user and transmit them to the adder one by one with 4 bits transmitted in parallel.
- 4-bit Adder/Counter: Considering the various possible forms of the final output and the rules for adding N
   4-bit data, we decompose the addition between them into that for the first four bits and the last four bits. For the addition of the first four bits, we use the memory part to store the result of the previous operation and add it to the new input, and each resulting carry is entered into the adder for the last four bits for accumulation.
- Connector: Reassemble the 8 bits of the operation result.
- FSM (Control Unit): Provide all control signals and the clock signal. (Ensure that the circuit can run when N data have been inputted and stored and the final outcome can be presented when and only when N data have participated in operation.)
- Output: Display the final result.

### 2.2 Task 12: Realization of Each Block in Section 2.1

The circuit realization of the proposed generalised adder is shown in Figure 6 in Appendix B.

<sup>&</sup>lt;sup>1</sup>All testbench files together with the entire project manual can be seen at test



### 2.3 Task 13-14: VHDL Behavioural Description and Simulation of Each Block in Section 2.1

All the behavioural descriptions of the proposed generalised adder is shown in Appendix E.2. All The simulation<sup>2</sup> methods and results are presented in sections below<sup>3</sup>.

### 2.3.1 Counter (Higher 4 bits adder)

According to the specifications in Section 2.1, this counter should be able to add one whenever the result of the lower 4 bits have a carry. The simulation waveform in Figure 7 in Appendix C is explained as follows:

- 1. 0 ns: The counter is initialized to 0.
- 2. 0-30 ns: The counter keeps its value because the c signal is not asserted.
- 3. 30-80 ns: c signal is asserted at 30 ns, the counter increases by one at 40 ns and 60 ns where the clock is a rising edge.
- 4. 80-110 ns: Same as step 2.
- 5. 110-140 ns: Same as step 3.

#### 2.3.2 Connector

According to the specifications in Section 2.1, this connector should be able to reassemble the lower and higher 4 bits of the operation result. The simulation waveform in Figure 8 in Appendix C is explained as follows:

- 1. 0 ns: rh and rl is initialized to 0x0A and 0x05 respectively.
- 2. 0-30 ns: The connector keeps its value because the rst signal<sup>4</sup> is not asserted.
- 3. 30-60 ns: rst signal is asserted at 30 ns, so the connector immediately shows the assembled result 0xA5.
- 4. 60-120 ns: The connector losts its value at 60 ns because the rst signal is unasserted.

### 2.3.3 Data Storage Part (EEPROM)

According to the specifications in Section 2.1, the user first store all the operand data here before running the calculations. The simulation waveform in Figure 9 in Appendix C is explained as follows:

- 1. 0-10 ns: The stored data in address 0x00 contains a random value (which happens to be 0x00). Write enable (we) signal is unasserted.
- 2. 10-40 ns: we signal is asserted at 10 ns, the data 0x0A is written into address 0x00.
- 3. 40-120 ns: we signal is unasserted at 40 ns, therefore, as long as the address line is not changed, the data in address 0x00 is always 0x0A.
- 4. 120-130 ns: The address line is changed to 0x01, the output dout is showing the number randomly initialized in that address.
- 5. 130-150 ns: we signal is asserted at 130 ns. The data 0x0C is written into address 0x01 at 130 ns because the clock is rising edge.
- 6. 150-190 ns: Same as step 3.

<sup>&</sup>lt;sup>2</sup>All testbench files in VHDL can be seen at test

<sup>&</sup>lt;sup>3</sup>Only part of the FSM (Control Unit) block (isMax) is simulated here, other parts are simulated in Section 2.4.

<sup>&</sup>lt;sup>4</sup>Though this rst signal is called "reset", in the case of this connector, it makes the final calculation visible at the end of calculation. For other modules, this rst clear the memory inside.

### 2.3.4 isMax module (Part of the FSM)

This module should be able to determine whether the calculation is finished (i.e., if the addr signal is exactly equal to the predefined n value). The simulation waveform in Figure 10 in Appendix C is explained as follows:

- 1. 0-10, 30-40 ns: addr is exactly equal to n. Therefore, the rst signal is asserted.
- 2. 10-30, 40-50 ns: addr is not equal to n. Therefore, the rst signal is unasserted.

## 2.4 Task 15-16: VHDL Structural Description and Simulation of Our Complete Generalised Adder

The structural description of the complete system is shown in Listing 12 in Appendix E.3. According to the requirements of the task, we simulate the generalised adder with the following case<sup>5</sup>:

- N = 4 + 1 = 5.
- Input 0-3: 0011, 0101, 0111, 1001. (In decimal, they are 3, 5, 7, 9)

### 2.5 Task 17: Simulation Results of Our Generalised Adder

The output waveform is shown in Figure 11 in Appendix C. Here is the explanation of the waveform (which works perfectly as expected<sup>6</sup>):

- 1. O ns: Begin simulation by prompt the user to enter the number of data N (in this case, there are 4 data, so N=4+1=5).
- 2. 0-25 ns: Assert write and forcerst signal and wait for a clock period to clear the random memory in EEPROM.
- 3. 25-65 ns: Unassert write but keep forcerst asserted to clear the random bits inside all counters and buffers.
- 4. 65-145 ns: Assert write signal and unassert forcerst to write the four data together with their corresponding addresses into the EEPROM. (For example, in 65-85 ns, the first data 0011 (3 in decimal) is written into the address 0.)
- 5. 145-230 ns: Unassert write signal and wait for calculation (the FSM will fetch all the data from the EEPROM and calculate their sum, since there are 4 operands, 4 clock cycles should be waited).
- 6. 230-250 ns: The final result 24 in decimal (which is the correct sum of 3+5+7+9) is displayed on the output.
- 7. >250 ns: If the user do not assert forcerst signal, the circuit will go back to step 4 and form a loop.

## 3 Conclusion

As can be seen from the circuit structure and simulation results, the universal adder we designed run smoothly and meet all the design requirements, including data input one by one, four bits of each data input in parallel, users can design the number of data involved in the calculation, the final result is output after all the data is involved in the calculation and so on.

For future work, we can further optimize and simplify the construction and connection of various components in the circuit to reduce the integrated complexity. In addition, if the cost of various logic gates and various integrated components is known, we can further reduce the design cost of the entire circuit through calculation. Finally, for better utilization in ANN, we can reuse the circuit and consider designing a more advanced adder that can perform more complex operations.

 $<sup>^{5}</sup>$ The last two numbers in decimal are 7 and 9, which are the sum of last two digits of our GUIDs.

 $<sup>^6</sup>$ For dynamic and visualized simulation, check out the Digital source files at  ${ t test}$ .

## A Figures for PART II

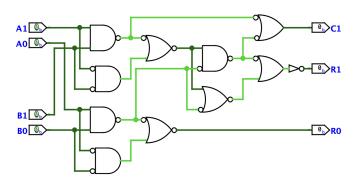


Figure 1: Circuit Structure of the 2-bit Adder

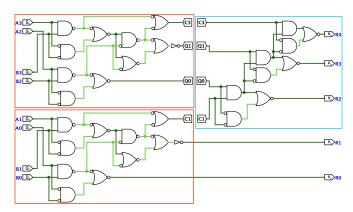


Figure 2: Circuit Structure of the 4-bit Adder



Figure 3: 2-bit adder simulation waveform



Figure 4: 4-bit adder simulation waveform

# **B** Circuit Figures for PART III

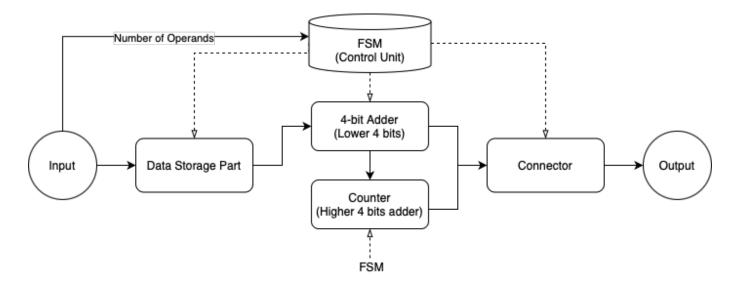


Figure 5: Block diagram of the proposed generalised adder<sup>7</sup>

<sup>&</sup>lt;sup>7</sup>Filled-in arrow denotes data flow, not-filled-in arrow denotes control signals and clock signal.

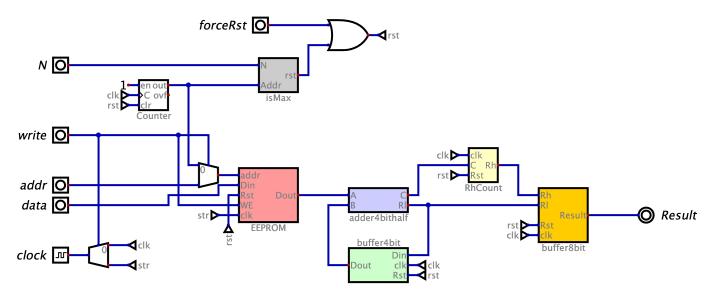


Figure 6: Circuit realization of the proposed generalised adder

## **Simulation Waveforms for PART III**

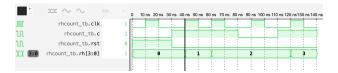


Figure 7: Counter (Higher 4 bits adder) simulation waveform

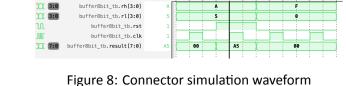




Figure 9: Data Storage Part (EEPROM) simulation waveform



Figure 10: isMax module (Part of the FSM) simulation waveform



Figure 11: Simulation waveform of the proposed generalised adder under the condition specified in Section 2.4

# **D** Subcircuit Figures for PART III

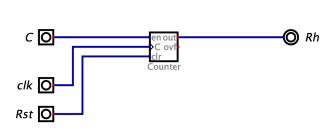


Figure 12: Counter (Higher 4 bits adder) subcircuit

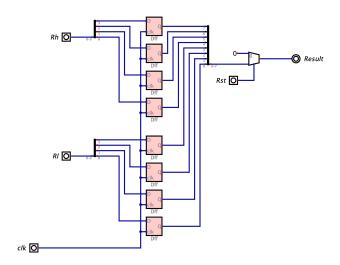


Figure 13: Connector (8-bit Buffer) subcircuit

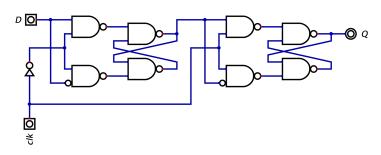


Figure 14: Master-slave D Flip-Flop subcircuit

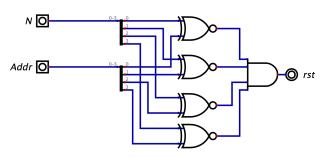


Figure 15: isMax module (Part of the FSM) subcircuit

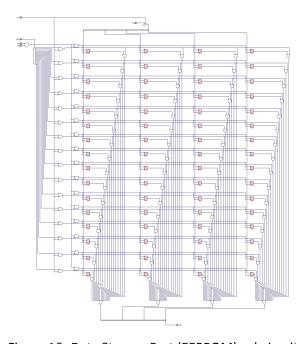


Figure 16: Data Storage Part (EEPROM) subcircuit

## **E VHDL Codes**

### E.1 VHDL Codes for PART II

Listing 1: Behavioural description of 2-bit adder

```
adder2bit.vhdl
1
   LIBRARY ieee:
2
   USE ieee.std_logic_1164.all;
3
4
5
   entity adder2bit is
       port (
6
            Ah: in std_logic;
7
           Al: in std_logic;
8
            Bh: in std_logic;
9
           Bl: in std_logic;
10
            C: out std_logic;
11
            Q1: out std_logic;
12
           Q0: out std_logic);
13
   end adder2bit;
14
15
   architecture Behavioral of adder2bit is
16
       signal sel: std_logic_vector(3 downto 0); Combine inputs into a single
17
      signal
   begin
18
          Combine inputs into a 4bit signal
19
       sel <= Ah & Al & Bh & Bl;
20
21
          Enumerate all possible input combinations and assign Q0
22
       Q0 <= '1' when sel = "0001" or
23
                       sel = "0011" or
24
                       sel = "0100" or
25
                        sel = "0110" or
26
                       sel = "1001" or
27
                       sel = "1011" or
28
                       sel = "1100" or
29
                       sel = "1110" else
30
              '0';
31
32
          Enumerate all possible input combinations and assign Q1
33
       Q1 <= '1' when sel = "0010" or
34
                       sel = "0011" or
35
                       sel = "0101" or
36
                        sel = "0110" or
37
                       sel = "1000" or
38
                        sel = "1001" or
39
40
                       sel = "1100" or
                       sel = "1111" else
41
              '0';
42
43
          Enumerate all possible input combinations and assign C
44
       C <= '1' \text{ when sel} = "0111" \text{ or}
45
                      sel = "1010" or
46
                      sel = "1011" or
47
                      sel = "1101" or
48
```

```
sel = "1110" or

sel = "1111" else

'0';

end Behavioral;
```

### Listing 2: Behavioural description of 4-bit adder

```
adder4bit.vhdl
1
   LIBRARY ieee;
2
   USE ieee.std_logic_1164.all;
   USE ieee.numeric_std.all;
4
5
   entity adder4bit is
6
        port (
7
8
            A: in std_logic_vector(3 downto 0);
            B: in std_logic_vector(3 downto 0);
9
            C: out std_logic; carry (R4)
10
11
            R1: out std_logic_vector(3 downto 0) Lower four bits of the result
        );
12
   end adder4bit:
13
14
15
   architecture Dataflow of adder4bit is
        signal s0: std_logic;
16
        signal s1: std_logic;
17
        signal s2: std_logic;
18
        signal s3: std_logic;
19
        signal s4: std_logic;
20
        signal s5: std_logic;
21
        signal s6: std_logic;
22
        signal s7: std_logic;
23
        signal s8: std_logic;
24
25
        signal s9: std_logic;
        signal s10: std_logic;
26
        signal s11: std_logic;
27
        signal s12: std_logic;
28
        signal s13: std_logic;
29
        signal s14: std_logic;
30
        signal s15: std_logic;
31
   begin
32
33
        s11 <= A(0);
        s10 \ll A(1);
34
        s7 <= A(2);
35
        s6 \ll A(3);
36
        s13 \ll B(0):
37
        s12 \ll B(1);
38
        s9 \ll B(2);
39
        s8 \ll B(3);
40
        gate0: entity work.adder2bit
41
42
        port map (
            Ah => s6
43
44
            Al => s7.
            Bh \Rightarrow s8.
45
            B1 \Rightarrow s9,
46
47
            C \Rightarrow s5.
            Q1 \Rightarrow s3,
48
```

```
Q0 => s0);
49
        gate1: entity work.adder2bit
50
        port map (
51
52
             Ah => s10,
53
             Al => s11,
             Bh \Rightarrow s12
54
             B1 => s13,
55
             C \Rightarrow s1
56
             01 \Rightarrow s14.
57
58
             Q0 => s15);
59
         s2 <= (s0 AND s1);
        s4 \ll (s3 AND s2);
60
        C \leftarrow NOT ((s5 AND s4) OR (NOT s5 AND NOT s4));
61
        R1(0) \le s15;
62
        Rl(1) <= s14;
63
        R1(2) \leftarrow NOT (s2 OR (NOT s0 AND NOT s1));
64
        R1(3) \leftarrow NOT (s4 OR (NOT s3 AND NOT s2));
65
   end Dataflow;
66
```

### E.2 VHDL Behavioural Description for Each Block in PART III

### E.2.1 Counter

Listing 3: DIG\_Counter as the subcircuit for RhCount and main

```
DIG_Counter.vhdl
1
   LIBRARY ieee;
2
  USE ieee.std_logic_1164.all;
3
  USE ieee.std_logic_unsigned.all;
   USE ieee.numeric_std.all;
5
6
7
   entity DIG_Counter is
8
       generic ( Bits: integer );
       port (
9
            p_out: out std_logic_vector((Bits1) downto 0);
10
            ovf: out std_logic;
11
12
            C: in std_logic;
            en: in std_logic;
13
            clr: in std_logic );
14
   end DIG_Counter;
15
16
17
   architecture Behavioral of DIG_Counter is
        signal count : std_logic_vector((Bits1) downto 0) := (others => '0');
18
19
   begin
            process (C, clr, en)
20
            begin
21
22
                if rising_edge(C) then
                    if clr='1' then
23
                        count <= (others => '0');
24
                    elsif en='1' then
25
                        count <= count + 1;</pre>
26
                    end if:
27
                end if:
28
            end process;
29
```

```
p_out <= count;

ovf <= en when count = ((2**Bits)1) else '0';

end Behavioral;</pre>
```

Listing 4: RhCount is just an instantiation of DIG\_Counter

```
RhCount.vhdl
1
2
   LIBRARY ieee;
3
   USE ieee.std_logic_1164.all;
4
   USE ieee.numeric_std.all;
6
   entity RhCount is
7
8
       port (
            clk: in std_logic;
9
            C: in std_logic;
10
            Rst: in std_logic;
11
            Rh: out std_logic_vector(3 downto 0) Higher four bits of the result
12
13
   end RhCount;
14
15
   architecture Behavioral of RhCount is
16
17
   begin
18
       gate0: entity work.DIG_Counter
19
            generic map (
                Bits \Rightarrow 4)
20
            port map (
21
22
                 en => C,
                 C \Rightarrow clk
23
                clr => Rst,
24
25
                 p_out => Rh);
   end Behavioral;
26
```

### E.2.2 Decoder

Listing 5: DECODER\_4 as the subcircuit for EEPROM and main

```
LIBRARY ieee;
   USE ieee.std_logic_1164.all;
2
3
4
   entity DECODER_4 is
5
       port (
            out_0: out std_logic;
6
            out_1: out std_logic;
7
8
            out_2: out std_logic;
            out_3: out std_logic;
9
            out_4: out std_logic;
10
            out_5: out std_logic;
11
            out_6: out std_logic;
12
            out_7: out std_logic;
13
            out_8: out std_logic;
14
15
            out_9: out std_logic;
            out_10: out std_logic;
16
```

```
out_11: out std_logic;
17
           out_12: out std_logic;
18
           out_13: out std_logic;
19
           out_14: out std_logic;
20
21
           out_15: out std_logic;
           sel: in std_logic_vector (3 downto 0) );
22
   end DECODER_4;
23
24
   architecture Behavioral of DECODER_4 is
25
   begin
26
       out_0 <= '1' when sel = "0000" else '0';
27
       out_1 <= '1' when sel = "0001" else '0';
28
       out_2 <= '1' when sel = "0010" else '0';
29
       out 3 <= '1' when sel = "0011" else '0':
30
       out_4 <= '1' when sel = "0100" else '0';
31
       out_5 <= '1' when sel = "0101" else '0';
32
       out_6 <= '1' when sel = "0110" else '0';
33
       out_7 <= '1' when sel = "0111" else '0';
34
       out 8 <= '1' when sel = "1000" else '0':
35
       out_9 <= '1' when sel = "1001" else '0';
36
37
       out_10 <= '1' when sel = "1010" else '0'
       out_11 <= '1' when sel = "1011" else '0';
38
       out_12 <= '1' when sel = "1100" else '0'
39
       out_13 <= '1' when sel = "1101" else '0';
40
       out_14 <= '1' when sel = "1110" else '0';
41
       out_15 <= '1' when sel = "1111" else '0';
42
43
   end Behavioral;
```

### E.2.3 Multiplexer

Listing 6: MUX\_GATE\_BUS\_1 as the subcircuit for EEPROM, buffer8bit, buffer4bit and main

```
MUX_GATE_BUS_1.vhdl
1
   LIBRARY ieee;
2
   USE ieee.std_logic_1164.all;
3
4
   entity MUX_GATE_BUS_1 is
5
6
       generic ( Bits : integer );
       port (
7
           p_out: out std_logic_vector ((Bits1) downto 0);
8
           sel: in std_logic;
9
10
            in_0: in std_logic_vector ((Bits1) downto 0);
11
           in_1: in std_logic_vector ((Bits1) downto 0) );
12
13
   end MUX_GATE_BUS_1;
14
   architecture Behavioral of MUX_GATE_BUS_1 is
15
16
   begin
17
       with sel select
18
           p_out <=
                in_0 when '0',
19
20
                in_1 when '1'
                (others => '0') when others;
21
22
   end Behavioral;
```

### E.2.4 D Flip-Flop

Listing 7: Dff as the subcircuit for EEPROM, buffer8bit, buffer4bit and main

```
Dff.vhdl
1
2
   LIBRARY ieee;
   USE ieee.std_logic_1164.all;
3
4
5
   entity Dff is
       port (
6
7
           D : in std_logic;
                                   Data input
            clk : in std_logic;
                                   Clock input
8
              : out std_logic
                                   Output
9
       );
10
   end Dff:
11
12
   architecture Behavioral of Dff is
13
       signal Q_internal : std_logic := '0';
                                                  Internal signal for flipflop state
14
15
   begin
       process(clk)
16
17
       begin
            if rising_edge(clk) then
18
19
                Q_internal <= D; Update internal state on clock's rising edge</pre>
20
            end if:
       end process;
21
22
       Q <= Q_internal;</pre>
                           Assign internal state to output
23
   end Behavioral:
24
```

### E.2.5 EEPROM (Data Storage Part)

Listing 8: EEPROM as the subcircuit for main

```
1
      EEPROM.vhdl
      This passed the testbench with a minor error, which is when Rst is set (and
2
      everything else remain the same, the output changes randomly, but it does not
      affect the functionality of the EEPROM.)
   LIBRARY ieee;
3
   USE ieee.std_logic_1164.all;
  USE ieee.numeric_std.all;
6
7
   entity EEPROM is
8
       port (
9
           addr: in std_logic_vector(3 downto 0);
           Din: in std_logic_vector(3 downto 0);
10
           Rst: in std_logic;
11
12
           WE: in std_logic;
           clk: in std_logic;
13
           Dout: out std_logic_vector(3 downto 0));
14
15
   end EEPROM;
16
   architecture Structural of EEPROM is
17
       signal s0: std_logic;
18
19
       signal s1: std_logic;
```

```
signal s2: std_logic;
20
       signal s3: std_logic_vector(3 downto 0);
21
22
       signal s4: std_logic;
       signal s5: std_logic;
23
       signal s6: std_logic;
24
       signal s7: std_logic;
25
26
       signal s8: std_logic;
       signal s9: std_logic;
27
       signal s10: std_logic;
28
       signal s11: std_logic;
29
30
       signal s12: std_logic;
       signal s13: std_logic;
31
       signal s14: std_logic;
32
33
       signal s15: std_logic;
       signal s16: std_logic;
34
35
       signal s17: std_logic;
       signal s18: std_logic;
36
       signal s19: std_logic;
37
38
       signal s20: std_logic;
       signal s21: std_logic;
39
40
       signal s22: std_logic;
41
       signal s23: std_logic;
       signal s24: std_logic;
42
43
       signal s25: std_logic;
       signal s26: std_logic;
44
45
       signal s27: std_logic;
       signal s28: std_logic;
46
       signal s29: std_logic;
47
       signal s30: std_logic;
48
       signal s31: std_logic;
49
50
       signal s32: std_logic;
51
       signal s33: std_logic;
       signal s34: std_logic;
52
53
       signal s35: std_logic;
       signal s36: std_logic;
54
       signal s37: std_logic;
55
56
       signal s38: std_logic;
57
       signal s39: std_logic;
58
       signal s40: std_logic;
59
       signal s41: std_logic;
       signal s42: std_logic;
60
       signal s43: std_logic;
61
       signal s44: std_logic;
62
63
       signal s45: std_logic;
       signal s46: std_logic;
64
       signal s47: std_logic;
65
66
       signal s48: std_logic;
67
       signal s49: std_logic;
       signal s50: std_logic;
68
69
       signal s51: std_logic;
       signal s52: std_logic;
70
       signal s53: std_logic;
71
72
       signal s54: std_logic;
       signal s55: std_logic;
73
       signal s56: std_logic;
74
```

```
signal s57: std_logic;
75
        signal s58: std_logic;
76
77
        signal s59: std_logic;
        signal s60: std_logic;
78
        signal s61: std_logic;
79
        signal s62: std_logic;
80
81
        signal s63: std_logic;
        signal s64: std_logic;
82
83
        signal s65: std_logic;
        signal s66: std_logic;
84
85
        signal s67: std_logic;
        signal s68: std_logic;
86
        signal s69: std_logic;
87
88
        signal s70: std_logic;
        signal s71: std_logic;
89
        signal s72: std_logic;
90
        signal s73: std_logic;
91
        signal s74: std_logic;
92
93
        signal s75: std logic:
        signal s76: std_logic;
94
95
        signal s77: std_logic;
96
        signal s78: std_logic;
        signal s79: std_logic;
97
98
        signal s80: std_logic;
99
        signal s81: std_logic;
        signal s82: std_logic;
100
101
        signal s83: std_logic;
102
        signal s84: std_logic;
        signal s85: std_logic;
103
        signal s86: std_logic;
104
105
        signal s87: std_logic;
        signal s88: std_logic;
106
        signal s89: std_logic;
107
        signal s90: std_logic;
108
        signal s91: std_logic;
109
        signal s92: std_logic;
110
        signal s93: std_logic;
111
112
        signal s94: std_logic;
113
        signal s95: std_logic;
114
        signal s96: std_logic;
        signal s97: std_logic;
115
        signal s98: std_logic;
116
        signal s99: std_logic;
117
        signal s100: std_logic;
118
        signal s101: std_logic;
119
        signal s102: std_logic;
120
121
        signal s103: std_logic;
122
        signal s104: std_logic;
        signal s105: std_logic;
123
124
        signal s106: std_logic;
        signal s107: std_logic;
125
        signal s108: std_logic;
126
        signal s109: std_logic;
127
        signal s110: std_logic;
128
        signal s111: std_logic;
129
```

```
signal s112: std_logic;
130
         signal s113: std_logic;
131
         signal s114: std_logic;
132
         signal s115: std_logic;
133
134
         signal s116: std_logic;
         signal s117: std_logic;
135
136
    begin
         gate0: entity work.DECODER_4
137
              port map (
138
139
                   sel => addr.
140
                   out_0 => s101,
                   out_1 => s102,
141
                   out_2 => s103,
142
143
                   out_3 => s104,
                   out_4 \Rightarrow s105,
144
                   out_5 => s106,
145
                   out_6 \Rightarrow s107,
146
147
                   out_7 => s108,
148
                   out 8 =  109.
                   out_9 => s110,
149
                   out_10 => s111,
150
151
                   out_{11} => s112,
                   out_12 => s113,
152
                   out_13 => s114,
153
154
                   out_14 => s115,
                   out_15 => s116);
155
156
         s117 \ll (clk AND WE);
         gate1: entity work.MUX_GATE_BUS_1
157
              generic map (
158
159
                   Bits \Rightarrow 4)
160
              port map (
161
                   sel => Rst,
162
                   in_0 => Din_1
                   in_1 = "0000",
163
                   p_out => s3);
164
165
         s52 \le (s101 \ OR \ Rst);
         s49 \le (s102 \ OR \ Rst);
166
         s46 \le (s103 \ OR \ Rst);
167
168
         s43 \le (s104 \ OR \ Rst);
         s40 \ll (s105 \ OR \ Rst);
169
         s37 <= (s106 \ OR \ Rst);
170
         s34 \le (s107 \ OR \ Rst);
171
         s31 \le (s108 \ OR \ Rst);
172
173
         s28 \le (s109 \ OR \ Rst);
         s25 \ll (s110 \ OR \ Rst);
174
         s22 \le (s111 \ OR \ Rst);
175
         s19 \ll (s112 \ OR \ Rst);
176
177
         s16 <= (s113 OR Rst);
         s13 \ll (s114 \ OR \ Rst);
178
         s10 <= (s115 OR Rst);
179
         s7 <= (s116 OR Rst);
180
         s4 \ll s3(0);
181
         s5 \le s3(1);
182
         s6 \le s3(2);
183
         s0 \ll s3(3);
184
```

```
s50 \le (s117 \text{ AND } s52);
185
           s47 \ll (s117 \text{ AND } s49);
186
           s44 \le (s117 \text{ AND } s46);
187
          s41 \le (s117 \text{ AND } s43);
188
189
           s38 \le (s117 \text{ AND } s40);
           s35 \le (s117 \text{ AND } s37);
190
          s32 \le (s117 \text{ AND } s34);
191
192
          s29 \le (s117 \text{ AND } s31);
          s26 \le (s117 \text{ AND } s28);
193
194
           s23 \le (s117 \text{ AND } s25);
           s20 \le (s117 \text{ AND } s22);
195
          s17 \ll (s117 \text{ AND } s19);
196
          s14 \ll (s117 \text{ AND } s16);
197
          s11 \leftarrow (s117 \text{ AND } s13);
198
           s8 \le (s117 \text{ AND } s10);
199
200
           s1 \ll (s117 \text{ AND } s7);
          gate2: entity work.Dff
201
                port map (
202
203
                     D \Rightarrow s0,
204
                     clk => s1,
205
                     Q \Rightarrow s2;
206
          gate3: entity work.Dff
207
                port map (
208
                     D \Rightarrow s0,
209
                     clk => s8,
                     Q => s9);
210
211
          gate4: entity work.Dff
212
                port map (
213
                     D \Rightarrow s0,
214
                     clk => s11,
215
                     Q => s12);
216
          gate5: entity work.Dff
217
                port map (
218
                     D \Rightarrow s0,
219
                     clk \Rightarrow s14,
220
                     Q => s15);
          gate6: entity work.Dff
221
222
                port map (
223
                     D \Rightarrow s0,
                     clk \Rightarrow s17,
224
225
                     Q => s18);
          gate7: entity work.Dff
226
227
                port map (
228
                     D \Rightarrow s0,
229
                     clk => s20,
230
                     Q => s21);
231
          gate8: entity work.Dff
                port map (
232
233
                     D \Rightarrow s0,
234
                     clk => s23,
235
                     Q => s24);
236
          gate9: entity work.Dff
237
                port map (
238
                     D \Rightarrow s0,
239
                     clk => s26,
```

```
240
                   Q => s27);
         gate10: entity work.Dff
241
              port map (
242
                   D \Rightarrow s0,
243
244
                   clk => s29,
                   Q => s30);
245
         gate11: entity work.Dff
246
              port map (
247
248
                   D \Rightarrow s0,
                   clk => s32,
249
250
                   Q => s33);
251
         gate12: entity work.Dff
              port map (
252
253
                   D \Rightarrow s0,
254
                   clk => s35,
255
                   Q => s36);
         gate13: entity work.Dff
256
257
              port map (
258
                   D \Rightarrow s0,
                   clk => s38,
259
                   Q => s39);
260
261
         gate14: entity work.Dff
262
              port map (
263
                   D \Rightarrow s0,
264
                   clk \Rightarrow s41,
265
                   Q => s42);
         gate15: entity work.Dff
266
              port map (
267
                   D \Rightarrow s0,
268
269
                   clk => s44,
                   Q => s45);
270
         gate16: entity work.Dff
271
272
              port map (
                   D \Rightarrow s0,
273
                   clk => s47,
274
275
                   Q => s48);
         gate17: entity work.Dff
276
277
              port map (
278
                   D \Rightarrow s0,
279
                   clk => s50,
280
                   Q => s51);
          gate18: entity work.Dff
281
              port map (
282
283
                   D \Rightarrow s6,
284
                   clk \Rightarrow s1,
285
                   Q => s53);
          gate19: entity work.Dff
286
287
              port map (
288
                   D \Rightarrow s6,
                   clk => s8,
289
                   Q => s54);
290
291
         gate20: entity work.Dff
              port map (
292
293
                   D \Rightarrow s6,
294
                   clk \Rightarrow s11,
```

```
295
                   Q => s55);
         gate21: entity work.Dff
296
297
              port map (
                   D \Rightarrow s6,
298
299
                   clk => s14,
                   Q => s56);
300
         gate22: entity work.Dff
301
302
              port map (
                   D \Rightarrow s6,
303
                   clk => s17,
304
305
                   Q => s57);
         gate23: entity work.Dff
306
307
              port map (
                   D \Rightarrow s6,
308
                   clk => s20,
309
310
                   Q => s58);
         gate24: entity work.Dff
311
312
              port map (
313
                   D \Rightarrow s6
                   clk => s23,
314
                   Q => s59);
315
316
         gate25: entity work.Dff
317
              port map (
                   D \Rightarrow s6,
318
319
                   clk => s26,
320
                   Q => s60);
         gate26: entity work.Dff
321
              port map (
322
323
                   D \Rightarrow s6
324
                   clk => s29,
                   Q => s61);
325
         gate27: entity work.Dff
326
327
              port map (
                   D \Rightarrow s6,
328
                   clk => s32,
329
330
                   Q => s62);
         gate28: entity work.Dff
331
332
              port map (
333
                   D \Rightarrow s6,
                   clk => s35,
334
335
                   Q => s63);
         gate29: entity work.Dff
336
              port map (
337
338
                   D \Rightarrow s6,
339
                   clk => s38,
340
                   Q => s64);
         gate30: entity work.Dff
341
342
              port map (
343
                   D \Rightarrow s6,
                   clk => s41,
344
                   Q => s65);
345
         gate31: entity work.Dff
346
              port map (
347
348
                   D \Rightarrow s6,
349
                   clk => s44,
```

```
350
                   Q => s66);
         gate32: entity work.Dff
351
              port map (
352
                   D \Rightarrow s6,
353
354
                   clk => s47,
                   Q => s67);
355
         gate33: entity work.Dff
356
357
              port map (
                   D \Rightarrow s6,
358
                   clk => s50,
359
                   Q => s68);
360
361
         gate34: entity work.Dff
              port map (
362
363
                   D \Rightarrow s5,
364
                   clk => s1,
365
                   Q => s69);
         gate35: entity work.Dff
366
367
              port map (
368
                   D \Rightarrow s5
                   clk => s8,
369
                   Q => s70);
370
371
         gate36: entity work.Dff
372
              port map (
                   D \Rightarrow s5,
373
374
                   clk \Rightarrow s11,
375
                   Q => s71);
         gate37: entity work.Dff
376
              port map (
377
378
                   D \Rightarrow s5
379
                   clk => s14,
                   Q => s72);
380
         gate38: entity work.Dff
381
              port map (
382
                   D \Rightarrow s5,
383
                   clk => s17,
384
385
                   Q => s73);
         gate39: entity work.Dff
386
387
              port map (
388
                   D \Rightarrow s5,
                   clk => s20,
389
390
                   Q => s74);
         gate40: entity work.Dff
391
              port map (
392
393
                   D \Rightarrow s5,
394
                   clk => s23,
395
                   Q => s75);
         gate41: entity work.Dff
396
397
              port map (
398
                   D \Rightarrow s5,
                   clk => s26,
399
                   Q => s76);
400
         gate42: entity work.Dff
401
              port map (
402
403
                   D \Rightarrow s5,
404
                   clk => s29,
```

```
405
                   Q => s77);
         gate43: entity work.Dff
406
              port map (
407
408
                   D \Rightarrow s5,
409
                   clk => s32,
410
                   Q => s78);
         gate44: entity work.Dff
411
412
              port map (
                   D \Rightarrow s5,
413
414
                   clk => s35,
415
                   Q => s79);
         gate45: entity work.Dff
416
417
              port map (
                   D \Rightarrow s5,
418
                   clk => s38,
419
420
                   Q => s80);
421
         gate46: entity work.Dff
422
              port map (
423
                   D \Rightarrow s5
                   clk => s41,
424
425
                   Q => s81);
426
         gate47: entity work.Dff
427
              port map (
428
                   D \Rightarrow s5,
429
                   clk => s44,
430
                   Q => s82);
431
         gate48: entity work.Dff
432
              port map (
433
                   D \Rightarrow s5
434
                   clk => s47,
                   Q => s83);
435
         gate49: entity work.Dff
436
437
              port map (
                   D \Rightarrow s5,
438
                   clk => s50,
439
440
                   Q => s84);
         gate50: entity work.Dff
441
              port map (
442
443
                   D \Rightarrow s4,
                   clk \Rightarrow s1,
444
445
                   Q => s85);
         gate51: entity work.Dff
446
              port map (
447
448
                   D \Rightarrow s4
449
                   clk => s8,
                   Q => s86);
450
         gate52: entity work.Dff
451
452
              port map (
                   D \Rightarrow s4,
453
454
                   clk => s11,
                   Q => s87);
455
         gate53: entity work.Dff
456
              port map (
457
458
                   D \Rightarrow s4,
459
                   clk => s14,
```

```
460
                   Q => s88);
         gate54: entity work.Dff
461
              port map (
462
                   D \Rightarrow s4,
463
464
                   clk => s17,
                   Q => s89);
465
         gate55: entity work.Dff
466
              port map (
467
                   D \Rightarrow s4,
468
469
                   clk => s20,
                   Q => s90);
470
         gate56: entity work.Dff
471
472
              port map (
                   D \Rightarrow s4,
473
                   clk => s23,
474
475
                   Q => s91);
476
         gate57: entity work.Dff
              port map (
477
478
                   D \Rightarrow s4
479
                   clk => s26,
                   Q => s92);
480
481
         gate58: entity work.Dff
482
              port map (
483
                   D \Rightarrow s4
484
                   clk => s29,
485
                   Q => s93);
486
         gate59: entity work.Dff
487
              port map (
488
                   D \Rightarrow s4
489
                   clk => s32,
                   Q => s94);
490
         gate60: entity work.Dff
491
492
              port map (
                   D \Rightarrow s4,
493
                   clk => s35,
494
495
                   Q => s95);
496
         gate61: entity work.Dff
497
              port map (
498
                   D \Rightarrow s4,
                   clk => s38,
499
500
                   Q => s96);
         gate62: entity work.Dff
501
502
              port map (
503
                   D \Rightarrow s4,
504
                   clk \Rightarrow s41,
                   Q => s97);
505
         gate63: entity work.Dff
506
507
              port map (
508
                   D \Rightarrow s4,
509
                   clk => s44,
                   Q => s98);
510
         gate64: entity work.Dff
511
              port map (
512
513
                   D \Rightarrow s4,
                   clk => s47,
514
```

```
515
                 Q => s99);
        gate65: entity work.Dff
516
             port map (
517
                 D \Rightarrow s4,
518
519
                 clk => s50.
                 0 => s100);
520
        Dout(0) <= ((s52 AND s100) OR (s49 AND s99) OR (s46 AND s98) OR (s43 AND s97)
521
        OR (s40 AND s96) OR (s37 AND s95) OR (s34 AND s94) OR (s31 AND s93) OR (s28
       AND s92) OR (s25 AND s91) OR (s22 AND s90) OR (s19 AND s89) OR (s16 AND s88)
       OR (s13 AND s87) OR (s10 AND s86) OR (s7 AND s85));
        Dout(1) <= ((s52 AND s84) OR (s49 AND s83) OR (s46 AND s82) OR (s43 AND s81)
522
       OR (s40 AND s80) OR (s37 AND s79) OR (s34 AND s78) OR (s31 AND s77) OR (s28
       AND s76) OR (s25 AND s75) OR (s22 AND s74) OR (s19 AND s73) OR (s16 AND s72)
       OR (s13 AND s71) OR (s10 AND s70) OR (s7 AND s69));
        Dout(2) <= ((s52 AND s68) OR (s49 AND s67) OR (s46 AND s66) OR (s43 AND s65)
523
       OR (s40 AND s64) OR (s37 AND s63) OR (s34 AND s62) OR (s31 AND s61) OR (s28
       AND s60) OR (s25 AND s59) OR (s22 AND s58) OR (s19 AND s57) OR (s16 AND s56)
       OR (s13 AND s55) OR (s10 AND s54) OR (s7 AND s53));
        Dout(3) \le ((s52 \text{ AND } s51) \text{ OR } (s49 \text{ AND } s48) \text{ OR } (s46 \text{ AND } s45) \text{ OR } (s43 \text{ AND } s42)
524
       OR (s40 AND s39) OR (s37 AND s36) OR (s34 AND s33) OR (s31 AND s30) OR (s28
       AND s27) OR (s25 AND s24) OR (s22 AND s21) OR (s19 AND s18) OR (s16 AND s15)
       OR (s13 AND s12) OR (s10 AND s9) OR (s7 AND s2));
    end Structural;
525
```

### E.2.6 FSM (Control Unit)

Listing 9: isMax as the subcircuit for main, providing control signals

```
isMax.vhdl
1
   LIBRARY ieee;
3
   USE ieee.std_logic_1164.all;
4
   entity isMax is
5
       port (
6
           N: in std_logic_vector(3 downto 0);
7
           Addr: in std_logic_vector(3 downto 0);
8
           rst: out std_logic);
9
   end isMax;
10
11
   architecture Behavioral of isMax is
12
13
   begin
       process(N, Addr)
14
15
       begin
           if N = Addr then
16
                rst <= '1';
                              rst is asserted when N equals Addr
17
           else
18
                              rst is deasserted otherwise
19
                rst <= '0';
20
           end if:
       end process:
21
   end Behavioral;
```

#### E.2.7 Counter (Higher 4 bits adder)

Listing 10: buffer4bit as the subcircuit for main

```
buffer4bit.vhdl
1
2
   LIBRARY ieee;
3
   USE ieee.std_logic_1164.all;
4
   USE ieee.numeric_std.all;
5
6
   entity buffer4bit is
7
       port (
8
            Rst: in std_logic;
9
            clk: in std_logic;
10
            Din: in std_logic_vector(3 downto 0);
11
            Dout: out std_logic_vector(3 downto 0));
12
13
   end buffer4bit:
14
   architecture Structural of buffer4bit is
15
        signal s0: std_logic_vector(3 downto 0);
16
        signal s1: std_logic;
17
        signal s2: std_logic;
18
        signal s3: std_logic;
19
        signal s4: std_logic;
20
21
        signal s5: std_logic;
        signal s6: std_logic;
22
        signal s7: std_logic;
23
24
        signal s8: std_logic;
   begin
25
        gate0: entity work.MUX_GATE_BUS_1
26
            generic map (
27
                 Bits \Rightarrow 4)
28
            port map (
29
                 sel => Rst,
30
31
                 in_0 => Din,
                 in_1 => "0000",
32
                 p_out => s0);
33
        s1 \ll s0(0);
34
        s3 <= s0(1);
35
        s5 \le s0(2);
36
        s7 <= s0(3);
37
        gate1: entity work.Dff
38
            port map (
39
                 D \Rightarrow s1
40
41
                 clk => clk,
                 Q \Rightarrow s2;
42
        gate2: entity work.Dff
43
44
            port map (
                 D \Rightarrow s3,
45
46
                 clk => clk,
                 Q => s4);
47
48
        gate3: entity work.Dff
49
            port map (
                 D \Rightarrow s5.
50
51
                 clk => clk,
52
                 Q \Rightarrow s6;
        gate4: entity work.Dff
53
```

```
port map (
54
                  D \Rightarrow s7
55
                  clk => clk,
56
                  Q \Rightarrow s8;
57
        Dout(0) \le s2;
58
        Dout(1) <= s4;
59
        Dout(2) <= s6;
60
        Dout(3) <= s8;
61
   end Structural;
62
```

### E.2.8 Connector (8-bit Buffer)

Listing 11: buffer8bit as the subcircuit for main

```
buffer8bit.vhdl
1
2
3
   LIBRARY ieee;
   USE ieee.std_logic_1164.all;
4
   USE ieee.numeric std.all:
5
6
7
   entity buffer8bit is
       port (
8
            Rh: in std_logic_vector(3 downto 0);
9
            R1: in std_logic_vector(3 downto 0);
10
            Rst: in std_logic;
11
            clk: in std_logic;
12
            Result: out std_logic_vector(7 downto 0));
13
14
   end buffer8bit;
15
   architecture Structural of buffer8bit is
16
17
        signal s0: std_logic;
        signal s1: std_logic;
18
       signal s2: std_logic;
19
       signal s3: std_logic;
20
       signal s4: std_logic;
21
       signal s5: std_logic;
22
        signal s6: std_logic;
23
24
       signal s7: std_logic;
       signal s8: std_logic;
25
       signal s9: std_logic;
26
        signal s10: std_logic;
27
        signal s11: std_logic;
28
       signal s12: std_logic;
29
       signal s13: std_logic;
30
        signal s14: std_logic;
31
       signal s15: std_logic;
32
        signal s16: std_logic_vector(7 downto 0);
33
   begin
34
       s6 \ll Rh(0);
35
36
        s4 \le Rh(1):
       s2 \ll Rh(2);
37
38
       s0 \ll Rh(3);
39
       s14 \ll Rl(0);
40
       s12 \ll Rl(1);
```

```
s10 \ll R1(2);
41
        s8 \ll R1(3);
42
43
        gate0: entity work.Dff
44
             port map (
                  D \Rightarrow s0,
45
                  clk => clk,
46
47
                  Q \Rightarrow s1;
        gate1: entity work.Dff
48
49
             port map (
                  D \Rightarrow s2,
50
                  clk => clk,
51
52
                  Q => s3);
53
        gate2: entity work.Dff
54
             port map (
55
                  D \Rightarrow s4
                  clk => clk,
56
                  Q => s5);
57
        gate3: entity work.Dff
58
59
             port map (
                  D \Rightarrow s6,
60
61
                  clk => clk,
62
                  Q => s7);
        gate4: entity work.Dff
63
             port map (
64
65
                  D \Rightarrow s8,
                  clk => clk,
66
                  Q => s9);
67
        gate5: entity work.Dff
68
69
             port map (
                  D \Rightarrow s10,
70
71
                  clk => clk,
                  Q => s11);
72
        gate6: entity work.Dff
73
74
             port map (
75
                  D \Rightarrow s12,
76
                  clk => clk,
                  Q => s13);
77
78
        gate7: entity work.Dff
79
             port map (
80
                  D => s14,
81
                  clk => clk,
82
                  Q => s15);
        s16(0) <= s15;
83
84
        s16(1) <= s13;
        s16(2) <= s11;
85
        s16(3) <= s9;
86
        s16(4) <= s7;
87
88
        s16(5) \le s5;
        s16(6) <= s3;
89
        s16(7) <= s1;
90
        gate8: entity work.MUX_GATE_BUS_1
91
92
             generic map (
93
                 Bits \Rightarrow 8)
             port map (
94
                  sel => Rst,
95
```

```
in_0 => "00000000",
in_1 => s16,
p_out => Result);
end Structural;
```

### E.3 VHDL Codes for the Complete Generalised Adder

Listing 12: Structural description of the complete generalised adder reusing the codes in Appendix E.2

```
main.vhdl
1
2
   LIBRARY ieee;
3
   USE ieee.std_logic_1164.all;
4
5
   USE ieee.numeric_std.all;
6
   entity main is
7
       port (
8
            N: in std_logic_vector(3 downto 0);
9
            data: in std_logic_vector(3 downto 0);
10
            write: in std_logic;
11
            addr: in std_logic_vector(3 downto 0);
12
            forceRst: in std_logic;
13
            clock: in std_logic;
14
15
            Result: out std_logic_vector(7 downto 0));
   end main;
16
17
   architecture Structural of main is
18
19
        signal clk: std_logic;
        signal rst: std_logic:
20
       signal s0: std_logic_vector(3 downto 0);
21
22
       signal s1: std_logic_vector(3 downto 0);
       signal s2: std_logic_vector(3 downto 0);
23
       signal s3: std_logic;
24
       signal s4: std_logic_vector(3 downto 0);
25
       signal s5: std_logic_vector(3 downto 0);
26
       signal s6: std_logic;
27
       signal s7: std_logic_vector(3 downto 0);
28
       signal str: std_logic;
29
30
   begin
       gate0: entity work.DEMUX_GATE_1
31
32
            generic map (
                Default => 0)
33
            port map (
34
35
                sel => write,
                p_in => clock,
36
                out_0 => clk,
37
                out_1 => str);
38
       gate1: entity work.DIG_Counter
39
40
            generic map (
                Bits \Rightarrow 4)
41
            port map (
42
                en => '1',
43
                C \Rightarrow clk,
44
                clr => rst.
45
```

```
p_out => s0);
46
        gate2: entity work.RhCount
47
             port map (
48
                  clk => clk,
49
                  C \Rightarrow s3,
50
                  Rst => rst,
51
52
                  Rh \Rightarrow s5;
        gate3: entity work.buffer4bit
53
54
             port map (
55
                  Rst => rst,
56
                  clk => clk,
57
                  Din => s4,
                  Dout \Rightarrow s2);
58
59
        gate4: entity work.buffer8bit
             port map (
60
                  Rh => s5,
61
                  R1 \Rightarrow s4,
62
63
                  Rst => rst,
64
                  clk => clk,
                  Result => Result);
65
66
        gate5: entity work.isMax
67
             port map (
                  N => N,
68
69
                  Addr => s0,
70
                  rst \Rightarrow s6);
        gate6: entity work.EEPROM
71
             port map (
72
73
                  addr => s7,
                  Din => data,
74
                  Rst => rst,
75
76
                  WE => write,
                  clk => str,
77
78
                  Dout \Rightarrow s1);
79
        gate7: entity work.MUX_GATE_BUS_1
             generic map (
80
81
                  Bits \Rightarrow 4)
             port map (
82
83
                  sel => write,
84
                  in_0 => s0,
85
                  in_1 => addr,
                  p_out => s7);
86
        rst <= (forceRst OR s6);
87
        gate8: entity work.adder4bit
88
89
             port map (
                  A \Rightarrow s1,
90
                  B \Rightarrow s2,
91
92
                  C \Rightarrow s3,
93
                  R1 \Rightarrow s4);
94
   end Structural;
```