

**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<sup>1</sup> 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>1</sup>All testbench files can be seen at <https://github.com/Marcobisky/dcd-generalized-adder>

## 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 loses 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>2</sup>All testbench files in VHDL can be seen at <https://github.com/Marcobisky/dcd-generalized-adder>

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

<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. 0 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 <https://github.com/Marcobisky/dcd-generalized-adder> for the Digital source files.

## A Figures for PART II

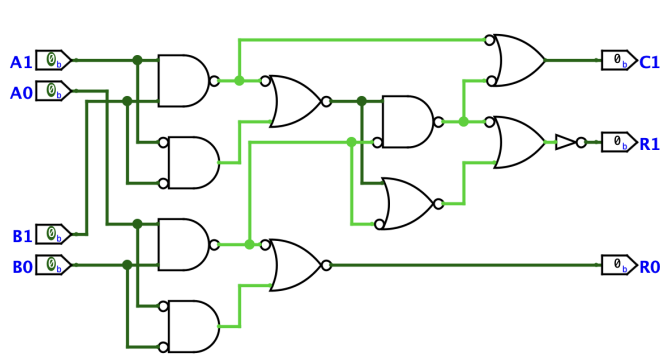


Figure 1: Circuit Structure of the 2-bit Adder

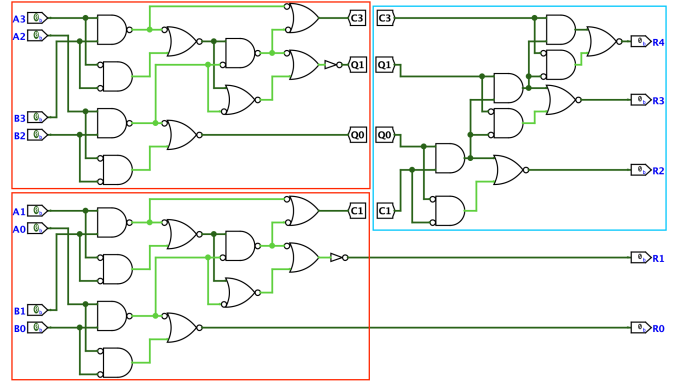


Figure 2: Circuit Structure of the 4-bit Adder

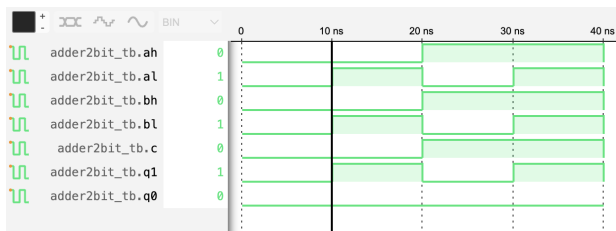


Figure 3: 2-bit adder simulation waveform

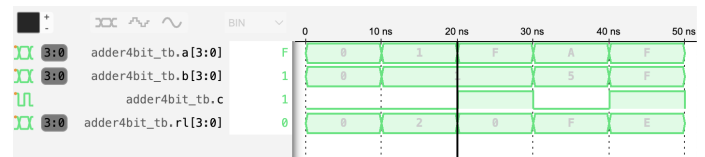


Figure 4: 4-bit adder simulation waveform

## B Circuit Figures for PART III

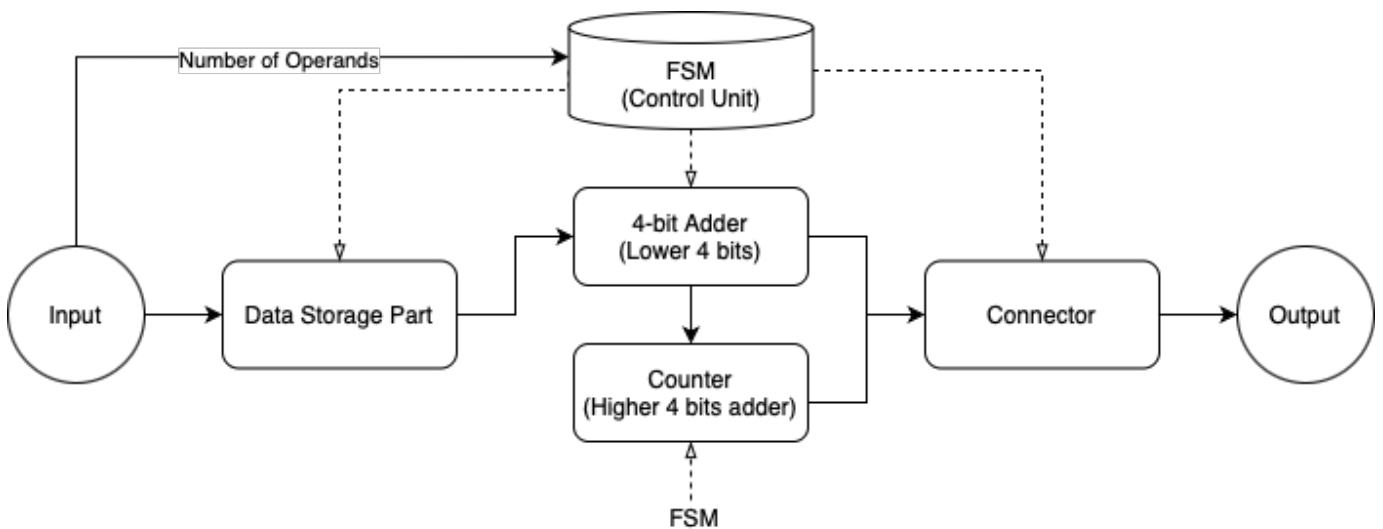


Figure 5: Block diagram of the proposed generalised adder<sup>7</sup>

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

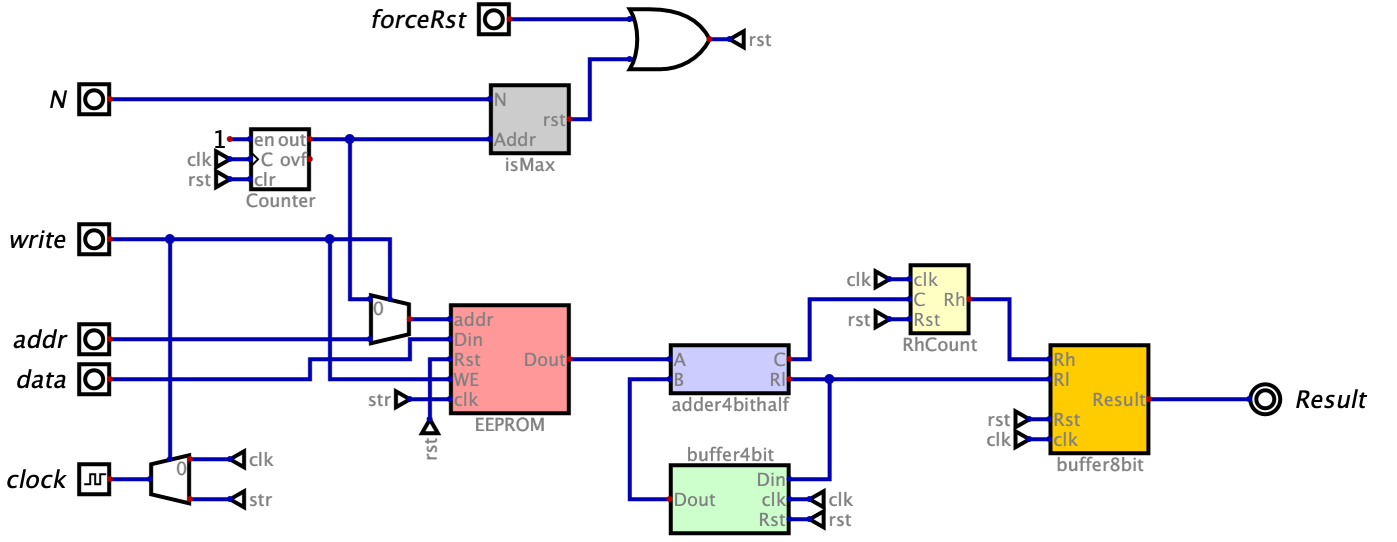


Figure 6: Circuit realization of the proposed generalised adder

### C Simulation Waveforms for PART III

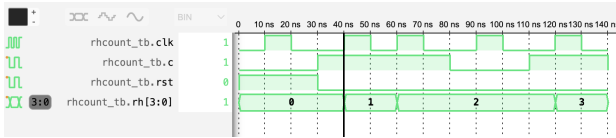


Figure 7: Counter (Higher 4 bits adder) simulation waveform

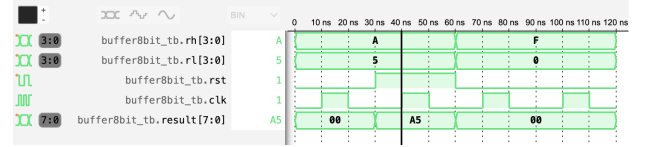


Figure 8: Connector simulation waveform

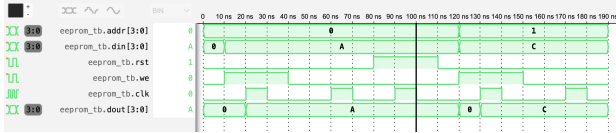


Figure 9: Data Storage Part (EEPROM) simulation waveform

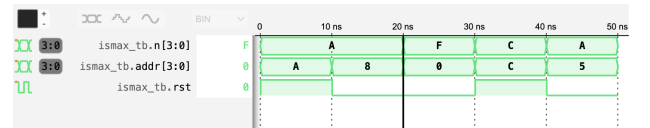


Figure 10: isMax module (Part of the FSM) simulation waveform

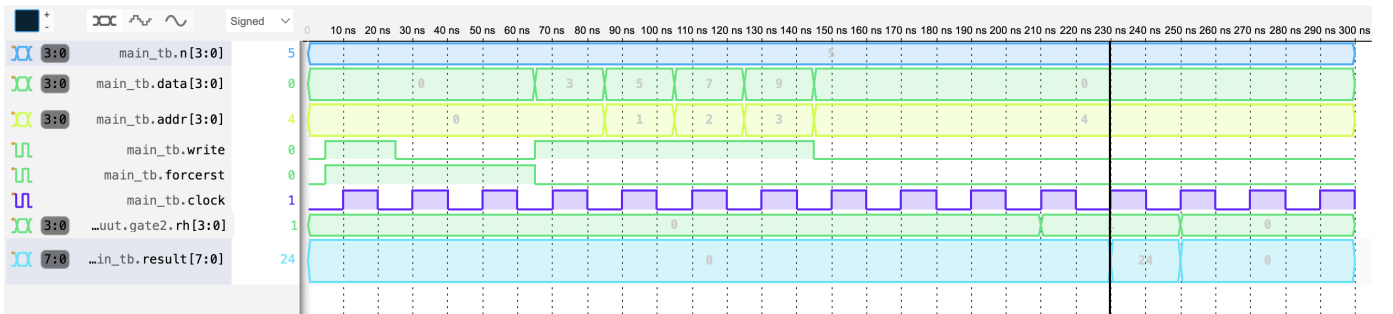


Figure 11: Simulation waveform of the proposed generalised adder under the condition specified in Section 2.4

## D Subcircuit Figures for PART III

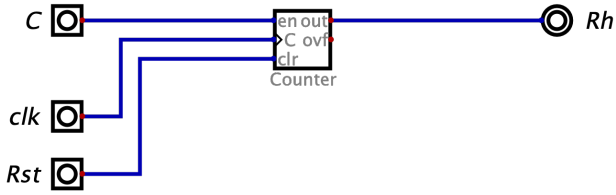


Figure 12: Counter (Higher 4 bits adder) subcircuit

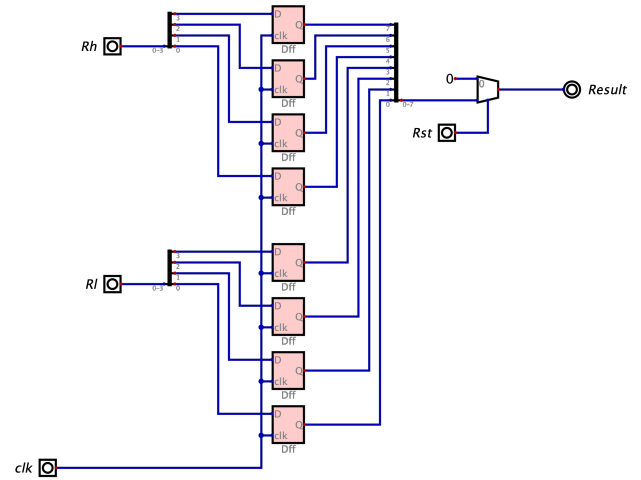


Figure 13: Connector (8-bit Buffer) subcircuit

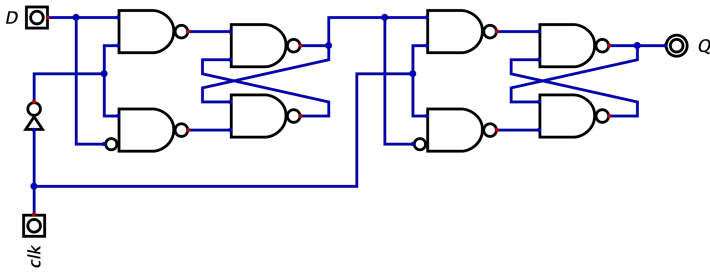


Figure 14: Master-slave D Flip-Flop subcircuit

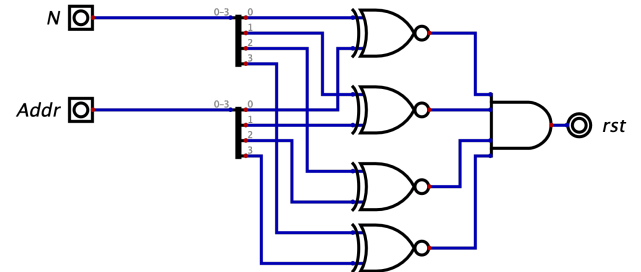


Figure 15: isMax module (Part of the FSM) subcircuit

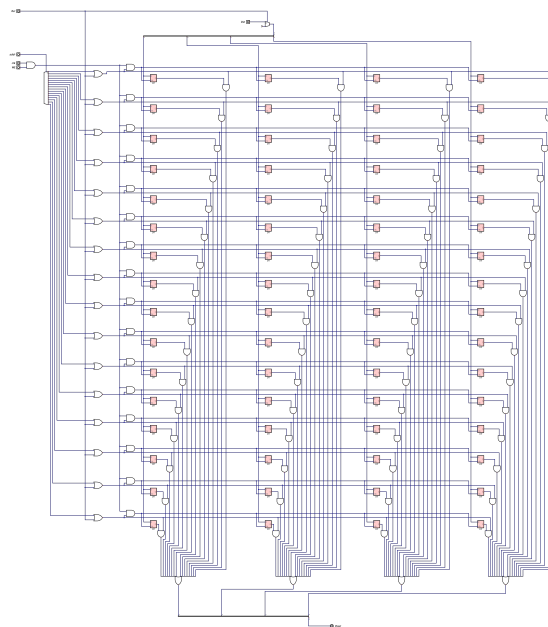


Figure 16: Data Storage Part (EEPROM) subcircuit

## E VHDL Codes

### E.1 VHDL Codes for PART II

Listing 1: Behavioural description of 2-bit adder

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

```

49         sel = "1110" or
50         sel = "1111" else
51         '0';
52 end Behavioral;

```

Listing 2: Behavioural description of 4-bit adder

```

1  adder4bit.vhdl
2  LIBRARY ieee;
3  USE ieee.std_logic_1164.all;
4  USE ieee.numeric_std.all;
5
6  entity adder4bit is
7      port (
8          A: in std_logic_vector(3 downto 0);
9          B: in std_logic_vector(3 downto 0);
10         C: out std_logic; carry (R4)
11         R1: out std_logic_vector(3 downto 0) Lower four bits of the result
12     );
13 end adder4bit;
14
15 architecture Dataflow of adder4bit is
16     signal s0: std_logic;
17     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     signal s7: std_logic;
24     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 begin
33     s11 <= A(0);
34     s10 <= A(1);
35     s7 <= A(2);
36     s6 <= A(3);
37     s13 <= B(0);
38     s12 <= B(1);
39     s9 <= B(2);
40     s8 <= B(3);
41     gate0: entity work.adder2bit
42     port map (
43         Ah => s6,
44         Al => s7,
45         Bh => s8,
46         Bl => s9,
47         C => s5,
48         Q1 => s3,

```



```

49     Q0 => s0);
50 gate1: entity work.adder2bit
51 port map (
52     Ah => s10,
53     Al => s11,
54     Bh => s12,
55     Bl => s13,
56     C => s1,
57     Q1 => s14,
58     Q0 => s15);
59 s2 <= (s0 AND s1);
60 s4 <= (s3 AND s2);
61 C <= NOT ((s5 AND s4) OR (NOT s5 AND NOT s4));
62 Rl(0) <= s15;
63 Rl(1) <= s14;
64 Rl(2) <= NOT (s2 OR (NOT s0 AND NOT s1));
65 Rl(3) <= NOT (s4 OR (NOT s3 AND NOT s2));
66 end Dataflow;

```

## 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

```

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

```

```

30         p_out <= count;
31         ovf <= en when count = ((2**Bits)-1) else '0';
32     end Behavioral;
33

```

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

```

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

```

### E.2.2 Decoder

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

```

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

```

```

17     out_11: out std_logic;
18     out_12: out std_logic;
19     out_13: out std_logic;
20     out_14: out std_logic;
21     out_15: out std_logic;
22     sel: in std_logic_vector (3 downto 0) );
23 end DECODER_4;
24
25 architecture Behavioral of DECODER_4 is
26 begin
27     out_0 <= '1' when sel = "0000" else '0';
28     out_1 <= '1' when sel = "0001" else '0';
29     out_2 <= '1' when sel = "0010" else '0';
30     out_3 <= '1' when sel = "0011" else '0';
31     out_4 <= '1' when sel = "0100" else '0';
32     out_5 <= '1' when sel = "0101" else '0';
33     out_6 <= '1' when sel = "0110" else '0';
34     out_7 <= '1' when sel = "0111" else '0';
35     out_8 <= '1' when sel = "1000" else '0';
36     out_9 <= '1' when sel = "1001" else '0';
37     out_10 <= '1' when sel = "1010" else '0';
38     out_11 <= '1' when sel = "1011" else '0';
39     out_12 <= '1' when sel = "1100" else '0';
40     out_13 <= '1' when sel = "1101" else '0';
41     out_14 <= '1' when sel = "1110" else '0';
42     out_15 <= '1' when sel = "1111" else '0';
43 end Behavioral;

```

### E.2.3 Multiplexer

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

```

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

```

## E.2.4 D Flip-Flop

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

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

## E.2.5 EEPROM (Data Storage Part)

Listing 8: EEPROM as the subcircuit for main

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

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

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

```

130 signal s112: std_logic;
131 signal s113: std_logic;
132 signal s114: std_logic;
133 signal s115: std_logic;
134 signal s116: std_logic;
135 signal s117: std_logic;
136 begin
137   gate0: entity work.DECODER_4
138     port map (
139       sel => addr,
140       out_0 => s101,
141       out_1 => s102,
142       out_2 => s103,
143       out_3 => s104,
144       out_4 => s105,
145       out_5 => s106,
146       out_6 => s107,
147       out_7 => s108,
148       out_8 => s109,
149       out_9 => s110,
150       out_10 => s111,
151       out_11 => s112,
152       out_12 => s113,
153       out_13 => s114,
154       out_14 => s115,
155       out_15 => s116);
156   s117 <= (clk AND WE);
157   gate1: entity work.MUX_GATE_BUS_1
158     generic map (
159       Bits => 4)
160     port map (
161       sel => Rst,
162       in_0 => Din,
163       in_1 => "0000",
164       p_out => s3);
165   s52 <= (s101 OR Rst);
166   s49 <= (s102 OR Rst);
167   s46 <= (s103 OR Rst);
168   s43 <= (s104 OR Rst);
169   s40 <= (s105 OR Rst);
170   s37 <= (s106 OR Rst);
171   s34 <= (s107 OR Rst);
172   s31 <= (s108 OR Rst);
173   s28 <= (s109 OR Rst);
174   s25 <= (s110 OR Rst);
175   s22 <= (s111 OR Rst);
176   s19 <= (s112 OR Rst);
177   s16 <= (s113 OR Rst);
178   s13 <= (s114 OR Rst);
179   s10 <= (s115 OR Rst);
180   s7 <= (s116 OR Rst);
181   s4 <= s3(0);
182   s5 <= s3(1);
183   s6 <= s3(2);
184   s0 <= s3(3);

```

```

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

```



```

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

```

```

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

```

```

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

```

```

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

```

```

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

```

```

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

```

## E.2.6 FSM (Control Unit)

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

```

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

```

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

Listing 10: buffer4bit as the subcircuit for main

```

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

```

```

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

```

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

Listing 11: buffer8bit as the subcircuit for main

```

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

```



```

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

```

```

96         in_0 => "00000000",
97         in_1 => s16,
98         p_out => Result);
99 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

```

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

```

```

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

```