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[1] 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.

[1]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[2] methods and results are presented in sections below[3].

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[4] 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.

[2]All testbench files in VHDL can be seen at `https://github.com/Marcobisky/dcd-generalized-adder`

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

[4]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[5]:

- $N = 4 + 1 = 5$.
- 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[6]):

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.

[5]The last two numbers in decimal are 7 and 9, which are the sum of last two digits of our GUIDs.

[6]For dynamic and visualized simulation, check out `https://github.com/Marcobisky/dcd-generalized-adder` for the `Digital` source files.

A Figures for PART II

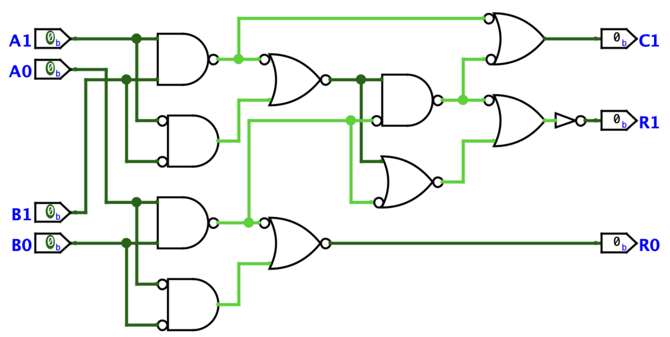


Figure 1: Circuit Structure of the 2-bit Adder

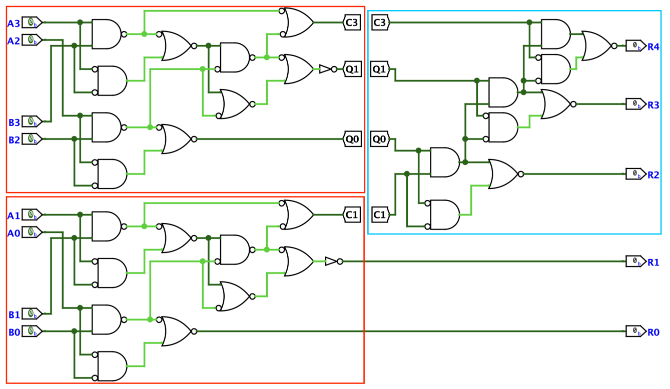
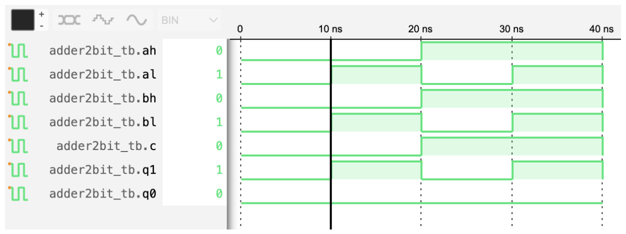
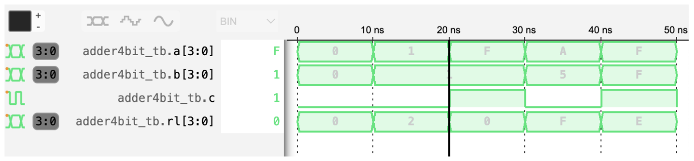


Figure 2: Circuit Structure of the 4-bit Adder



```
adder2bit_tb.ah   0
adder2bit_tb.al   1
adder2bit_tb.bh   0
adder2bit_tb.bl   1
adder2bit_tb.c    0
adder2bit_tb.q1   1
adder2bit_tb.q0   0
```

Figure 3: 2-bit adder simulation waveform



```
3:0  adder4bit_tb.a[3:0]    F   0 | 1 | F | A | F
3:0  adder4bit_tb.b[3:0]    1   0 |   |   | 5 | F
     adder4bit_tb.c         1
3:0  adder4bit_tb.rl[3:0]   0   0 | 2 | 0 | F | E
```

Figure 4: 4-bit adder simulation waveform

B Circuit Figures for PART III

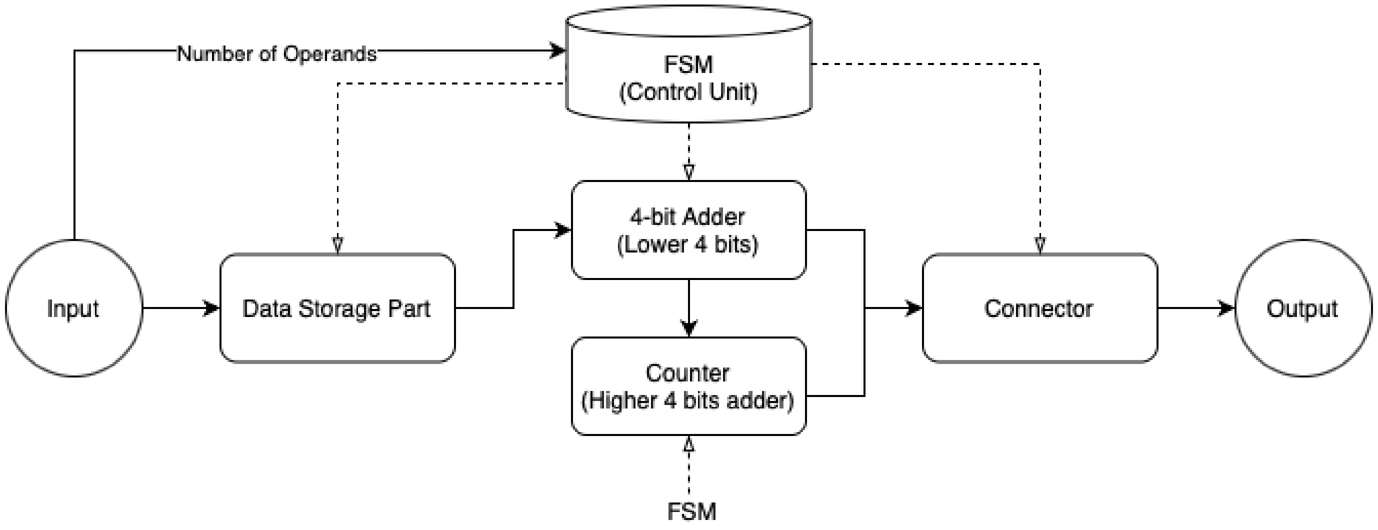


Figure 5: Block diagram of the proposed generalised adder[7]

[7] Filled-in arrow denotes data flow, not-filled-in arrow denotes control signals and clock signal.

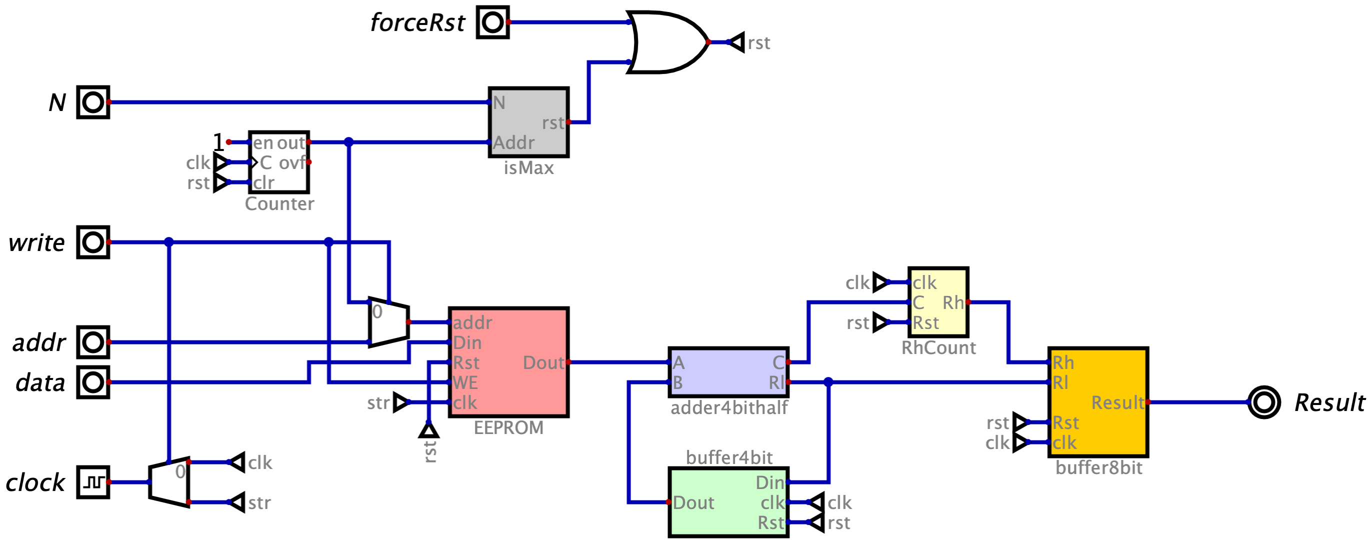


Figure 6: Circuit realization of the proposed generalised adder

C Simulation Waveforms for PART III

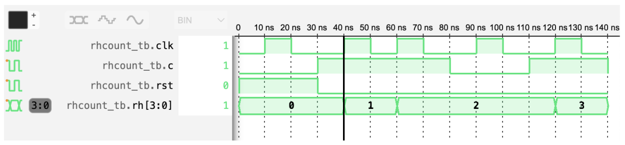


Figure 7: Counter (Higher 4 bits adder) simulation waveform

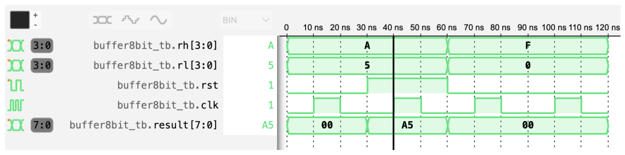


Figure 8: Connector simulation waveform

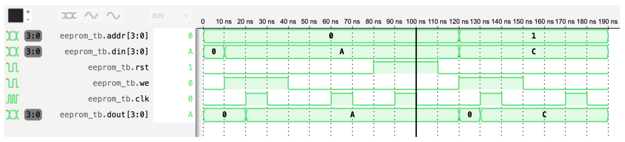


Figure 9: Data Storage Part (EEPROM) simulation waveform

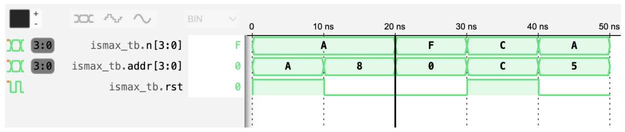


Figure 10: `isMax` module (Part of the FSM) simulation waveform

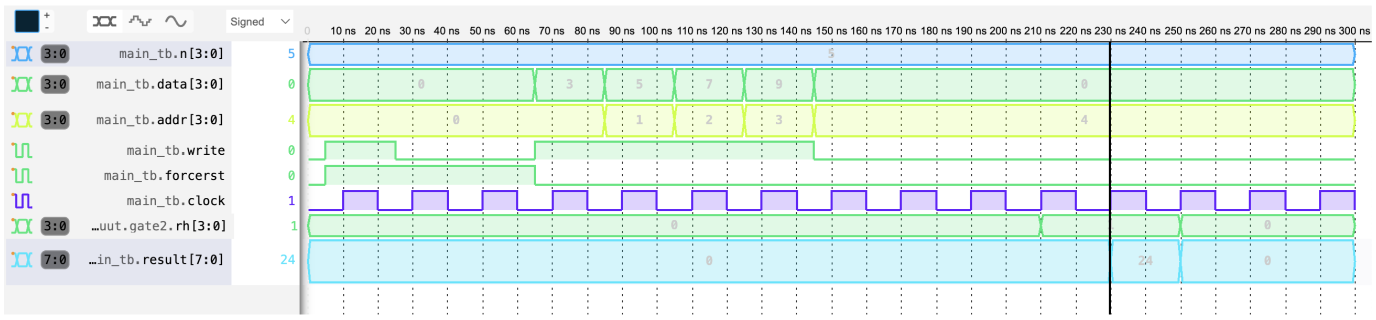


Figure 11: Simulation waveform of the proposed generalised adder under the condition specified in Section 2.4

D Subcircuit Figures for PART III

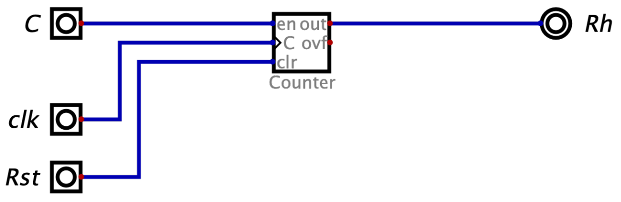


Figure 12: Counter (Higher 4 bits adder) subcircuit

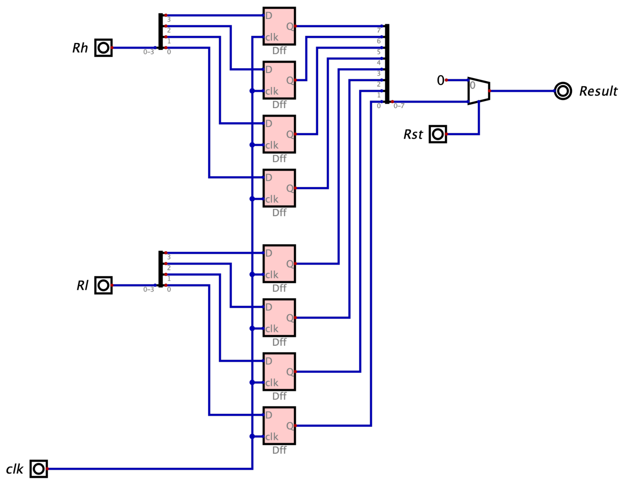


Figure 13: Connector (8-bit Buffer) subcircuit

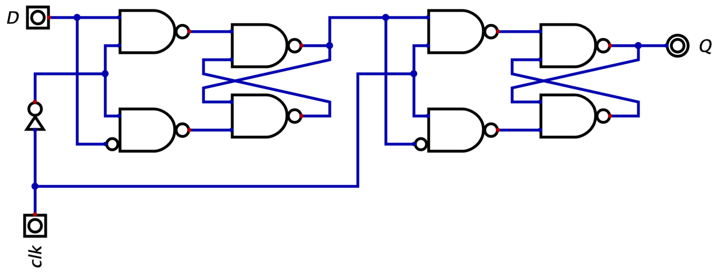


Figure 14: Master-slave D Flip-Flop subcircuit

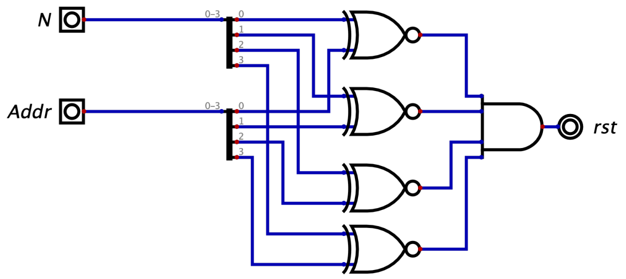


Figure 15: `isMax` module (Part of the FSM) subcircuit

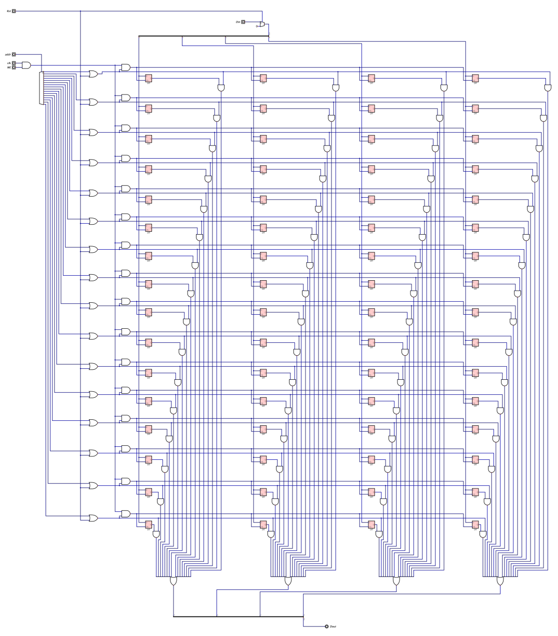


Figure 16: Data Storage Part (EEPROM) subcircuit

E VHDL Codes

E.1 VHDL Codes for PART II

Listing 1: Behavioural description of 2-bit adder

```
  adder2bit.vhdl
LIBRARY ieee;
USE ieee.std_logic_1164.all;

entity adder2bit is
    port (
        Ah: in std_logic;
        Al: in std_logic;
        Bh: in std_logic;
        Bl: in std_logic;
        C: out std_logic;
        Q1: out std_logic;
        Q0: out std_logic);
end adder2bit;

architecture Behavioral of adder2bit is
    signal sel: std_logic_vector(3 downto 0);  Combine inputs into a single
   signal
begin
      Combine inputs into a 4bit signal
    sel <= Ah & Al & Bh & Bl;

      Enumerate all possible input combinations and assign Q0
    Q0 <= '1' when sel = "0001" or
                   sel = "0011" or
                   sel = "0100" or
                   sel = "0110" or
                   sel = "1001" or
                   sel = "1011" or
                   sel = "1100" or
                   sel = "1110" else
          '0';

      Enumerate all possible input combinations and assign Q1
    Q1 <= '1' when sel = "0010" or
                   sel = "0011" or
                   sel = "0101" or
                   sel = "0110" or
                   sel = "1000" or
                   sel = "1001" or
                   sel = "1100" or
                   sel = "1111" else
          '0';

      Enumerate all possible input combinations and assign C
    C <= '1' when sel = "0111" or
                  sel = "1010" or
                  sel = "1011" or
                  sel = "1101" or
                  sel = "1110" or
                  sel = "1111" else
          '0';
end Behavioral;
```

Listing 2: Behavioural description of 4-bit adder

```
   adder4bit.vhdl
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.all;

entity adder4bit is
    port (
        A: in std_logic_vector(3 downto 0);
        B: in std_logic_vector(3 downto 0);
        C: out std_logic;  carry (R4)
        Rl: out std_logic_vector(3 downto 0)  Lower four bits of the result
    );
end adder4bit;

architecture Dataflow of adder4bit is
    signal s0: std_logic;
    signal s1: std_logic;
    signal s2: std_logic;
    signal s3: std_logic;
    signal s4: std_logic;
    signal s5: std_logic;
    signal s6: std_logic;
    signal s7: std_logic;
    signal s8: std_logic;
    signal s9: std_logic;
    signal s10: std_logic;
    signal s11: std_logic;
    signal s12: std_logic;
    signal s13: std_logic;
    signal s14: std_logic;
    signal s15: std_logic;
begin
    s11 <= A(0);
    s10 <= A(1);
    s7 <= A(2);
    s6 <= A(3);
    s13 <= B(0);
    s12 <= B(1);
    s9 <= B(2);
    s8 <= B(3);
    gate0: entity work.adder2bit
    port map (
        Ah => s6,
        Al => s7,
        Bh => s8,
        Bl => s9,
        C => s5,
        Q1 => s3,
        Q0 => s0);
    gate1: entity work.adder2bit
    port map (
        Ah => s10,
        Al => s11,
        Bh => s12,
        Bl => s13,
        C => s1,
        Q1 => s14,
        Q0 => s15);
    s2 <= (s0 AND s1);
    s4 <= (s3 AND s2);
    C <= NOT ((s5 AND s4) OR (NOT s5 AND NOT s4));
    Rl(0) <= s15;
    Rl(1) <= s14;
    Rl(2) <= NOT (s2 OR (NOT s0 AND NOT s1));
    Rl(3) <= NOT (s4 OR (NOT s3 AND NOT s2));
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`

```
   DIG_Counter.vhdl
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_unsigned.all;
USE ieee.numeric_std.all;

entity DIG_Counter is
    generic ( Bits: integer );
    port (
        p_out: out std_logic_vector((Bits1) downto 0);
        ovf: out std_logic;
        C: in std_logic;
        en: in std_logic;
        clr: in std_logic );
end DIG_Counter;

architecture Behavioral of DIG_Counter is
     signal count : std_logic_vector((Bits1) downto 0) := (others => '0');
begin
        process (C, clr, en)
        begin
            if rising_edge(C) then
                if clr='1' then
                    count <= (others => '0');
                elsif en='1' then
                    count <= count + 1;
                end if;
            end if;
        end process;

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

Listing 4: `RhCount` is just an instantiation of `DIG_Counter`

```
   RhCount.vhdl

LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.all;

entity RhCount is
    port (
        clk: in std_logic;
        C: in std_logic;
        Rst: in std_logic;
        Rh: out std_logic_vector(3 downto 0)  Higher four bits of the result
        );
end RhCount;

architecture Behavioral of RhCount is
begin
    gate0: entity work.DIG_Counter
        generic map (
            Bits => 4)
        port map (
            en => C,
            C => clk,
            clr => Rst,
            p_out => Rh);
end Behavioral;
```

E.2.2 Decoder

Listing 5: `DECODER_4` as the subcircuit for `EEPROM` and `main`

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

entity DECODER_4 is
    port (
        out_0: out std_logic;
        out_1: out std_logic;
        out_2: out std_logic;
        out_3: out std_logic;
        out_4: out std_logic;
        out_5: out std_logic;
        out_6: out std_logic;
        out_7: out std_logic;
        out_8: out std_logic;
        out_9: out std_logic;
        out_10: out std_logic;
        out_11: out std_logic;
        out_12: out std_logic;
        out_13: out std_logic;
        out_14: out std_logic;
        out_15: out std_logic;
        sel: in std_logic_vector (3 downto 0) );
end DECODER_4;

architecture Behavioral of DECODER_4 is
begin
    out_0 <= '1' when sel = "0000" else '0';
    out_1 <= '1' when sel = "0001" else '0';
    out_2 <= '1' when sel = "0010" else '0';
    out_3 <= '1' when sel = "0011" else '0';
    out_4 <= '1' when sel = "0100" else '0';
    out_5 <= '1' when sel = "0101" else '0';
    out_6 <= '1' when sel = "0110" else '0';
    out_7 <= '1' when sel = "0111" else '0';
    out_8 <= '1' when sel = "1000" else '0';
    out_9 <= '1' when sel = "1001" else '0';
    out_10 <= '1' when sel = "1010" else '0';
    out_11 <= '1' when sel = "1011" else '0';
    out_12 <= '1' when sel = "1100" else '0';
    out_13 <= '1' when sel = "1101" else '0';
    out_14 <= '1' when sel = "1110" else '0';
    out_15 <= '1' when sel = "1111" else '0';
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
LIBRARY ieee;
USE ieee.std_logic_1164.all;

entity MUX_GATE_BUS_1 is
    generic ( Bits : integer );
    port (
        p_out: out std_logic_vector ((Bits1) downto 0);
        sel: in std_logic;

        in_0: in std_logic_vector ((Bits1) downto 0);
        in_1: in std_logic_vector ((Bits1) downto 0) );
end MUX_GATE_BUS_1;

architecture Behavioral of MUX_GATE_BUS_1 is
begin
    with sel select
        p_out <=
            in_0 when '0',
            in_1 when '1',
            (others => '0') when others;
end Behavioral;
```

E.2.4 D Flip-Flop

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

```vhdl
   Dff.vhdl
LIBRARY ieee;
USE ieee.std_logic_1164.all;

entity Dff is
    port (
        D   : in std_logic;   Data input
        clk : in std_logic;   Clock input
        Q   : out std_logic   Output
    );
end Dff;

architecture Behavioral of Dff is
    signal Q_internal : std_logic := '0';   Internal signal for flipflop state
begin
    process(clk)
    begin
        if rising_edge(clk) then
            Q_internal <= D;   Update internal state on clock's rising edge
        end if;
    end process;

    Q <= Q_internal;   Assign internal state to output
end Behavioral;
```

E.2.5 EEPROM (Data Storage Part)

Listing 8: `EEPROM` as the subcircuit for `main`

```vhdl
   EEPROM.vhdl
   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.)
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.all;

entity EEPROM is
    port (
        addr: in std_logic_vector(3 downto 0);
        Din: in std_logic_vector(3 downto 0);
        Rst: in std_logic;
        WE: in std_logic;
        clk: in std_logic;
        Dout: out std_logic_vector(3 downto 0));
end EEPROM;

architecture Structural of EEPROM is
    signal s0: std_logic;
    signal s1: std_logic;
    signal s2: std_logic;
    signal s3: std_logic_vector(3 downto 0);
    signal s4: std_logic;
    signal s5: std_logic;
    signal s6: std_logic;
    signal s7: std_logic;
    signal s8: std_logic;
    signal s9: std_logic;
    signal s10: std_logic;
    signal s11: std_logic;
    signal s12: std_logic;
    signal s13: std_logic;
    signal s14: std_logic;
    signal s15: std_logic;
    signal s16: std_logic;
    signal s17: std_logic;
    signal s18: std_logic;
    signal s19: std_logic;
    signal s20: std_logic;
    signal s21: std_logic;
    signal s22: std_logic;
    signal s23: std_logic;
    signal s24: std_logic;
    signal s25: std_logic;
    signal s26: std_logic;
    signal s27: std_logic;
    signal s28: std_logic;
    signal s29: std_logic;
    signal s30: std_logic;
    signal s31: std_logic;
    signal s32: std_logic;
    signal s33: std_logic;
    signal s34: std_logic;
    signal s35: std_logic;
    signal s36: std_logic;
    signal s37: std_logic;
    signal s38: std_logic;
    signal s39: std_logic;
    signal s40: std_logic;
    signal s41: std_logic;
    signal s42: std_logic;
    signal s43: std_logic;
    signal s44: std_logic;
    signal s45: std_logic;
    signal s46: std_logic;
    signal s47: std_logic;
    signal s48: std_logic;
    signal s49: std_logic;
    signal s50: std_logic;
    signal s51: std_logic;
    signal s52: std_logic;
    signal s53: std_logic;
    signal s54: std_logic;
    signal s55: std_logic;
    signal s56: std_logic;
    signal s57: std_logic;
    signal s58: std_logic;
    signal s59: std_logic;
    signal s60: std_logic;
    signal s61: std_logic;
    signal s62: std_logic;
    signal s63: std_logic;
    signal s64: std_logic;
    signal s65: std_logic;
    signal s66: std_logic;
    signal s67: std_logic;
    signal s68: std_logic;
    signal s69: std_logic;
    signal s70: std_logic;
    signal s71: std_logic;
    signal s72: std_logic;
    signal s73: std_logic;
    signal s74: std_logic;
    signal s75: std_logic;
    signal s76: std_logic;
    signal s77: std_logic;
    signal s78: std_logic;
    signal s79: std_logic;
    signal s80: std_logic;
    signal s81: std_logic;
    signal s82: std_logic;
    signal s83: std_logic;
    signal s84: std_logic;
    signal s85: std_logic;
    signal s86: std_logic;
    signal s87: std_logic;
    signal s88: std_logic;
    signal s89: std_logic;
    signal s90: std_logic;
    signal s91: std_logic;
    signal s92: std_logic;
    signal s93: std_logic;
    signal s94: std_logic;
    signal s95: std_logic;
    signal s96: std_logic;
    signal s97: std_logic;
    signal s98: std_logic;
    signal s99: std_logic;
    signal s100: std_logic;
    signal s101: std_logic;
    signal s102: std_logic;
    signal s103: std_logic;
    signal s104: std_logic;
    signal s105: std_logic;
    signal s106: std_logic;
    signal s107: std_logic;
    signal s108: std_logic;
    signal s109: std_logic;
    signal s110: std_logic;
    signal s111: std_logic;
    signal s112: std_logic;
    signal s113: std_logic;
    signal s114: std_logic;
    signal s115: std_logic;
    signal s116: std_logic;
    signal s117: std_logic;
begin
    gate0: entity work.DECODER_4
        port map (
            sel => addr,
            out_0 => s101,
            out_1 => s102,
            out_2 => s103,
            out_3 => s104,
            out_4 => s105,
            out_5 => s106,
            out_6 => s107,
            out_7 => s108,
            out_8 => s109,
            out_9 => s110,
            out_10 => s111,
            out_11 => s112,
            out_12 => s113,
            out_13 => s114,
            out_14 => s115,
            out_15 => s116);
    s117 <= (clk AND WE);
    gate1: entity work.MUX_GATE_BUS_1
        generic map (
            Bits => 4)
        port map (
            sel => Rst,
            in_0 => Din,
            in_1 => "0000",
            p_out => s3);
    s52 <= (s101 OR Rst);
    s49 <= (s102 OR Rst);
    s46 <= (s103 OR Rst);
    s43 <= (s104 OR Rst);
    s40 <= (s105 OR Rst);
    s37 <= (s106 OR Rst);
    s34 <= (s107 OR Rst);
    s31 <= (s108 OR Rst);
    s28 <= (s109 OR Rst);
    s25 <= (s110 OR Rst);
    s22 <= (s111 OR Rst);
    s19 <= (s112 OR Rst);
    s16 <= (s113 OR Rst);
    s13 <= (s114 OR Rst);
    s10 <= (s115 OR Rst);
    s7 <= (s116 OR Rst);
    s4 <= s3(0);
    s5 <= s3(1);
    s6 <= s3(2);
    s0 <= s3(3);
    s50 <= (s117 AND s52);
    s47 <= (s117 AND s49);
    s44 <= (s117 AND s46);
    s41 <= (s117 AND s43);
    s38 <= (s117 AND s40);
    s35 <= (s117 AND s37);
    s32 <= (s117 AND s34);
    s29 <= (s117 AND s31);
    s26 <= (s117 AND s28);
    s23 <= (s117 AND s25);
    s20 <= (s117 AND s22);
    s17 <= (s117 AND s19);
    s14 <= (s117 AND s16);
    s11 <= (s117 AND s13);
    s8 <= (s117 AND s10);
    s1 <= (s117 AND s7);
    gate2: entity work.Dff
        port map (
            D => s0,
            clk => s1,
            Q => s2);
    gate3: entity work.Dff
        port map (
            D => s0,
            clk => s8,
            Q => s9);
    gate4: entity work.Dff
        port map (
            D => s0,
            clk => s11,
            Q => s12);
    gate5: entity work.Dff
        port map (
            D => s0,
            clk => s14,
            Q => s15);
    gate6: entity work.Dff
        port map (
            D => s0,
            clk => s17,
            Q => s18);
    gate7: entity work.Dff
        port map (
            D => s0,
            clk => s20,
            Q => s21);
    gate8: entity work.Dff
        port map (
            D => s0,
            clk => s23,
            Q => s24);
    gate9: entity work.Dff
        port map (
            D => s0,
            clk => s26,
            Q => s27);
    gate10: entity work.Dff
        port map (
            D => s0,
            clk => s29,
            Q => s30);
    gate11: entity work.Dff
        port map (
            D => s0,
            clk => s32,
            Q => s33);
    gate12: entity work.Dff
        port map (
            D => s0,
            clk => s35,
            Q => s36);
    gate13: entity work.Dff
        port map (
            D => s0,
            clk => s38,
            Q => s39);
    gate14: entity work.Dff
        port map (
            D => s0,
            clk => s41,
            Q => s42);
    gate15: entity work.Dff
        port map (
            D => s0,
            clk => s44,
            Q => s45);
    gate16: entity work.Dff
        port map (
            D => s0,
            clk => s47,
            Q => s48);
    gate17: entity work.Dff
        port map (
            D => s0,
            clk => s50,
            Q => s51);
    gate18: entity work.Dff
        port map (
            D => s6,
            clk => s1,
            Q => s53);
    gate19: entity work.Dff
        port map (
            D => s6,
            clk => s8,
            Q => s54);
    gate20: entity work.Dff
        port map (
            D => s6,
            clk => s11,
                Q => s55);
    gate21: entity work.Dff
        port map (
            D => s6,
            clk => s14,
            Q => s56);
    gate22: entity work.Dff
        port map (
            D => s6,
            clk => s17,
            Q => s57);
    gate23: entity work.Dff
        port map (
            D => s6,
            clk => s20,
            Q => s58);
    gate24: entity work.Dff
        port map (
            D => s6,
            clk => s23,
            Q => s59);
    gate25: entity work.Dff
        port map (
            D => s6,
            clk => s26,
            Q => s60);
    gate26: entity work.Dff
        port map (
            D => s6,
            clk => s29,
            Q => s61);
    gate27: entity work.Dff
        port map (
            D => s6,
            clk => s32,
            Q => s62);
    gate28: entity work.Dff
        port map (
            D => s6,
            clk => s35,
            Q => s63);
    gate29: entity work.Dff
        port map (
            D => s6,
            clk => s38,
            Q => s64);
    gate30: entity work.Dff
        port map (
            D => s6,
            clk => s41,
            Q => s65);
    gate31: entity work.Dff
        port map (
            D => s6,
            clk => s44,
            Q => s66);
    gate32: entity work.Dff
        port map (
            D => s6,
            clk => s47,
            Q => s67);
    gate33: entity work.Dff
        port map (
            D => s6,
            clk => s50,
            Q => s68);
    gate34: entity work.Dff
        port map (
            D => s5,
            clk => s1,
            Q => s69);
    gate35: entity work.Dff
        port map (
            D => s5,
            clk => s8,
            Q => s70);
    gate36: entity work.Dff
        port map (
            D => s5,
            clk => s11,
            Q => s71);
    gate37: entity work.Dff
        port map (
            D => s5,
            clk => s14,
            Q => s72);
    gate38: entity work.Dff
        port map (
            D => s5,
            clk => s17,
            Q => s73);
    gate39: entity work.Dff
        port map (
            D => s5,
            clk => s20,
            Q => s74);
    gate40: entity work.Dff
        port map (
            D => s5,
            clk => s23,
            Q => s75);
    gate41: entity work.Dff
        port map (
            D => s5,
            clk => s26,
            Q => s76);
    gate42: entity work.Dff
        port map (
            D => s5,
            clk => s29,
            Q => s77);
    gate43: entity work.Dff
        port map (
            D => s5,
            clk => s32,
            Q => s78);
    gate44: entity work.Dff
        port map (
            D => s5,
            clk => s35,
            Q => s79);
    gate45: entity work.Dff
        port map (
            D => s5,
            clk => s38,
            Q => s80);
    gate46: entity work.Dff
        port map (
            D => s5,
            clk => s41,
            Q => s81);
    gate47: entity work.Dff
        port map (
            D => s5,
            clk => s44,
            Q => s82);
    gate48: entity work.Dff
        port map (
            D => s5,
            clk => s47,
            Q => s83);
    gate49: entity work.Dff
        port map (
            D => s5,
            clk => s50,
            Q => s84);
    gate50: entity work.Dff
        port map (
            D => s4,
            clk => s1,
            Q => s85);
    gate51: entity work.Dff
        port map (
            D => s4,
            clk => s8,
            Q => s86);
    gate52: entity work.Dff
        port map (
            D => s4,
            clk => s11,
            Q => s87);
    gate53: entity work.Dff
        port map (
            D => s4,
            clk => s14,
            Q => s88);
    gate54: entity work.Dff
        port map (
            D => s4,
            clk => s17,
            Q => s89);
    gate55: entity work.Dff
        port map (
            D => s4,
            clk => s20,
            Q => s90);
    gate56: entity work.Dff
        port map (
            D => s4,
            clk => s23,
            Q => s91);
    gate57: entity work.Dff
        port map (
            D => s4,
            clk => s26,
            Q => s92);
    gate58: entity work.Dff
        port map (
            D => s4,
            clk => s29,
            Q => s93);
    gate59: entity work.Dff
        port map (
            D => s4,
            clk => s32,
            Q => s94);
    gate60: entity work.Dff
        port map (
            D => s4,
            clk => s35,
            Q => s95);
    gate61: entity work.Dff
        port map (
            D => s4,
            clk => s38,
            Q => s96);
    gate62: entity work.Dff
        port map (
            D => s4,
            clk => s41,
            Q => s97);
    gate63: entity work.Dff
        port map (
            D => s4,
            clk => s44,
            Q => s98);
    gate64: entity work.Dff
        port map (
            D => s4,
            clk => s47,
                Q => s99);
    gate65: entity work.Dff
        port map (
            D => s4,
            clk => s50,
            Q => s100);
    Dout(0) <= ((s52 AND s100) OR (s49 AND s99) OR (s46 AND s98) OR (s43 AND s97) 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) 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) 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) <= ((s52 AND s51) OR (s49 AND s48) OR (s46 AND s45) OR (s43 AND s42) 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;
```

E.2.6 FSM (Control Unit)

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

```
   isMax.vhdl
LIBRARY ieee;
USE ieee.std_logic_1164.all;

entity isMax is
    port (
        N: in std_logic_vector(3 downto 0);
        Addr: in std_logic_vector(3 downto 0);
        rst: out std_logic);
end isMax;

architecture Behavioral of isMax is
begin
    process(N, Addr)
    begin
        if N = Addr then
            rst <= '1';   rst is asserted when N equals Addr
        else
            rst <= '0';   rst is deasserted otherwise
        end if;
    end process;
end Behavioral;
```

E.2.7 Counter (Higher 4 bits adder)

Listing 10: `buffer4bit` as the subcircuit for `main`

```
   buffer4bit.vhdl

LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.all;

entity buffer4bit is
    port (
        Rst: in std_logic;
        clk: in std_logic;
        Din: in std_logic_vector(3 downto 0);
        Dout: out std_logic_vector(3 downto 0));
end buffer4bit;

architecture Structural of buffer4bit is
    signal s0: std_logic_vector(3 downto 0);
    signal s1: std_logic;
    signal s2: std_logic;
    signal s3: std_logic;
    signal s4: std_logic;
    signal s5: std_logic;
    signal s6: std_logic;
    signal s7: std_logic;
    signal s8: std_logic;
begin
    gate0: entity work.MUX_GATE_BUS_1
        generic map (
            Bits => 4)
        port map (
            sel => Rst,
            in_0 => Din,
            in_1 => "0000",
            p_out => s0);
    s1 <= s0(0);
    s3 <= s0(1);
    s5 <= s0(2);
    s7 <= s0(3);
    gate1: entity work.Dff
        port map (
            D => s1,
            clk => clk,
            Q => s2);
    gate2: entity work.Dff
        port map (
            D => s3,
            clk => clk,
            Q => s4);
    gate3: entity work.Dff
        port map (
            D => s5,
            clk => clk,
            Q => s6);
    gate4: entity work.Dff
```

```
        port map (
            D => s7,
            clk => clk,
            Q => s8);
    Dout(0) <= s2;
    Dout(1) <= s4;
    Dout(2) <= s6;
    Dout(3) <= s8;
end Structural;
```

E.2.8 Connector (8-bit Buffer)

Listing 11: `buffer8bit` as the subcircuit for `main`

```
   buffer8bit.vhdl

LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.all;

entity buffer8bit is
    port (
        Rh: in std_logic_vector(3 downto 0);
        Rl: in std_logic_vector(3 downto 0);
        Rst: in std_logic;
        clk: in std_logic;
        Result: out std_logic_vector(7 downto 0));
end buffer8bit;

architecture Structural of buffer8bit is
    signal s0: std_logic;
    signal s1: std_logic;
    signal s2: std_logic;
    signal s3: std_logic;
    signal s4: std_logic;
    signal s5: std_logic;
    signal s6: std_logic;
    signal s7: std_logic;
    signal s8: std_logic;
    signal s9: std_logic;
    signal s10: std_logic;
    signal s11: std_logic;
    signal s12: std_logic;
    signal s13: std_logic;
    signal s14: std_logic;
    signal s15: std_logic;
    signal s16: std_logic_vector(7 downto 0);
begin
    s6 <= Rh(0);
    s4 <= Rh(1);
    s2 <= Rh(2);
    s0 <= Rh(3);
    s14 <= Rl(0);
    s12 <= Rl(1);
```

```vhdl
    s10 <= Rl(2);
    s8 <= Rl(3);
    gate0: entity work.Dff
        port map (
            D => s0,
            clk => clk,
            Q => s1);
    gate1: entity work.Dff
        port map (
            D => s2,
            clk => clk,
            Q => s3);
    gate2: entity work.Dff
        port map (
            D => s4,
            clk => clk,
            Q => s5);
    gate3: entity work.Dff
        port map (
            D => s6,
            clk => clk,
            Q => s7);
    gate4: entity work.Dff
        port map (
            D => s8,
            clk => clk,
            Q => s9);
    gate5: entity work.Dff
        port map (
            D => s10,
            clk => clk,
            Q => s11);
    gate6: entity work.Dff
        port map (
            D => s12,
            clk => clk,
            Q => s13);
    gate7: entity work.Dff
        port map (
            D => s14,
            clk => clk,
            Q => s15);
    s16(0) <= s15;
    s16(1) <= s13;
    s16(2) <= s11;
    s16(3) <= s9;
    s16(4) <= s7;
    s16(5) <= s5;
    s16(6) <= s3;
    s16(7) <= s1;
    gate8: entity work.MUX_GATE_BUS_1
        generic map (
            Bits => 8)
        port map (
            sel => Rst,
            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

LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.all;

entity main is
    port (
        N: in std_logic_vector(3 downto 0);
        data: in std_logic_vector(3 downto 0);
        write: in std_logic;
        addr: in std_logic_vector(3 downto 0);
        forceRst: in std_logic;
        clock: in std_logic;
        Result: out std_logic_vector(7 downto 0));
end main;

architecture Structural of main is
    signal clk: std_logic;
    signal rst: std_logic;
    signal s0: std_logic_vector(3 downto 0);
    signal s1: std_logic_vector(3 downto 0);
    signal s2: std_logic_vector(3 downto 0);
    signal s3: std_logic;
    signal s4: std_logic_vector(3 downto 0);
    signal s5: std_logic_vector(3 downto 0);
    signal s6: std_logic;
    signal s7: std_logic_vector(3 downto 0);
    signal str: std_logic;
begin
    gate0: entity work.DEMUX_GATE_1
        generic map (
            Default => 0)
        port map (
            sel => write,
            p_in => clock,
            out_0 => clk,
            out_1 => str);
    gate1: entity work.DIG_Counter
        generic map (
            Bits => 4)
        port map (
            en => '1',
            C => clk,
            clr => rst,
            p_out => s0);
    gate2: entity work.RhCount
        port map (
            clk => clk,
            C => s3,
            Rst => rst,
            Rh => s5);
    gate3: entity work.buffer4bit
        port map (
            Rst => rst,
            clk => clk,
            Din => s4,
            Dout => s2);
    gate4: entity work.buffer8bit
        port map (
            Rh => s5,
            Rl => s4,
            Rst => rst,
            clk => clk,
            Result => Result);
    gate5: entity work.isMax
        port map (
            N => N,
            Addr => s0,
            rst => s6);
    gate6: entity work.EEPROM
        port map (
            addr => s7,
            Din => data,
            Rst => rst,
            WE => write,
            clk => str,
            Dout => s1);
    gate7: entity work.MUX_GATE_BUS_1
        generic map (
            Bits => 4)
        port map (
            sel => write,
            in_0 => s0,
            in_1 => addr,
            p_out => s7);
    rst <= (forceRst OR s6);
    gate8: entity work.adder4bit
        port map (
            A => s1,
            B => s2,
            C => s3,
            Rl => s4);
end Structural;
```