CSU22022 Computer Architecture I

Seventh Lecture - Instantiations

Michael Manzke

2023-2024

Trinity College Dublin

Full Adder - First Example

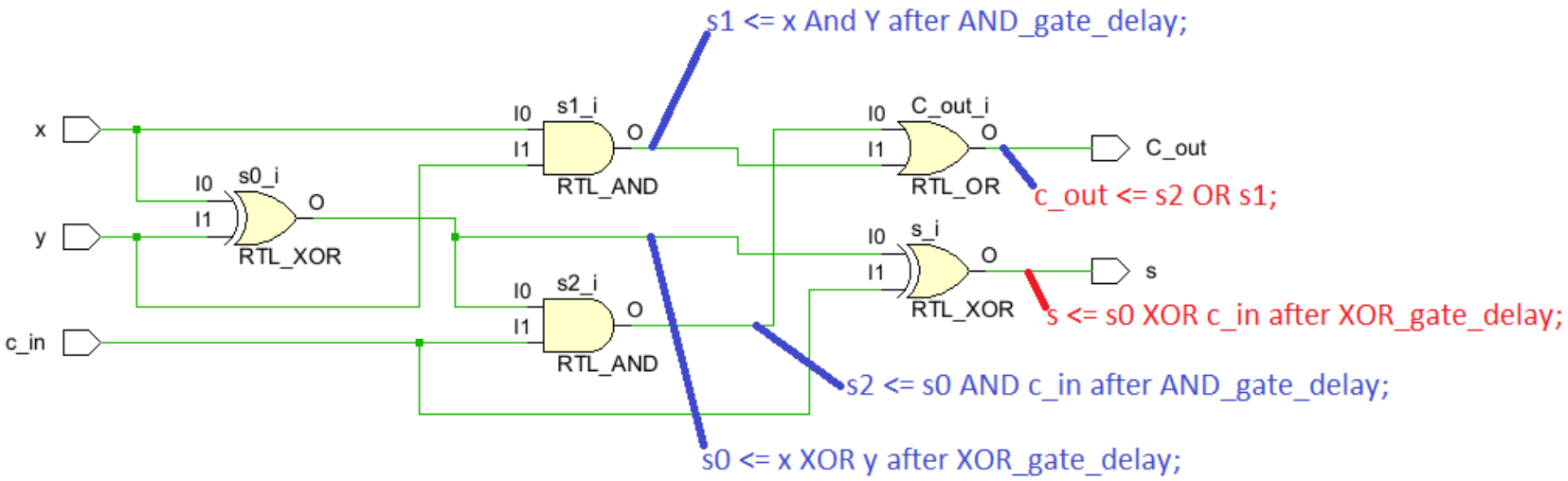

Figure 1: Implementation of a Full Adder with Two Half Adders and an OR Gate

Full Adder VHDL Code - One

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity Full_Adder is
    Port ( x : in STD_LOGIC;
           y : in STD_LOGIC;
           c_in : in STD_LOGIC;
           s : out STD_LOGIC;
           C_out : out STD_LOGIC);
end Full_Adder;
```

Listing 1: entity Full_Adder

Full Adder VHDL Code - Two

```
architecture Behavioral of Full_Adder is
    signal s0, s1, s2 : std_logic;
    --  Propagation Delay according to StdentID e.g. 26 33 57 25(DEC)
    constant AND_gate_delay : Time := 6ns; -- least significant digit 6=5+1
    constant NAND_gate_delay : Time := 3ns;-- next more significant digit 3=2+1
    constant OR_gate_delay : Time := 8ns;  -- next more significant digit 8=7+1
    constant NOR_gate_delay : Time := 6ns; -- next more significant digit 6=5+1
    constant XOR_gate_delay : Time := 4ns; -- next more significant digit 4=3+1
    constant XNOR_gate_delay : Time := 4ns;-- next more significant digit 4=3+1
    constant NOT_gate_delay : Time := 7ns; -- next more significant digit 7=6+1
```

Listing 2: signals and constants

Full Adder VHDL Code - Three

```
begin

    s0 <= x XOR y after XOR_gate_delay;
    s1 <= x And Y after AND_gate_delay;
    s2 <= s0 AND c_in after AND_gate_delay;
    s <= s0 XOR c_in after XOR_gate_delay;
    c_out <= s2 OR s1;

end Behavioral;
```

Listing 3: Concurrent signal assignment statements

8-bit Ripple Adder

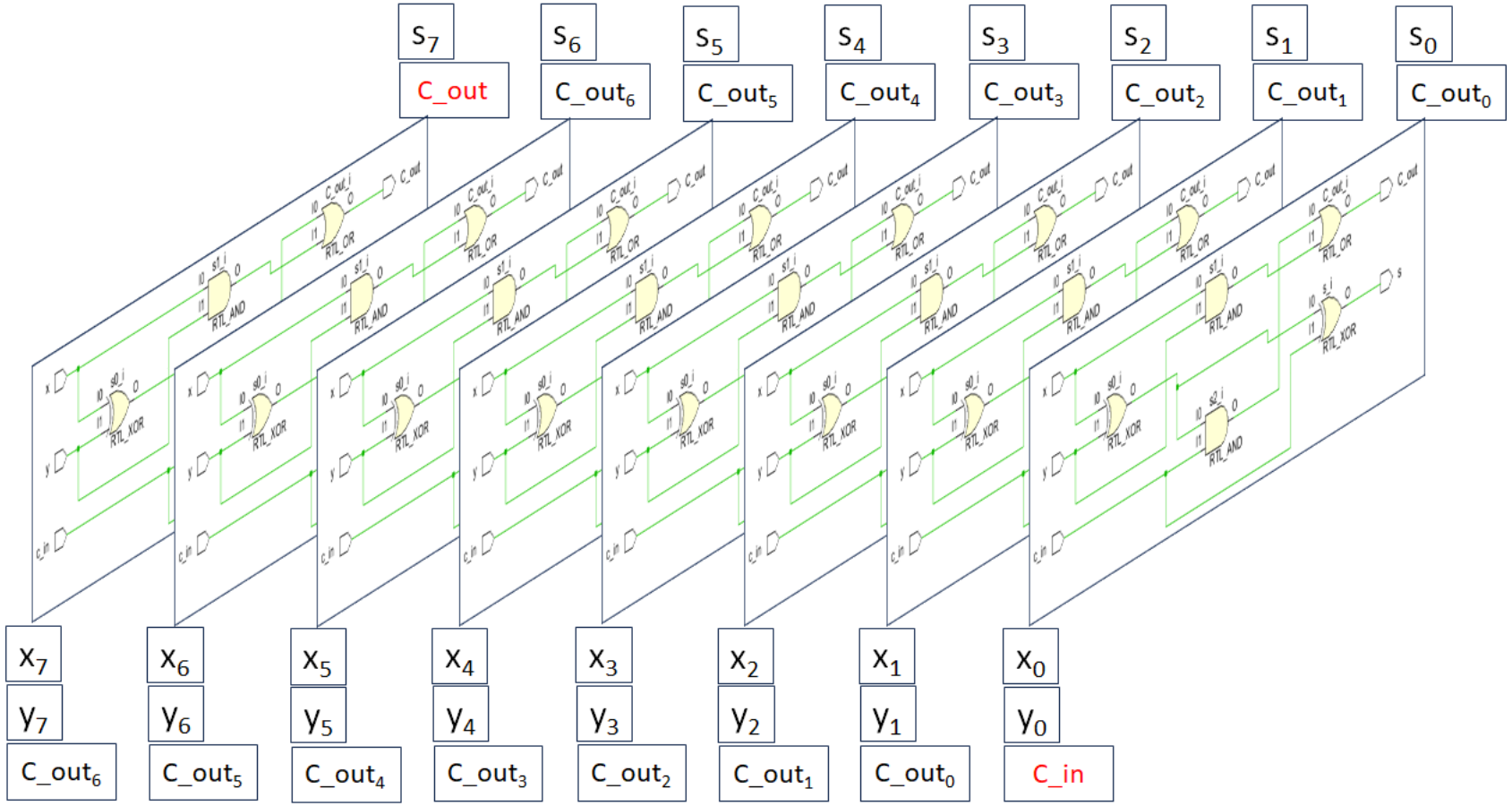

Figure 2: c_out from the less significant bit becomes the c_in for the next more significant bit.

Full 8-bit Ripple Adder VHDL Code - One

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity RippleCarryAdder8_bit is
    Port ( x, y : in STD_LOGIC_VECTOR (7 downto 0);
           c_rca_in : in STD_LOGIC;
           s : out STD_LOGIC_VECTOR (7 downto 0);
           c_rca_out : out STD_LOGIC);
end RippleCarryAdder8_bit;
```

Listing 4: entity Full_Adder

Full 8-bit Ripple Adder VHDL Code - Two

```
architecture Behavioral of RippleCarryAdder8_bit is

    COMPONENT Full_Adder
    Port ( x : in STD_LOGIC;
           y : in STD_LOGIC;
           c_in : in STD_LOGIC;
           s : out STD_LOGIC;
           c_out : out STD_LOGIC);
    END COMPONENT;
```

Listing 5: COMPONENT declaration

Full 8-bit Ripple Adder VHDL Code - Three

```
signal c_out0, c_out1, c_out2, c_out3 : std_logic;
signal c_out4, c_out5, c_out6 : std_logic;

--  Propagation Delay according to StdentID e.g. 26 33 57 25(DEC)
constant AND_gate_delay : Time := 6ns; -- least significant digit 6 =5+1
constant NAND_gate_delay : Time := 3ns;-- next more significant digit 3=2+1
constant OR_gate_delay : Time := 8ns;  -- next more significant digit 8=7+1
constant NOR_gate_delay : Time := 6ns; -- next more significant digit 6=5+1
constant XOR_gate_delay : Time := 4ns; -- next more significant digit 4=3+1
constant XNOR_gate_delay : Time := 4ns;-- next more significant digit 4=3+1
constant NOT_gate_delay : Time := 7ns; -- next more significant digit 7=6+1
```

Listing 6: signals and constants

Full 8-bit Ripple Adder VHDL Code - Four

```
-- Instantiate the least significant bit
bit0: Full_Adder PORT MAP (
        x => x(0),
        y => y(0),
        c_in => c_rca_in,
        s => s(0),
        c_out => c_out0
      );

bit1: Full_Adder PORT MAP (
        x => x(1),
        y => y(1),
        c_in => c_out0,
        s => s(1),
        c_out => c_out1
      );
```

Listing 7: Instantiate Full_Adders for bit0 and bit1

Full 8-bit Ripple Adder VHDL Code - Five

```
-- Instantiate the least significant bit
bit2: Full_Adder PORT MAP (
        x => x(2),
        y => y(2),
        c_in => c_out1,
        s => s(2),
        c_out => c_out2
      );

bit3: Full_Adder PORT MAP (
        x => x(3),
        y => y(3),
        c_in => c_out2,
        s => s(3),
        c_out => c_out3
      );
```

Listing 8: Instantiate Full_Adders for bit2 and bit3

Full 8-bit Ripple Adder VHDL Code - Six

```
-- Instantiate the least significant bit
bit4: Full_Adder PORT MAP (
        x => x(4),
        y => y(4),
        c_in => c_out3,
        s => s(4),
        c_out => c_out4
      );

bit5: Full_Adder PORT MAP (
        x => x(5),
        y => y(5),
        c_in => c_out4,
        s => s(5),
        c_out => c_out5
      );
```

Listing 9: Instantiate Full_Adders for bit4 and bit5

Full 8-bit Ripple Adder VHDL Code - Seven

```
-- Instantiate the least significant bit
bit5: Full_Adder PORT MAP (
        x => x(5),
        y => y(5),
        c_in => c_out4,
        s => s(5),
        c_out => c_out5
      );

bit6: Full_Adder PORT MAP (
        x => x(6),
        y => y(6),
        c_in => c_out5,
        s => s(5),
        c_out => c_out6
      );
```

Listing 10: Instantiate Full_Adders for bit6 and bit7

8-bit Ripple Adder Sources Panel

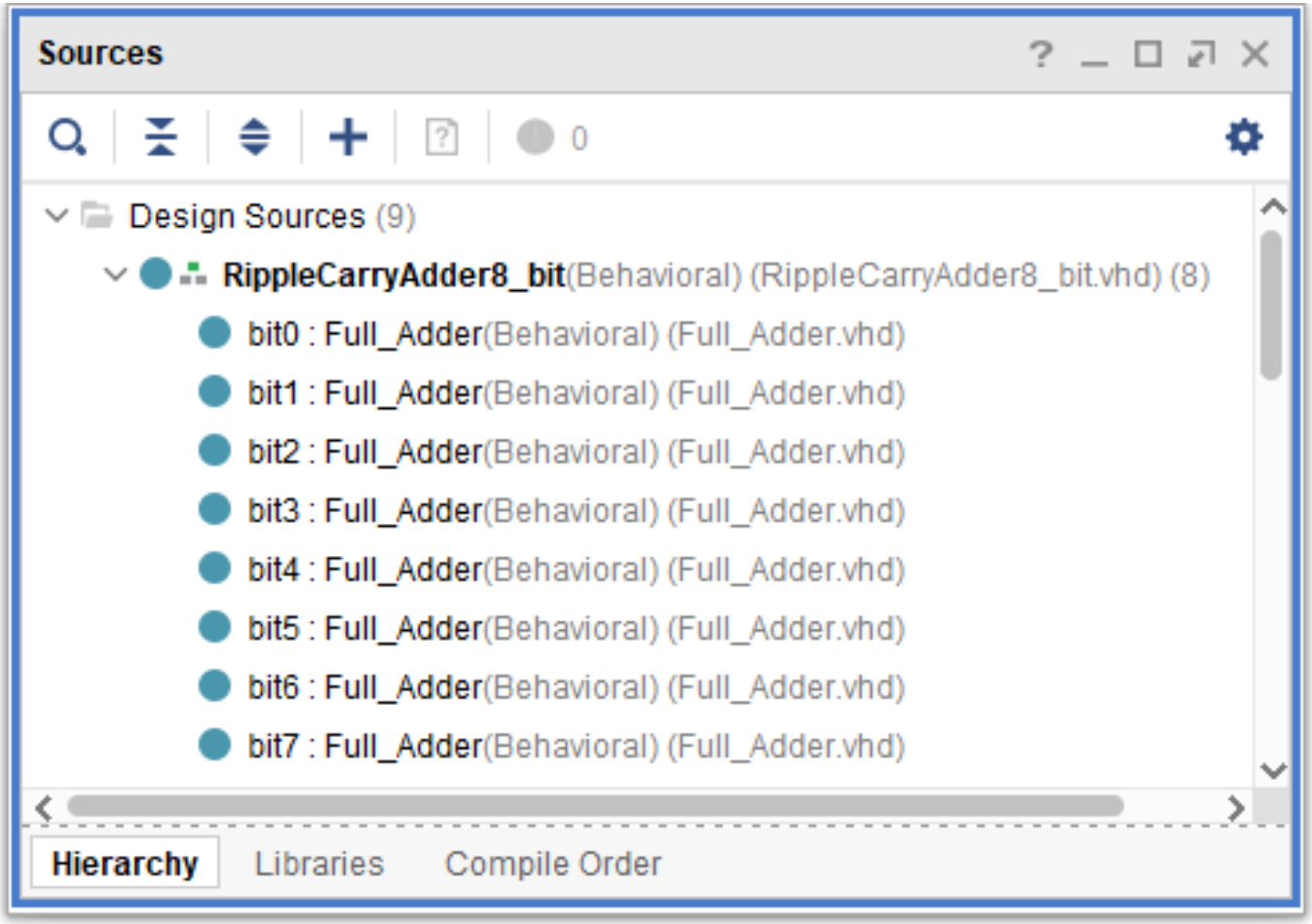

Figure 3: Labels bit0 to bit7 allow us to identify the instantiations of the entity Full_Adder.

8-bit Ripple Adder Schematic

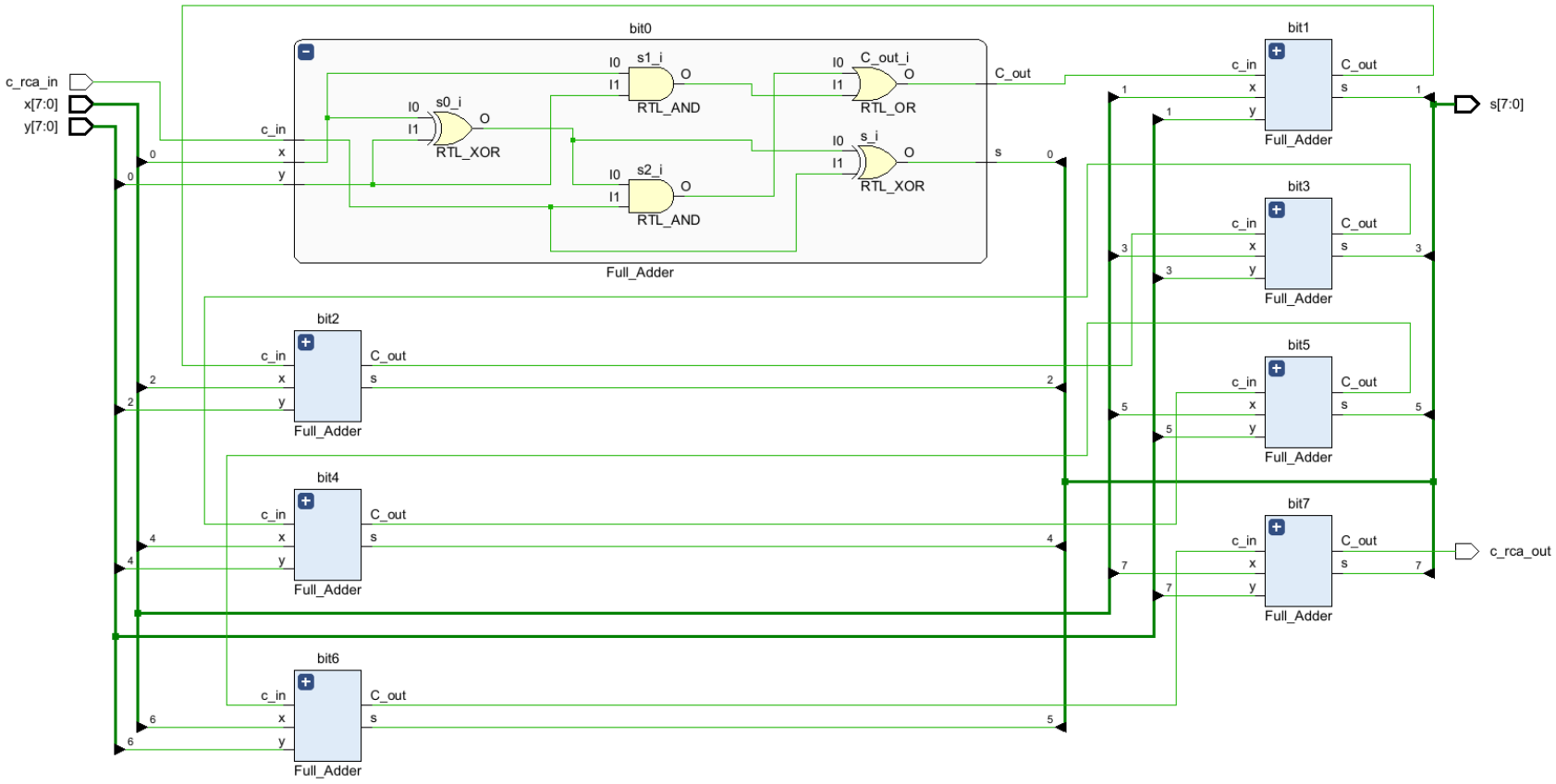

Figure 4: Schematic with detail for the least significant bit.

8-to-1-Line Multiplexer - Second Example

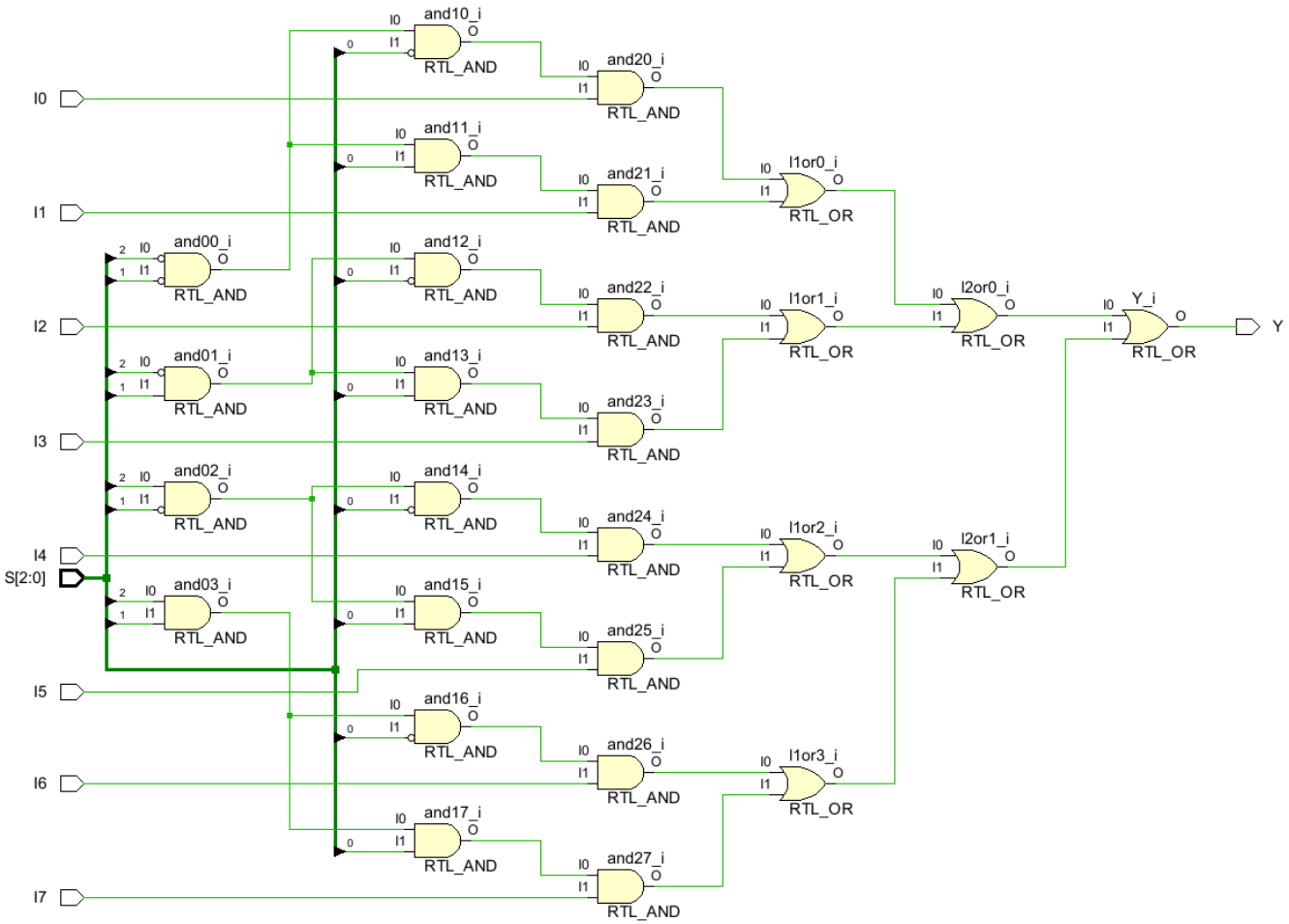

Figure 5: See the **Fifth Lecture** for VHDL code.

8-bit 8-to-1-Line Multiplexer

Figure 6: S[2:0] goes to all 8-to-1-Line Multiplexer to select the input vector.

8-bit 8-to-1-Line Multiplexer VHDL Code - One

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity Mux_8bit_8_to_1 is
  Port ( I0, I1, I2, I3 : in STD_LOGIC_VECTOR (7 downto 0);
         I4, I5, I6, I7 : in STD_LOGIC_VECTOR (7 downto 0);
         S : in STD_LOGIC_VECTOR (2 downto 0);
         y : out STD_LOGIC_VECTOR (7 downto 0));
end Mux_8bit_8_to_1;
```

Listing 11: entity Mux_8bit_8_to_1

8-bit 8-to-1-Line Multiplexer VHDL Code - Two

```
architecture Behavioral of Mux_8bit_8_to_1 is

    COMPONENT Mux_8_to_1
    Port ( I0, I1, I2, I3 : in STD_LOGIC;
           I4, I5, I6, I7 : in STD_LOGIC;
           S : in STD_LOGIC_VECTOR (2 downto 0);
           Y : out STD_LOGIC);
    END COMPONENT;
```

Listing 12: COMPONENT declaration

8-bit 8-to-1-Line Multiplexer VHDL Code - Three

```

    --  Propagation Delay according to StdentID e.g. 26 33 57 25(DEC)
    constant AND_gate_delay : Time := 6ns; -- least significant digit 6 =5+1
    constant NAND_gate_delay : Time := 3ns;-- next more significant digit 3=2+1
    constant OR_gate_delay : Time := 8ns;  -- next more significant digit 8=7+1
    constant NOR_gate_delay : Time := 6ns; -- next more significant digit 6=5+1
    constant XOR_gate_delay : Time := 4ns; -- next more significant digit 4=3+1
    constant XNOR_gate_delay : Time := 4ns;-- next more significant digit 4=3+1
    constant NOT_gate_delay : Time := 7ns; -- next more significant digit 7=6+1
```

Listing 13: constants

8-bit 8-to-1-Line Multiplexer VHDL Code - Four

```
begin

   -- Instantiate the least significant bit
   bit0: Mux_8_to_1 PORT MAP (
       I0 => I0(0), I1 => I1(0), I2 => I2(0), I3 => I3(0),
       I4 => I4(0), I5 => I5(0), I6 => I6(0), I7 => I7(0),
       S => S, Y => y(0));

   bit1: Mux_8_to_1 PORT MAP (
       I0 => I0(1), I1 => I1(1), I2 => I2(1), I3 => I3(1),
       I4 => I4(1), I5 => I5(1), I6 => I6(1), I7 => I7(1),
       S => S, Y => y(1));
```

Listing 14: Instantiate Mux_8_to_1 for bit0 and bit1

8-bit 8-to-1-Line Multiplexer VHDL Code - Five

```
bit2: Mux_8_to_1 PORT MAP (
    I0 => I0(2), I1 => I1(2), I2 => I2(2), I3 => I3(2),
    I4 => I4(2), I5 => I5(2), I6 => I6(2), I7 => I7(2),
    S => S, Y => y(2));

bit3: Mux_8_to_1 PORT MAP (
    I0 => I0(3), I1 => I1(3), I2 => I2(3), I3 => I3(3),
    I4 => I4(3), I5 => I5(3), I6 => I6(3), I7 => I7(3),
    S => S, Y => y(3));

bit4: Mux_8_to_1 PORT MAP (
    I0 => I0(4), I1 => I1(4), I2 => I2(4), I3 => I3(4),
    I4 => I4(4), I5 => I5(4), I6 => I6(4), I7 => I7(4),
    S => S, Y => y(4));
```

Listing 15: Instantiate Mux_8_to_1 for bit2 bit3 and bit4

8-bit 8-to-1-Line Multiplexer VHDL Code - Six

```
bit5: Mux_8_to_1 PORT MAP (
    I0 => I0(5), I1 => I1(5), I2 => I2(5), I3 => I3(5),
    I4 => I4(5), I5 => I5(5), I6 => I6(5), I7 => I7(5),
    S => S, Y => y(5));

bit6: Mux_8_to_1 PORT MAP (
    I0 => I0(6), I1 => I1(6), I2 => I2(6), I3 => I3(6),
    I4 => I4(6), I5 => I5(6), I6 => I6(6), I7 => I7(6),
    S => S, Y => y(6));

bit7: Mux_8_to_1 PORT MAP (
    I0 => I0(7), I1 => I1(7), I2 => I2(7), I3 => I3(7),
    I4 => I4(7), I5 => I5(7), I6 => I6(7), I7 => I7(7),
    S => S, Y => y(7));
```

Listing 16: Instantiate Mux_8_to_1 for bit5 bit6 and bit7

8-bit 8-to-1-Line Multiplexer Schematic

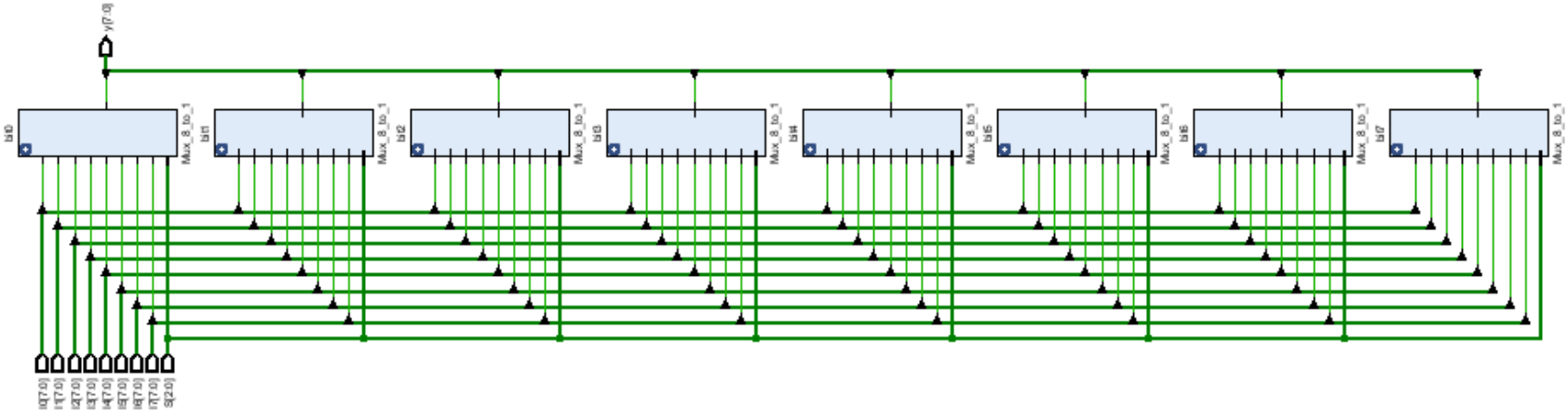

Figure 7: S[2:0] goes to all 8-to-1-Line Multiplexer to select the input vector.

VHDL Conditional Signal Assignment Statements - One

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity Mux_8bit_8_1_ConSigASS is
 Port ( I0, I1, I2, I3 : in STD_LOGIC_VECTOR (7 downto 0);
        I4, I5, I6, I7 : in STD_LOGIC_VECTOR (7 downto 0);
        S : in STD_LOGIC_VECTOR (2 downto 0);
        Y : out STD_LOGIC_VECTOR (7 downto 0));
end Mux_8bit_8_1_ConSigASS;
```

Listing 17: The port declaration is identical to concurrent assignment statement implementation.

VHDL Conditional Signal Assignment Statements - Two

```
1
2  architecture Behavioral of Mux_8bit_8_1_ConSigASS is
3
4  begin
5  Y<=I0 after 15ns when s(2)='0'and S(1)='0'and S(0)='0' else
6     I1 after 15ns when s(2)='0'and S(1)='0'and S(0)='1' else
7     I2 after 15ns when s(2)='0'and S(1)='1'and S(0)='0' else
8     I3 after 15ns when s(2)='0'and S(1)='1'and S(0)='1' else
9     I4 after 15ns when s(2)='1'and S(1)='0'and S(0)='0' else
10    I5 after 15ns when s(2)='1'and S(1)='0'and S(0)='1' else
11    I6 after 15ns when s(2)='1'and S(1)='1'and S(0)='0' else
12    I7 after 15ns when s(2)='1'and S(1)='1'and S(0)='1' else
13    "00000000";
14 end Behavioral;
```

Listing 18: These statements make the design easier but we will not use them because they abstract the under underlying hardware architecture.

VHDL Conditional Signal Assignment Statements

Figure 8:

Positive-Edge-Triggered D Flip-Flop - Third Example

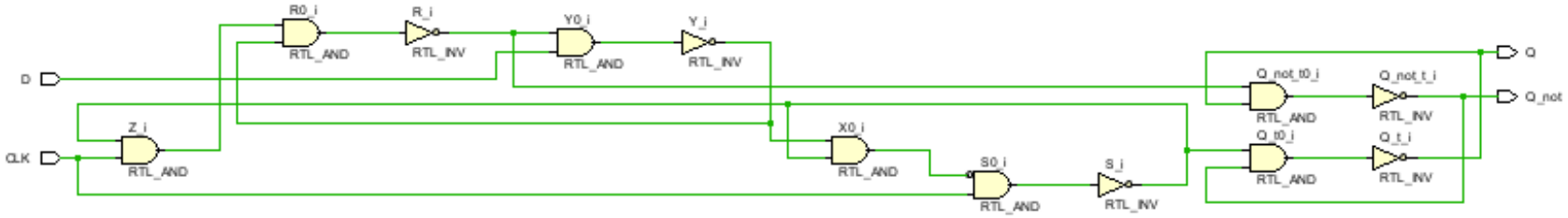

Figure 9: See the **Sixth Lecture** for VHDL code.

8-bit Register

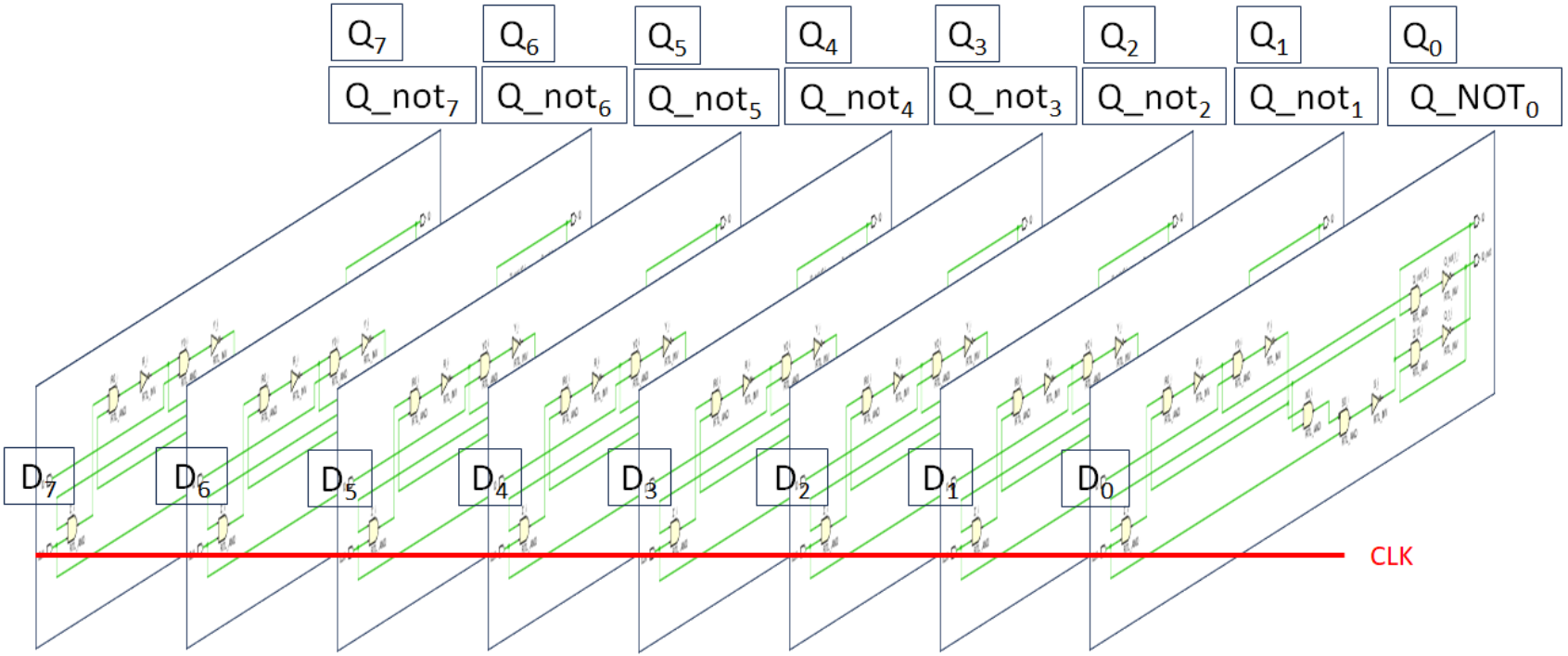

Figure 10: Implementation of a 8-bit Register

Positive-Edge-Triggered D Flip-Flop - Process

```
entity D_Flip_Flop_PEdge_Process is
    Port ( CLK, D : in STD_LOGIC;
           Q, Q_not : out STD_LOGIC);
end D_Flip_Flop_PEdge_Process;
architecture Behavioral of D_Flip_Flop_PEdge_Process is
    signal state : std_logic;
    constant NOT_gate_delay : Time := 7ns;
begin
    Q <= state;
    Q_not <= not state after NOT_gate_delay;
    process(CLK)
    begin
    if (CLK'event and CLK='1') then
        state <= D;
    end if;
    end process;
end Behavioral;
```

Listing 19: A process makes the design easier but we will not use them because they abstract the hardware architecture.

VHDL Process

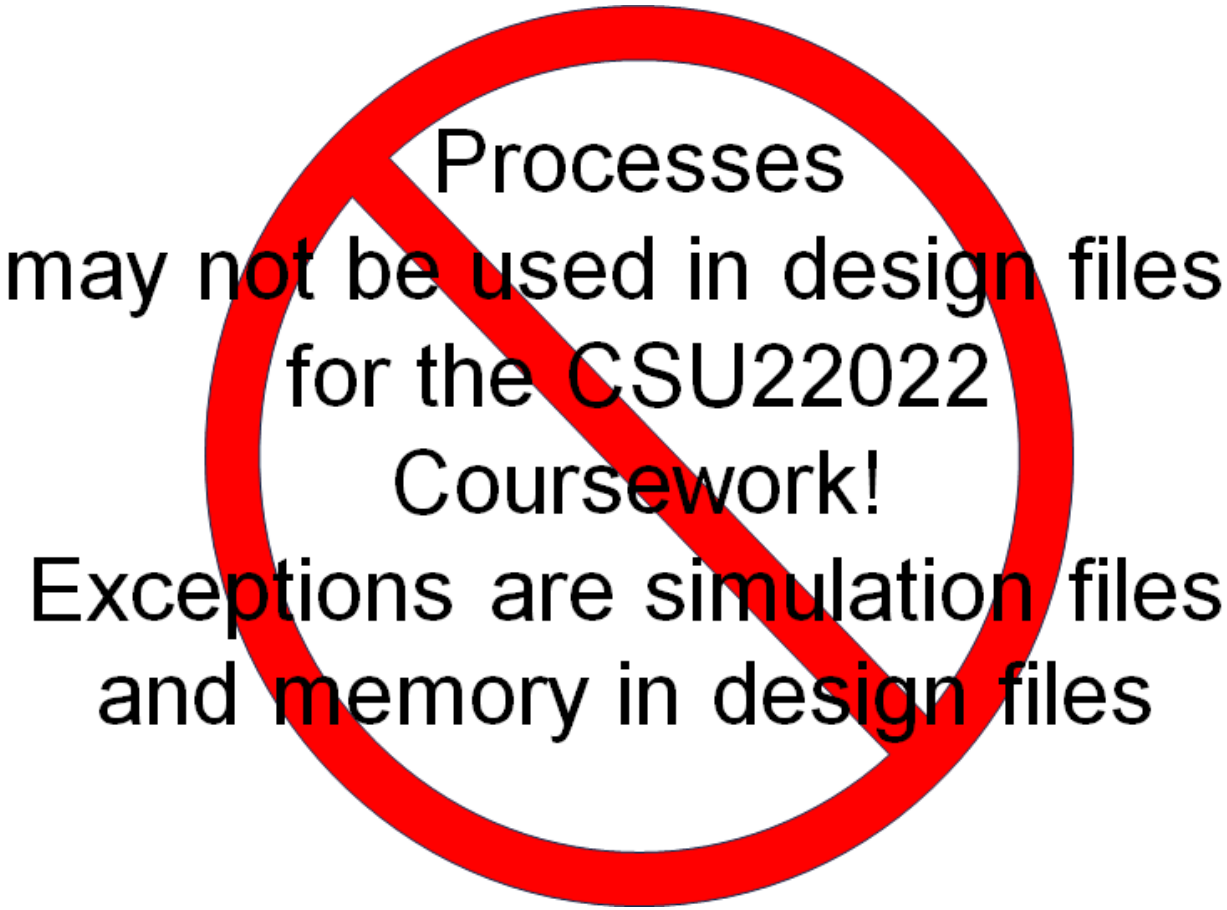

Figure 11: