HDL Week 3

Jochem Arends (495637)

Exercises Chapter 6

1. *Design and simulate a behavioral description of a 4-to-1 multiplexer.*

   I've pasted the simulation result at the end of this document. It might be difficult to see, but the multiplexer generates the correct output signal.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity mux_behavioral is
    port (
        input:   in  std_logic_vector(3 downto 0);
        control: in  std_logic_vector(1 downto 0);
        output:  out std_Logic
    );
end mux_behavioral;

architecture arch of mux_behavioral is
begin
    process (input, control)
    begin
        case control is
            when "00" => output <= input(0);
            when "01" => output <= input(1);
            when "10" => output <= input(2);
            when "11" => output <= input(3);
        end case;
    end process;
end arch;
```

2. *Implement the 4-to-1 multiplexer on the max 10 board.*

   TODO

3. *Design and simulate a structural description of a 1-to-4 demultiplexer.*

   I've pasted the simulation result at the end of this document.

```vhdl
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity demux is
    port (
        input:   in  std_logic;
        control: in  std_logic_vector(1 downto 0);
        output:  out std_logic_vector(3 downto 0)
    );
end demux;

architecture arch of demux is
    component and_gate is
        generic (
            N: natural := 3
        );

        port (
            input:  in  std_logic_vector(N - 1 downto 0);
            output: out std_logic
        );
    end component;

    component not_gate is
        port (
            input:  in  std_logic;
            output: out std_logic
        );
    end component;

    signal inv_control: std_logic_vector(1 downto 0);
begin
    -- for the inverse control
    u1: not_gate port map(control(0), inv_control(0));
    u2: not_gate port map(control(1), inv_control(1));

    u3: and_gate port map(inv_control                    & input, output(0));
    u4: and_gate port map(inv_control(1) &     control(0) & input, output(1));
    u5: and_gate port map(    control(1) & inv_control(0) & input, output(2));
    u6: and_gate port map(    control(1) &     control(0) & input, output(3));
end arch;
```

4. *Connect the two description in a block diagram and give a good simulation.*

   TODO

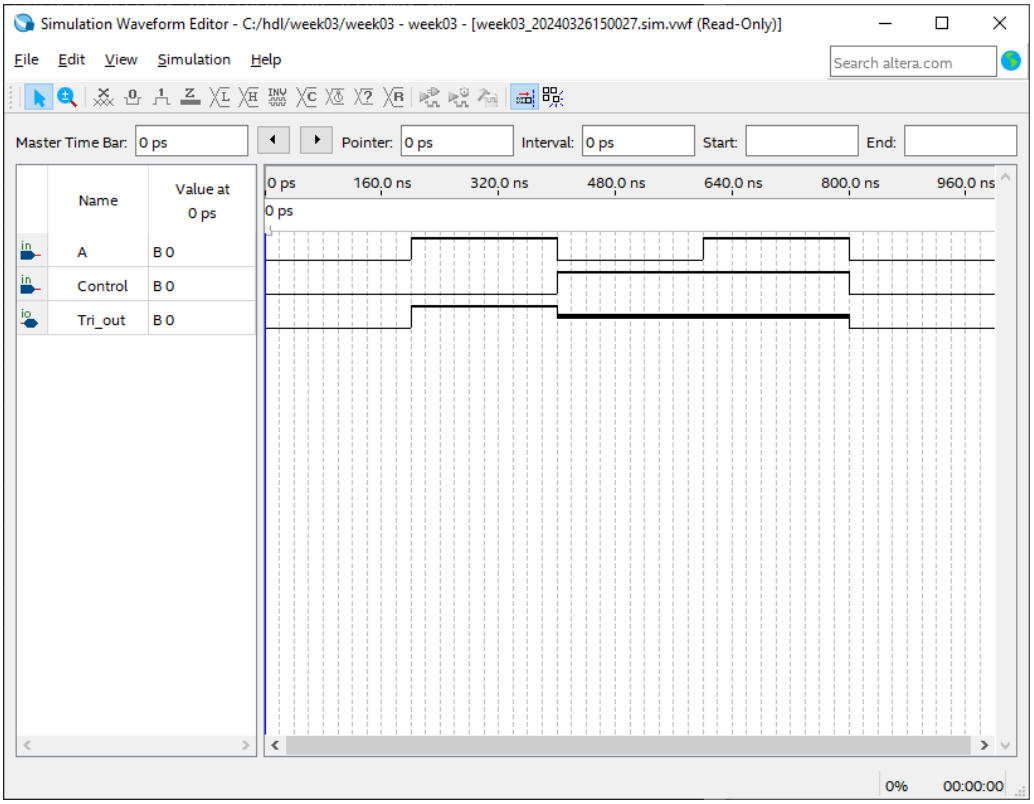
Self Test Chapter 7

1. *When is a tri-state needed?*

   A tri-state buffer may be needed when connecting multiple signals to a commmon bus.

Exercises Chapter 7

1. *Simulate in Quartus the tristate model given in this chapter.*



2. *Design and simulate an 8-bit tri-state buffer.*

```
use library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity tristate is
    generic (
        N: natural := 1
    );

    port (
        input:   in  std_logic_vector(N - 1 downto 0);
        control: in  std_logic_vector(N - 1 downto 0);
        output:  out std_logic_vector(N - 1 downto 0);
    );
end tristate;

architecture arch of tristate is
begin
    for i in input'range loop
        output(i) <= a when control(i) = '0' else 'z';
    end loop;
end arch;

tristate8: tristate generic map(N => 8)
    port map (
        input   => input,
        control => control,
        output  => output
    )
```

Self Test Chapter 8

1. *Why is it not advisable to gate a signal with the clock?*

   Because the clock signal might not be in a stable state. An example of this is during a rising or falling edge.

2. *Describe the difference working of the two next descriptions of a flip flop.*

   First Description

   ```
   process (reset, clock)
   begin
       if reset = '1' then
           q <= '0';
       elsif clock'EVENT and clock = '1' then
           q <= data;
       end if;
   end process;
   ```

Second Description

```
process (clock)
begin
    if reset = '1' then
        q <= '0';
    elsif clock'EVENT and clock = '1' then
        q <= data;
    end if;
end process;
```

The first description has both `reset` and `clock` in its priority list while the second description only has `clock` in its priority list. For the first description, the process statement gets evaluated when `reset` or `clock` changes. This flip flop uses an asynchronous reset. For the second description, the process statement only gets evaluated when `clock` changes. This flip flop resets when the `reset` signal is high when `clock` changes.

Exercises Chapter 8

1. *Write VHDL for an 8-bit D-flipflop register (with a negative).*

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity reg is
    generic (
        N: positive := 8
    );

    port (
        D:          in  std_logic_vector(N - 1 downto 0);
        CLK, CLEAR: in  std_logic;
        Q:          out std_logic_vector(N - 1 downto 0)
    );
end reg;

architecture arch of reg is
begin
    process (CLK)
    begin
        if falling_edge(CLK) then
            if CLEAR = '1' then
                Q <= (others => '0');
            else
                Q <= D;
            end if;
        end if;
    end process;
end arch;
```

Simulation Results

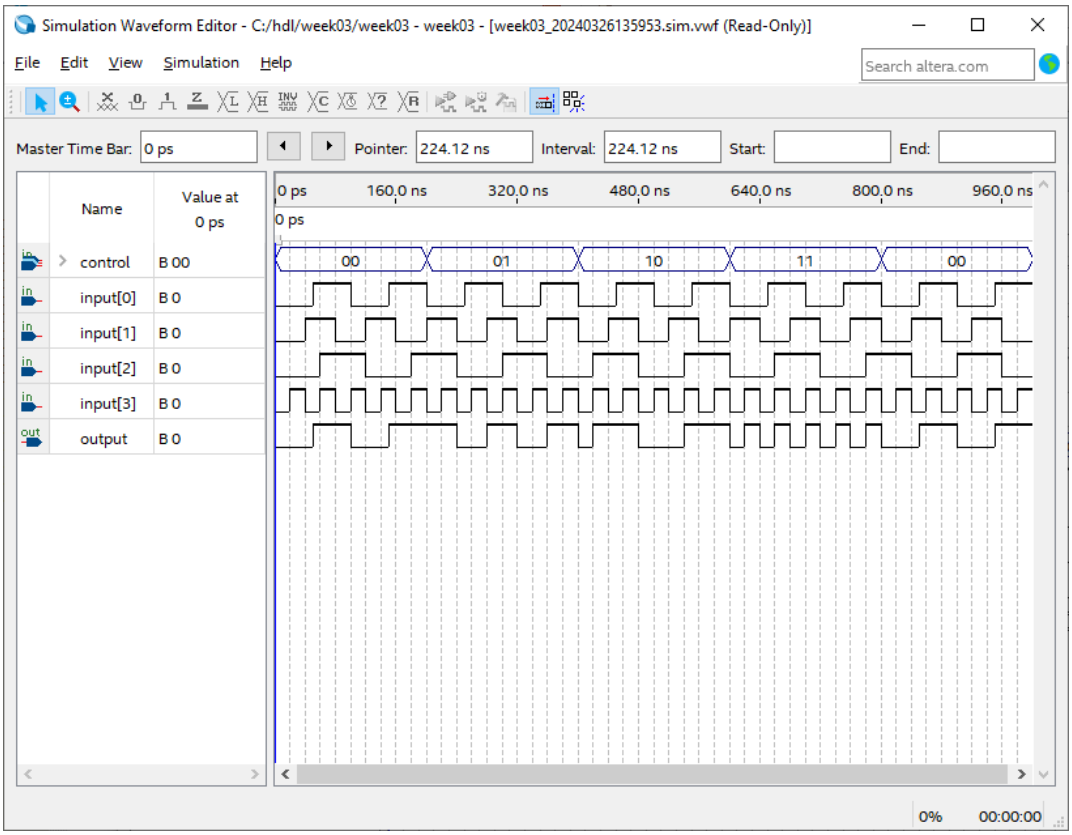


Figure 1. Simulation of Behavioral 4-to-1 Multiplexer

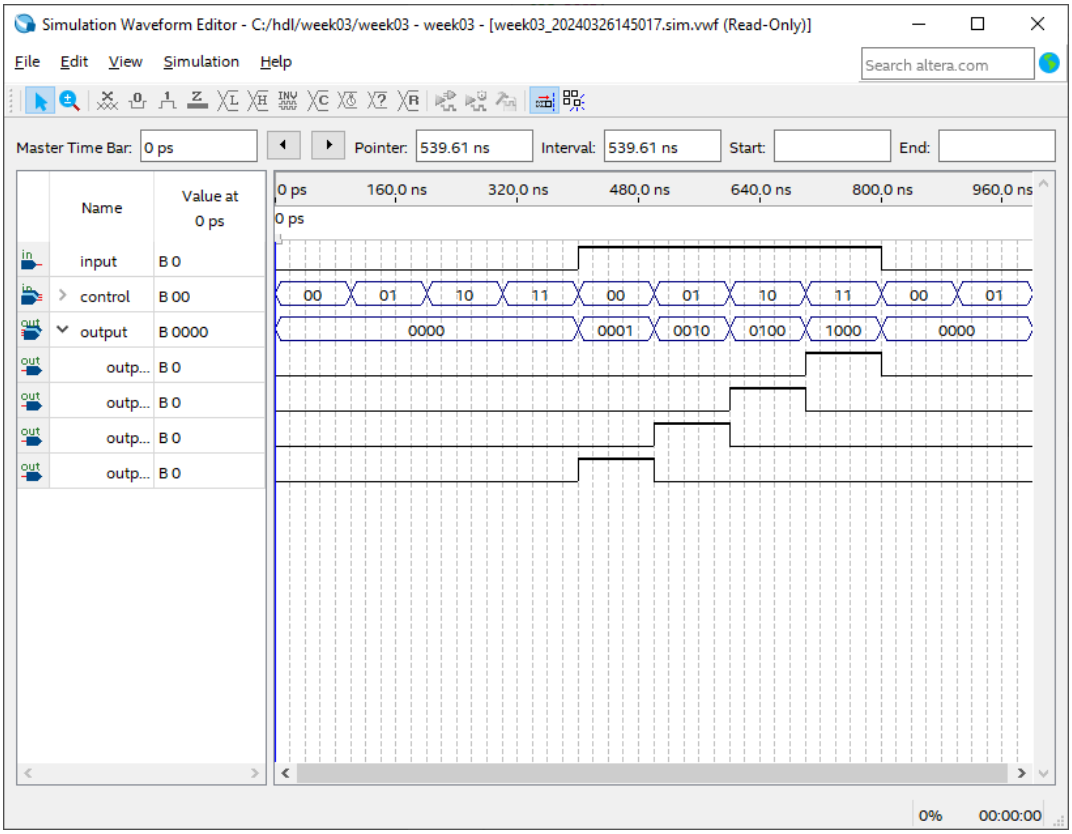


Figure 2. Simulation of Structural 1-to-4 Demultiplexer