Description

The four-bit register was declared as an entity with five input ports (four data signals and one clock signal) and four output ports (four output signals). It was implemented with a process using the algorithmic behavioral model. Four internal signals are declared as "storage." In the process "if" statements are used to check whether the clock is on a falling edge (CLK'event and CLK = '0') or on a rising edge (CLK'event and CLK = '1'). Therefore the only signal in the sensitivity list is the CLK input. On the falling edge event of the clock each position in the input data vector is put into the corresponding position in the storage vector. On the rising edge event of the clock the storage data is put into the output vector.

The four-bit tri-state buffer was implemented in a very similar way to the register. Instead of the clock input, there is an "enable" input. No internal signals are needed in the buffer. In the process, if the enable is equal to '1' the input vector is sent directly to the output vector. Otherwise the output vector will be set to 'Z.'

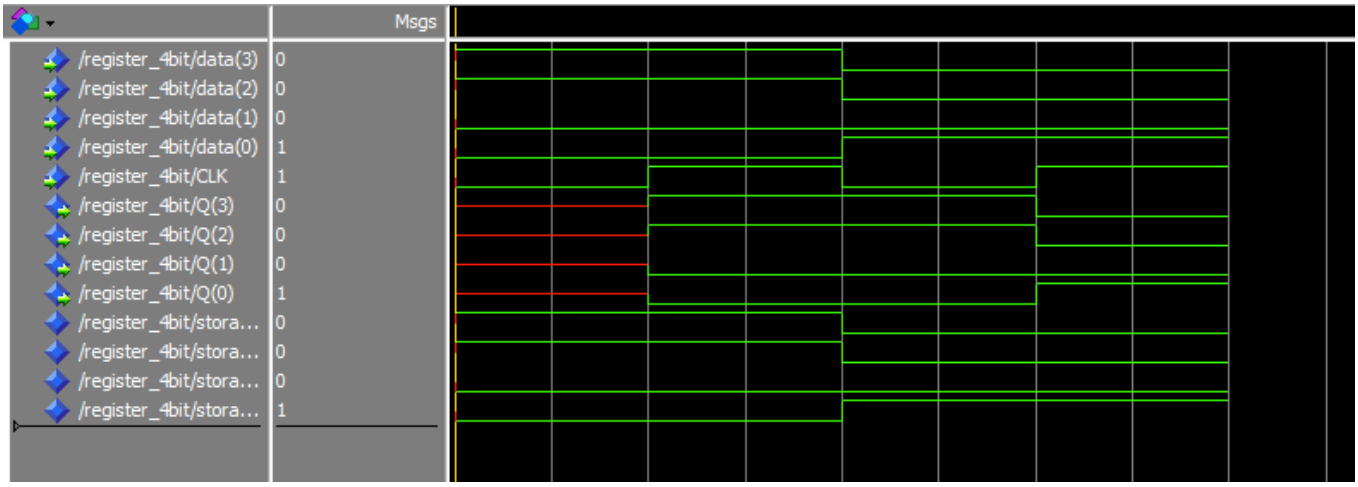
VDHL Code

```
--Four-bit Register
library ieee;
use ieee.std_logic_1164.all;

entity register_4bit is
	port(data : in std_logic_vector (3 downto 0);
		CLK : in std_logic;
		Q : out std_logic_vector (3 downto 0));
end register_4bit;

architecture behav of register_4bit is
	signal storage : std_logic_vector (3 downto 0);
begin
	reg_process: process(CLK)
	begin
		if (CLK'event and CLK = '0') then
			storage(0) <= data(0);
			storage(1) <= data(1);
```

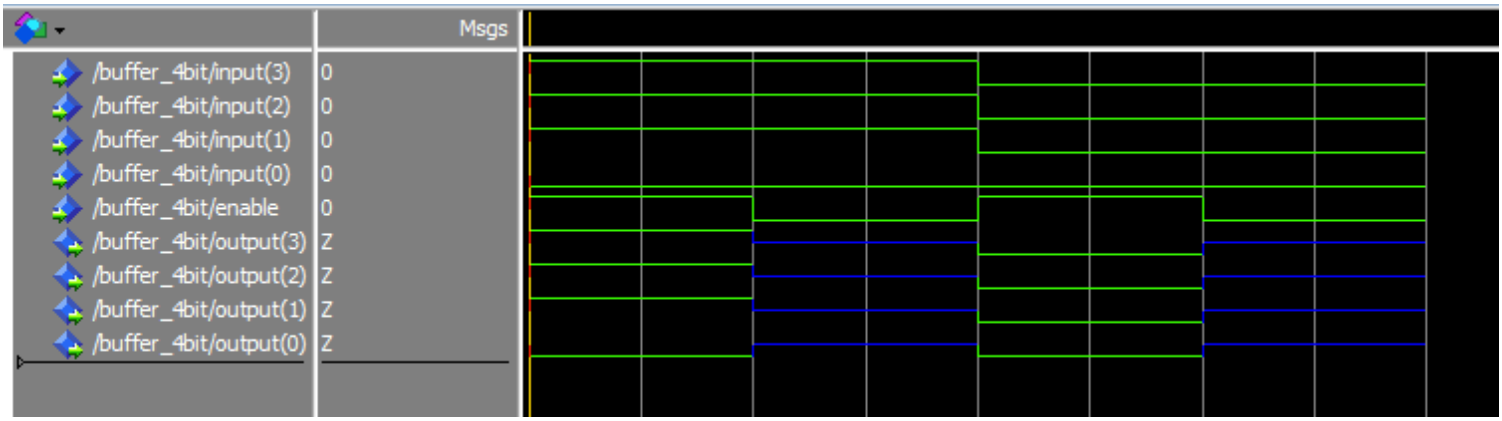
```
                storage(2) <= data(2);
                storage(3) <= data(3);
        elsif (CLK'event and CLK = '1') then
                Q(0) <= storage(0);
                Q(1) <= storage(1);
                Q(2) <= storage(2);
                Q(3) <= storage(3);
        end if;
    end process;
end behav;

--Four-bit Tri-state Buffer
library ieee;
use ieee.std_logic_1164.all;

entity buffer_4bit is
    port(input : in std_logic_vector (3 downto 0);
        enable : in std_logic;
        output : out std_logic_vector (3 downto 0));
end buffer_4bit;

architecture behav of buffer_4bit is
begin
    buffer_process: process(enable)
    begin
        if (enable = '1') then
        output(0) <= input(0);
        output(1) <= input(1);
        output(2) <= input(2);
        output(3) <= input(3);
    else
        output(0) <= 'Z';
        output(1) <= 'Z';
        output(2) <= 'Z';
            output(3) <= 'Z';
        end if;
    end process;
end behav;
```

Simulation

- Four-bit Register



- Four-bit Tri-state Buffer



Analysis

The four-bit register was initially loaded with 1100 (clock set to 0, the input vector values put into the storage vector). Then the clock was set to 1 and the same value was put into output. In the next cycle the clock was set back to 0, the input loaded with 0001, and the storage vector set equal to the inputs. On the next clock pulse the clock was set back to 1 and the output received 0001.

The four-bit tri-state buffer was initially loaded with 1110 and the enable set to 1. The output received 1110 on the same initial clock pulse. Following this the enable was changed to 0 and the outputs were all set to output nothing ('Z'). The next clock pulse took an enable of 1 and received a value of 0000 to the input. The output followed the input and also went to 0000. In the last clock pulse the enable is changed back to 0 and the output goes back to 'Z.'

Algorithmic modeling is a very intuitive way of modeling a circuit. The algorithms for a register and buffer are very simple. The sequential nature of a process allows the programmer to think in terms of a series of events happening instead of the relationship between many components. When there are only a few possible outcomes the algorithmic modeling is an efficient paradigm.

Do Scripts

- Four-bit Register

```
vsim -gui work.register_4bit
add wave  \
sim:/register_4bit/data(3) \
sim:/register_4bit/data(2) \
sim:/register_4bit/data(1) \
sim:/register_4bit/data(0) \
sim:/register_4bit/CLK
force -freeze sim:/register_4bit/data(3) 1 0
force -freeze sim:/register_4bit/data(2) 1 0
add wave  \
sim:/register_4bit/Q(3) \
sim:/register_4bit/Q(2) \
```

```
sim:/register_4bit/Q(1) \
sim:/register_4bit/Q(0)
force -freeze sim:/register_4bit/data(1) 0 0
force -freeze sim:/register_4bit/data(0) 0 0
add wave  \
sim:/register_4bit/storage(3) \
sim:/register_4bit/storage(2) \
sim:/register_4bit/storage(1) \
sim:/register_4bit/storage(0)
force -freeze sim:/register_4bit/CLK 0 0
run
force -freeze sim:/register_4bit/CLK 1 0
run
force -freeze sim:/register_4bit/data(3) 0 0
force -freeze sim:/register_4bit/data(2) 0 0
force -freeze sim:/register_4bit/data(0) 1 0
force -freeze sim:/register_4bit/CLK 0 0
run
force -freeze sim:/register_4bit/CLK 1 0
run
```

- Four-bit Tri-State Buffer

```
vsim -gui work.buffer_4bit
add wave  \
sim:/buffer_4bit/input(3) \
sim:/buffer_4bit/input(2) \
sim:/buffer_4bit/input(1) \
sim:/buffer_4bit/input(0) \
sim:/buffer_4bit/enable \
sim:/buffer_4bit/output(3) \
sim:/buffer_4bit/output(2) \
sim:/buffer_4bit/output(1) \
sim:/buffer_4bit/output(0)
force -freeze sim:/buffer_4bit/input(3) 1 0
force -freeze sim:/buffer_4bit/input(2) 1 0
force -freeze sim:/buffer_4bit/input(1) 1 0
force -freeze sim:/buffer_4bit/input(0) 0 0
force -freeze sim:/buffer_4bit/enable 1 0
```

```
run
force -freeze sim:/buffer_4bit/enable 0 0
run
force -freeze sim:/buffer_4bit/input(3) 0 0
force -freeze sim:/buffer_4bit/input(2) 0 0
force -freeze sim:/buffer_4bit/input(1) 0 0
force -freeze sim:/buffer_4bit/enable 1 0
run
force -freeze sim:/buffer_4bit/enable 0 0
run
```