RYERSON UNIVERSITY

Faculty of Engineering, Architecture and Science

Department of Electrical and Computer Engineering

Course Number	COE608
Course Title	Computer Organization and Architecture
Semester/Year	W2023

Instructor	Dr. Demetres Kostas

Report Title	Program Counter and Register Set Design

Submission Date	2023-01-25
Due Date	2023-01-30

Student Name	Student ID	Signature*
Den Carlo Dumlao	501018239	C.D

Table of Contents

1. 1-bit Register

a. Description

This register allows a 1-bit of data to be stored on the output Q. The state of the output can be changed based on the following:

- To store the value of the data d into this register ($Q = d$), the load/enable ld must be set to high, the clear clr needs to be disabled ($clr = 0$), and the clock clk must be at a rising edge. This can be observed in Figure 1 below at $t = 70ns$ of the register. Notice that when those conditions are met at that time instant, the output immediately transitions from 0 to the state of d which is 1.
- To reset the register such that $Q = 0$, the clear clr is needed to be enabled. From Figure 1, a reset is made at $t = 160ns$. Initially, the register holds a value of 1; but as soon as the clr is enabled, the output is immediately set to low.
- To maintain the value of the output, the load ld must be disabled ($ld = 0$). This is performed throughout the waveform in Figure 1. For instance, at $t = 10ns$, the register holds its initial value of 0 despite that the current state of the data d is 1.

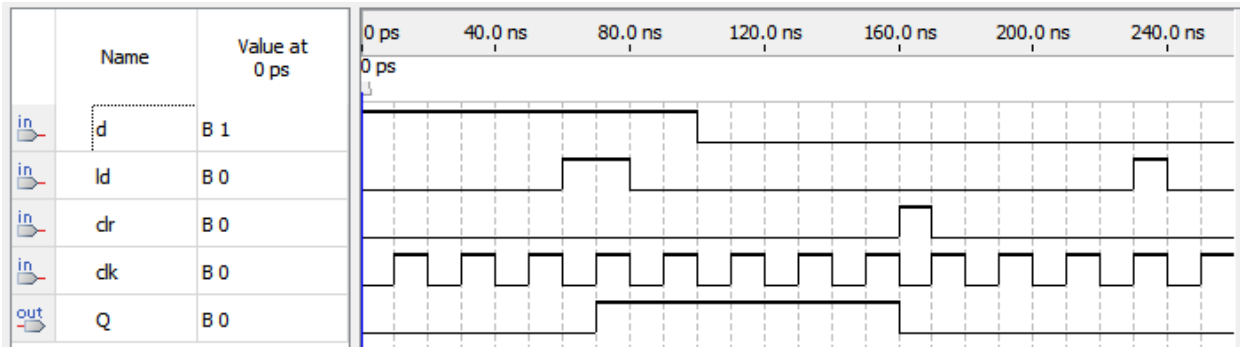

Figure 1. Output signal Q of the 1-bit register with varying input signal on d, ld, clr, and clk.

b. VHDL Code

```
1   library ieee;
2   use ieee.std_logic_1164.all;
3   use ieee.std_logic_arith.all;
4   use ieee.std_logic_unsigned.all;
5
6   entity register1 is
7   port  (
8         d  :  in std_logic;
9         ld :  in std_logic;
10        clr   :  in std_logic;
11        clk   :  in std_logic;
12        Q  :  out   std_logic
13        );
14  end register1;
15
16  architecture Behavior of register1 is
17  begin
18     process(ld, clr, clk)
19     begin
20        if clr = '1' then
21           Q <= '0';
22        elsif ((clk'event and clk = '1') and (ld = '1')) then
23           Q <= d;
24        end if;
25     end process;
26  end Behavior;
```

2. 32-bit Register

a. Description

Its operation is very similar to the 1-bit register – the only difference is that this register holds up to 32-bit of data. The behavior of its output Q is briefly discussed below:

- To transfer the 32-bit data d into the output, it must satisfy the following conditions: ld = 1, clr = 0, and $clk = rising\ edge$. Notice that in Figure 2, the register immediately updates its output Q into the current value of the data d at $t = 70ns$ and $t = 230ns$, which are times that those conditions are satisfied.
- To reset the 32-bit register, the clear clr must be enabled. At $t = 120ns$ in Figure 1, a reset is performed, forcing the output Q to have a value of 0.

- To hold the current value of Q, the load ld must be set to low. Note that in Figure 2 at $(100ns, 200ns)$ and $(300ns, 400ns)$, the values 836100318 and 3363007422 from d are never stored into the register since the ld remains disabled during those time intervals.

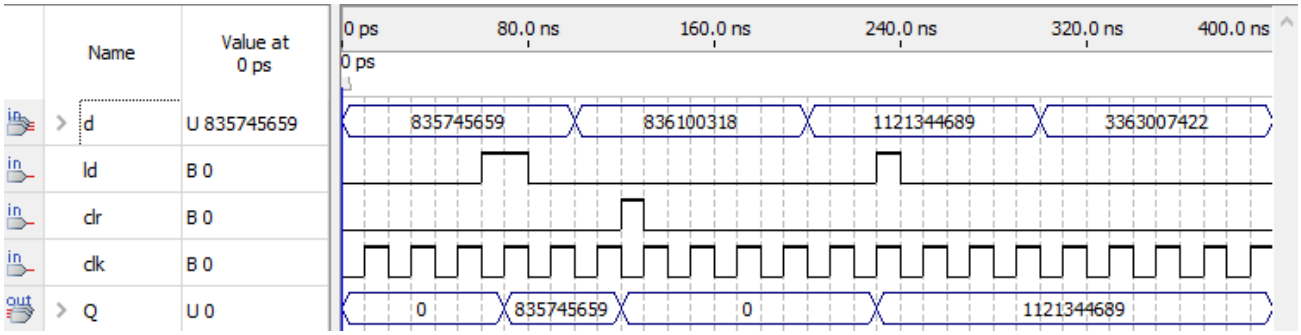

Figure 2. Output signal Q of the 32-bit register with varying input signal on d, ld, clr, and clk.

b. VHDL Code

```
1   library ieee;
2   use ieee.std_logic_1164.all;
3   use ieee.std_logic_arith.all;
4   use ieee.std_logic_unsigned.all;
5
6   entity register32 is
7   port  (
8         d  :  in std_logic_vector(31 downto 0);
9         ld :  in std_logic;
10        clr   :  in std_logic;
11        clk   :  in std_logic;
12        Q  :  out   std_logic_vector(31 downto 0)
13        );
14  end register32;
15
16  architecture Behavior of register32 is
17  begin
18     process(ld, clr, clk)
19     begin
20        if clr = '1' then
21           Q <= (others =>'0');
22        elsif ((clk'event and clk = '1') and (ld = '1')) then
23           Q <= d;
24        end if;
25     end process;
26  end Behavior;
```

3. Program Counter (PC)

a. Description

The program counter is made out of 3 components: adder, 2x1 multiplexer, and 32-bit register. Its main operations are summarized below:

- To increment the address q of the program counter, the increment inc from the multiplexer must be enabled such that the adder is directly connected to the data input of the register. This is performed in Figure 3 when $inc = 1$; that is, the time between $(0ns, 100ns)$ and $(170ns, 210ns)$. During these time intervals, the value of q is being incremented by 4 to simulate the execution of the next instruction from a MIPS architecture.
- To reset the program counter $(q = 0)$, the clear clr should be enabled in the register. In Figure 3, a reset is executed at $t = 140ns$, forcing the address q to change from 5 to 0.
- To change the address q into a specified value, the increment inc should be disabled so that the data d is connected to the input of the register. This can be seen in Figure 3, specifically for $t > 210ns$ when $inc = 0$. Rather than incrementing, the address q is always set to the current value of data d which is 5.
- To hold the address of q, the load ld must be disabled, or set to low. At $t = 50ns$ in Figure 3, the load ld is temporarily disabled; hence, the value of q did not increment despite that $inc = 1$, but rather, it holds the value of 8 until $t = 70ns$.

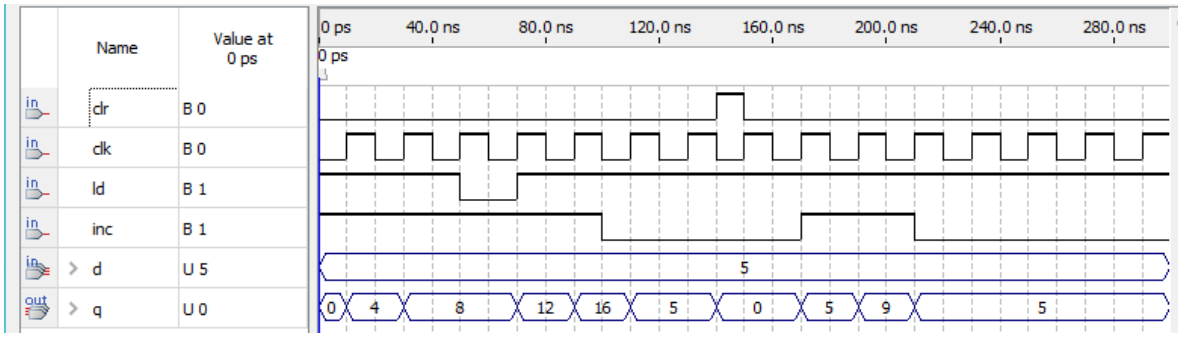

Figure 3. Output signal q of the PC with varying input signal on pins d, ld, inc, clr, and clk.

b. Program Counter VHDL Code

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.std_logic_arith.all;
4  use ieee.std_logic_unsigned.all;
5
6  entity pc is
7  port  (
8        clr   :  in std_logic;
9        clk   :  in std_logic;
10       ld :  in std_logic;
11       inc   :  in std_logic;
12       d  :  in std_logic_vector(31 downto 0);
13       q  :  out   std_logic_vector(31 downto 0)
14       );
15 end pc;
16
17 architecture Behavior of pc is
18    component add
19       port  (
20             A  :  in std_logic_vector(31 downto 0);
21             B  :  out   std_logic_vector(31 downto 0)
22          );
23    end component;
24    component mux2to1
25       port  (
26             s  :  in std_logic;
27             w0, w1   :  in std_logic_vector(31 downto 0);
28             f  :  out   std_logic_vector(31 downto 0)
29          );
30    end component;
31    component register32
32       port  (
33             d  :  in std_logic_vector(31 downto 0);
34             ld :  in std_logic;
35             clr   :  in std_logic;
36             clk   :  in std_logic;
37             Q  :  out   std_logic_vector(31 downto 0)
38          );
39    end component;
40    signal   add_out  :  std_logic_vector(31 downto 0);
41    signal   mux_out  :  std_logic_vector(31 downto 0);
42    signal   q_out :  std_logic_vector(31 downto 0);
43 begin
44    add0  :  add port map(q_out, add_out);
45    mux0  :  mux2to1 port map(inc, d, add_out, mux_out);
46    reg0  :  register32 port map(mux_out, ld, clr, clk, q_out);
47    q <= q_out;
48 end Behavior;
```

c. Adder VHDL Code

```
1   library ieee;
2   use ieee.std_logic_1164.all;
3   use ieee.std_logic_arith.all;
4   use ieee.std_logic_unsigned.all;
5
6   entity add is
7   port  (
8         A  :  in std_logic_vector(31 downto 0);
9         B  :  out   std_logic_vector(31 downto 0)
10        );
11  end add;
12
13  architecture Behavior of add is
14  begin
15  B <= A + 4;
16  end Behavior;
```

d. 2x1 Multiplexer VHDL Code

```
1   library ieee;
2   use ieee.std_logic_1164.all;
3
4   entity mux2to1 is
5   port  (
6         s  :  in std_logic;
7         w0, w1   :  in std_logic_vector(31 downto 0);
8         f  :  out   std_logic_vector(31 downto 0)
9     );
10  end mux2to1;
11
12  architecture Behavior of mux2to1 is
13  begin
14     with s select
15        f <= w0 when '0',
16             w1 when others;
17  end Behavior;
```