

EE 2000

Logic Circuit Design (Fall 2018)



Tutorial 5

Using Latch & Register in VHDL

Instructor: Dr. Ray C.C. Cheung

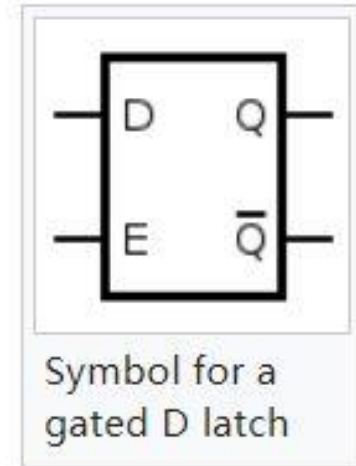
Objectives

- **Review** basic elements in sequential logic circuit design.
- Learn how to use a pipeline structure for simple calculations for the 4 to 2 encoder.
- Learn some definition and usage of calculations in VHDL.
- Using Vivado simulator to observe how it works.

REVIEW: 1-Bit Latch

Gated D latch truth table

E/C	D		Q	$\bar{Q}$	Comment
0	X		Q_{prev}	$\bar{Q}_{\text{prev}}$	No change
1	0		0	1	Reset
1	1		1	0	Set



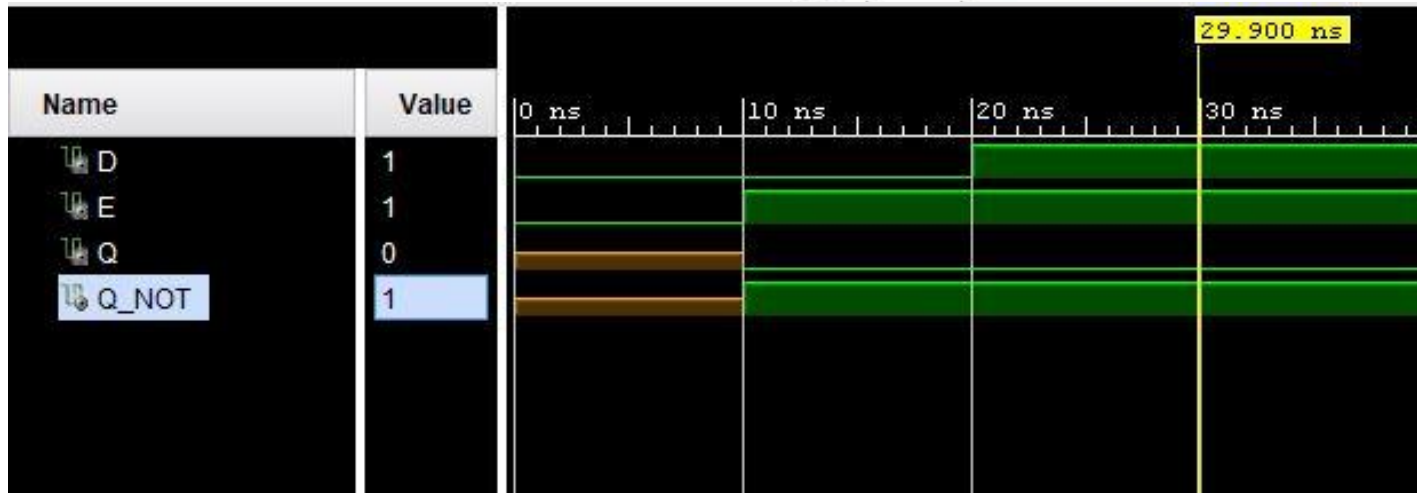
For the animated gates:

[https://en.wikipedia.org/wiki/Flip-flop_\(electronics\)#/media/File:GatedDLatch-lowres.gif](https://en.wikipedia.org/wiki/Flip-flop_(electronics)#/media/File:GatedDLatch-lowres.gif)

REVIEW: 1-Bit Latch

Students can download the file online

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3
4  entity latch_lbit is
5      Port ( D : in STD_LOGIC;
6            E : in STD_LOGIC;
7            Q : out STD_LOGIC;
8            Q_NOT : out STD_LOGIC);
9  end latch_lbit;
10
11  architecture Behavioral of latch_lbit is
12  begin
13
14      process (E)
15      begin
16          if (E = '1') then
17              Q <= D;
18              Q_NOT <= NOT D;
19          end if;
20      end process;
21
22  end Behavioral;
```



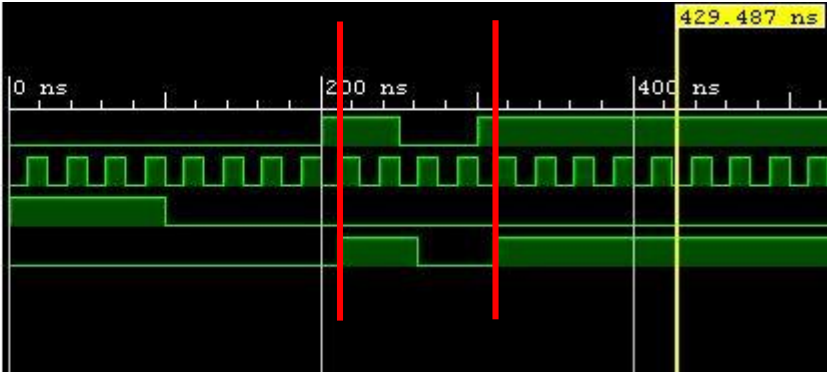
REVIEW: D Flip-flop

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3
4  entity flip_flop is
5      Port ( D : in STD_LOGIC;
6            clk : in STD_LOGIC;
7            rst : in STD_LOGIC;
8            Q : out STD_LOGIC);
9  end flip_flop;
10
11  architecture Behavioral of flip_flop is
12  begin
13
14      process(clk)
15      begin
16
17          if (rst = '1') then
18              Q <= '0';
19          else
20              if (rst = '0') then
21                  if clk' event and clk = '1' then
22                      Q <= D;
23                  end if;
24              end if;
25          end if;
26      end process;
27
28  end Behavioral;
```

Name	Value
D	1
clk	0
rst	0
Q	1

Clock	D	Q _{next}
Rising edge	0	0
Rising edge	1	1
Non-Rising	X	Q

Simple 1-bit D Flip-flop



Simulate with the testbench

- Finish the source file of flip_flop entity.
- Notice the format of VHDL and please avoid syntax errors.
- Please open the project of FF_template and add source file. Testbench has given in this project.

REVIEW: 1-Bit Register

CLK	RST	Data in	Data out
-	1	-	Z (or set it clear)
Rising Edge	0	D(next)	D(next)
Non-rising	0	-	D(old)

```

1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3
4  entity reg is
5      Port ( indata : in STD_LOGIC_VECTOR (1 downto 0);
6            clk : in STD_LOGIC;
7            rst : in STD_LOGIC;
8            outdata : out STD_LOGIC_VECTOR (1 downto 0));
9  end reg;
10
11 architecture Behavioral of reg is
12 begin
13     process (clk,rst)
14     begin
15         if (rst = '1') then
16             outdata <= "ZZ";
17         elsif (rst = '0') then
18             if (rising_edge(clk)) then
19                 outdata <= indata;
20             end if;
21         end if;
22     end process;
23
24
25 end Behavioral;
26

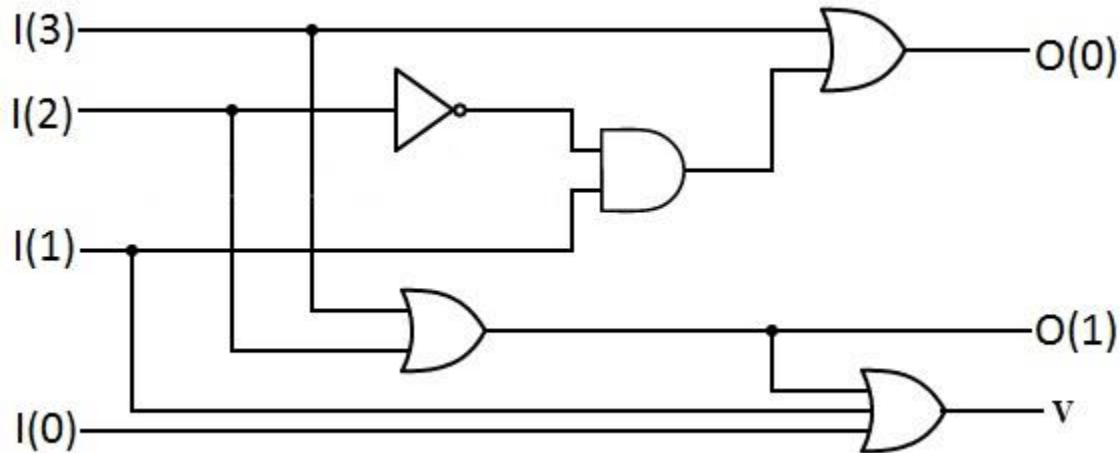
```

1. Change the code for 1 bit:
STD_LOGIC
2. Outdata <= 'Z'

Name	Value
indata	1
clk	0
rst	0
outdata	1



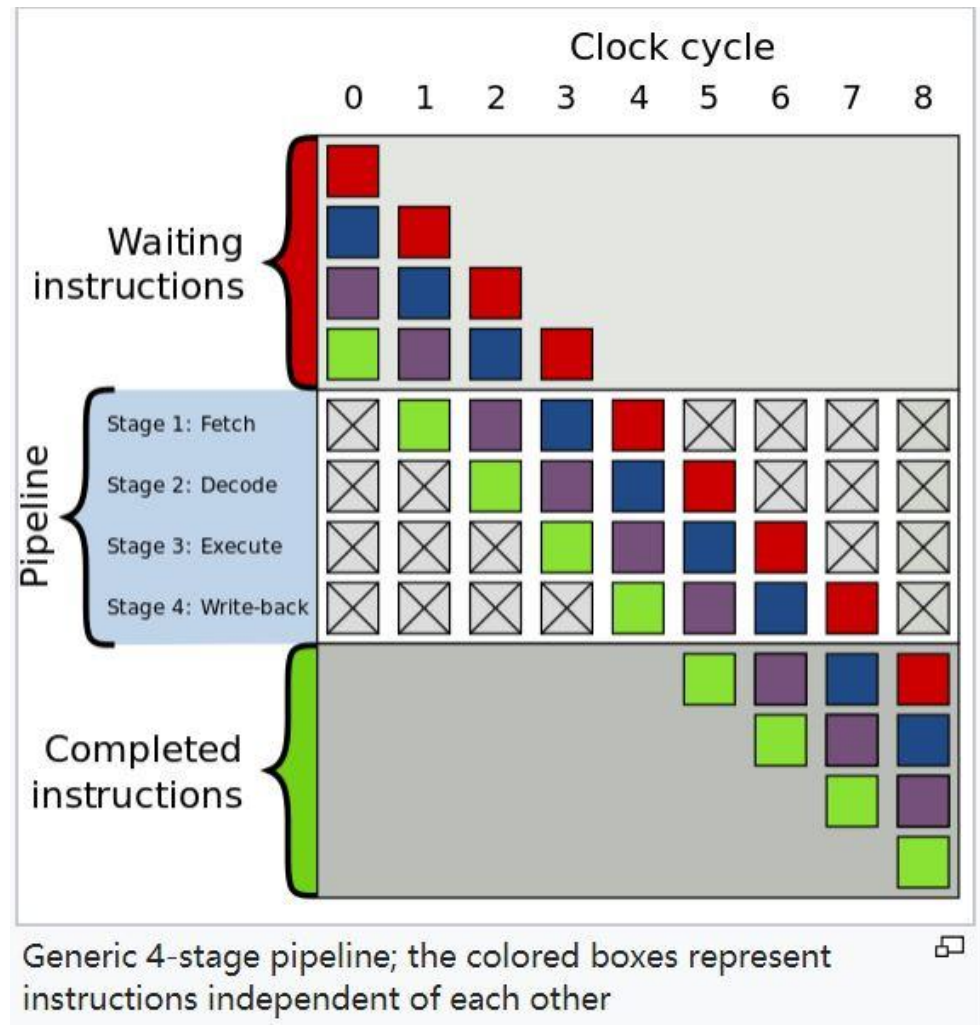
REVIEW: Priority Encoder



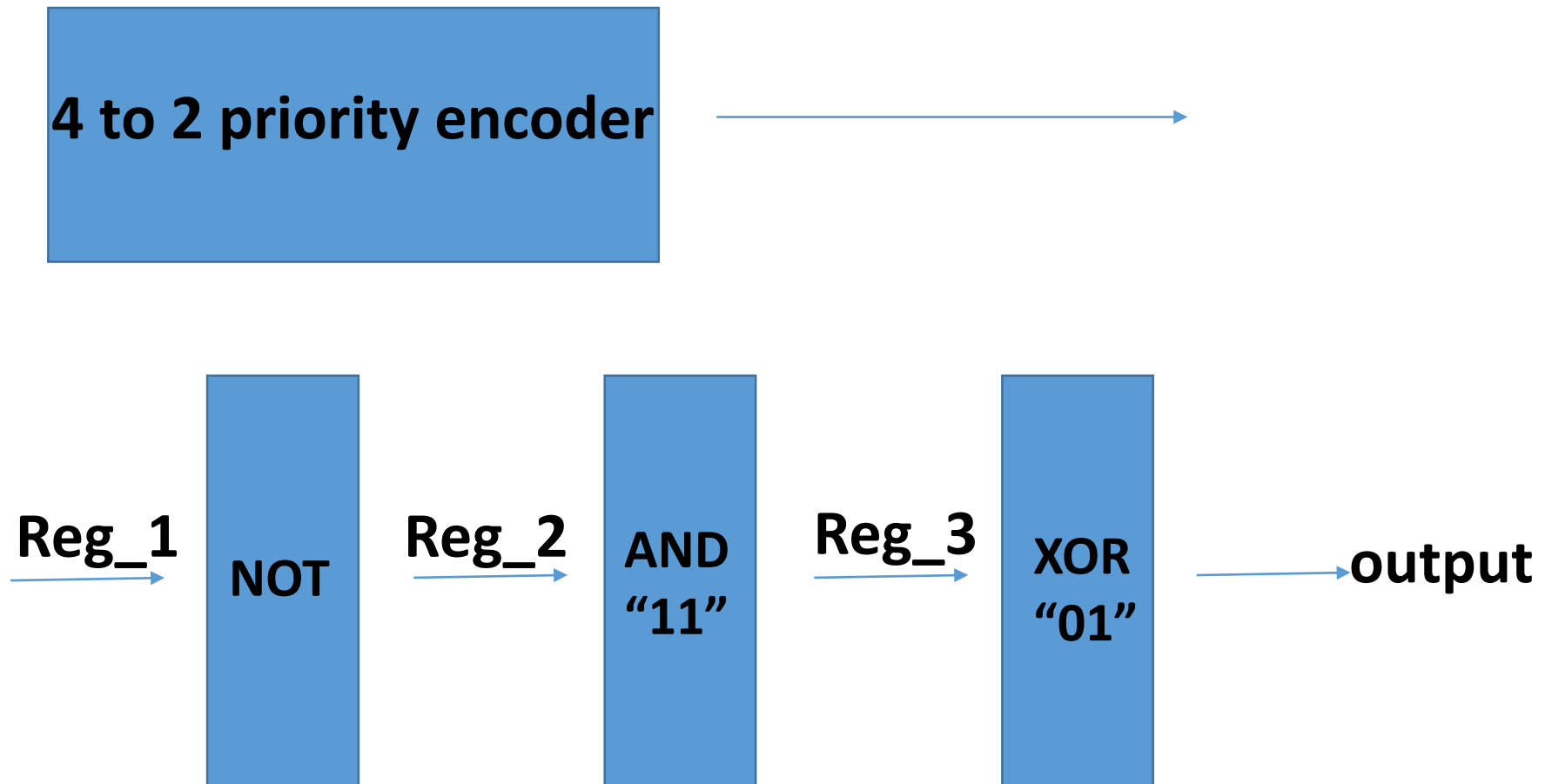
- A **priority encoder** is a circuit or algorithm that compresses multiple binary inputs into a smaller number of outputs.
- Please refer to `encoder_4to2` for answers in tutorial 4.

Pipeline Structure

Clock	Execution
0	•Four instructions are waiting to be executed
1	•The green instruction is fetched from memory
2	•The green instruction is decoded •The purple instruction is fetched from memory
3	•The green instruction is executed (actual operation is performed) •The purple instruction is decoded •The blue instruction is fetched
4	•The green instruction's results are written back to the register file or memory •The purple instruction is executed •The blue instruction is decoded •The red instruction is fetched
5	•The execution of green instruction is completed •The purple instruction is written back •The blue instruction is executed •The red instruction is decoded
6	•The execution of purple instruction is completed •The blue instruction is written back •The red instruction is executed
7	•The execution of blue instruction is completed •The red instruction is written back
8	•The execution of red instruction is completed
9	•The execution of all four instructions is completed



Design the entity as followings:



Step by Step

Open the template of tutorial 5



Add an entity for register



Revise the pipeline calculation
entity



Simulate the encoder

Register for sequential circuits

- A register is important for saving data so that the data will be stored to next time.
- For simple few bits register, we can define by ourselves.

failing_edge(clk) for failing edge trigger

For other usage can refer to clk' event from IEEE 1164 STANDARD

entity reg is

Port (indata : in STD_LOGIC_VECTOR (1 downto 0);

clk : in STD_LOGIC;

rst : in STD_LOGIC;

outdata : out STD_LOGIC_VECTOR (1 downto 0));

end reg;

architecture Behavioral of reg is

begin

process (clk,rst)

begin

if (rst = '1') then

outdata <= "ZZ";

elsif (rst = '0') then

if (rising_edge(clk)) then

outdata <= indata;

end if;

end if;

end process;

end Behavioral;

Top entity for pipeline structure

```

1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3
4  entity pipe_calcu is
5      Port ( S : in STD_LOGIC_VECTOR (3 downto 0);
6            O : out STD_LOGIC_VECTOR (1 downto 0);
7            clk : in STD_LOGIC;
8            rst : in STD_LOGIC;
9            en : in STD_LOGIC );
10 end pipe_calcu;
11
12 architecture Behavioral of pipe_calcu is
13
14     SIGNAL      V : STD_LOGIC;
15     SIGNAL o_en1 : STD_LOGIC_VECTOR (1 DOWNTO 0);
16     SIGNAL i_reg1 : STD_LOGIC_VECTOR (1 DOWNTO 0);
17     SIGNAL i_reg2 : STD_LOGIC_VECTOR (1 DOWNTO 0);
18     SIGNAL i_reg3 : STD_LOGIC_VECTOR (1 DOWNTO 0);
19     SIGNAL o_reg1 : STD_LOGIC_VECTOR (1 DOWNTO 0);
20     SIGNAL o_reg2 : STD_LOGIC_VECTOR (1 DOWNTO 0);
21     SIGNAL o_reg3 : STD_LOGIC_VECTOR (1 DOWNTO 0);
22
23     COMPONENT encoder      is
24     Port (
25         S : in STD_LOGIC_VECTOR(3 DOWNTO 0);
26         en : in STD_LOGIC;
27         V : out STD_LOGIC;
28         O : out STD_LOGIC_VECTOR(1 DOWNTO 0)
29     );
30 END COMPONENT;

```

Design Sources (3)

pipe_calcu(Behavioral) (pipe2.vhd) (4)

```

vh uut_encoder : encoder_4to2(Behavioral) (encoder_4to2
vh uut_reg1 : reg(Behavioral) (reg.vhd)
vh uut_reg2 : reg(Behavioral) (reg.vhd)
vh uut_reg3 : reg(Behavioral) (reg.vhd)

```

```

31
32 COMPONENT reg is
33 Port ( indata : in STD_LOGIC_VECTOR (1 downto 0);
34       clk : in STD_LOGIC;
35       rst : in STD_LOGIC;
36       outdata : out STD_LOGIC_VECTOR (1 downto 0));
37 END COMPONENT;
38
39 begin
40 uut_encoder : encoder      PORT MAP (S, en, V, o_en1);
41 uut_reg1   : reg          PORT MAP (i_reg1, clk, rst, O_reg1);
42 uut_reg2   : reg          PORT MAP (i_reg2, clk, rst, o_reg2);
43 uut_reg3   : reg          PORT MAP (i_reg3, clk, rst, o_reg3);
44
45     i_reg1 <= o_en1;
46     i_reg2 <= NOT O_reg1 ;
47     i_reg3 <= o_reg2 AND "11";
48     O <= O_reg3 XOR "01";
49 end Behavioral;

```

Testbench

```

1
2  library ieee;
3  use ieee.std_logic_1164.all;
4
5  entity tb_pipe_calcu is
6  end tb_pipe_calcu;
7
8  architecture tb of tb_pipe_calcu is
9
10     component pipe_calcu
11     port (S    : in std_logic_vector (3 downto 0);
12          O    : out std_logic_vector (1 downto 0);
13          clk  : in std_logic;
14          rst  : in std_logic;
15          en   : in std_logic);
16     end component;
17
18     signal S    : std_logic_vector (3 downto 0);
19     signal O    : std_logic_vector (1 downto 0);
20     signal clk  : std_logic;
21     signal rst  : std_logic;
22     signal en   : std_logic;
23
24     constant TbPeriod : time := 50 ns; -- EDIT Put right period here
25     signal TbClock    : std_logic := '0';
26     signal TbSimEnded : std_logic := '0';

```

```

27
28  begin
29
30     dut : pipe_calcu
31     port map (S    => S,
32              O    => O,
33              clk  => TbClock,
34              rst  => rst,
35              en   => en);
36
37     -- Clock generation
38     TbClock <= not TbClock after TbPeriod/2 when TbSimEnded /= '1' else '0';
39
40     -- EDIT: Replace YOURCLOCK SIGNAL below by the name of your clock as I have
41     -- YOURCLOCK SIGNAL <= TbClock;
42
43     stimuli : process
44     begin
45         -- EDIT Adapt initialization as needed
46
47         S <= (others => '0');
48         clk <= '0';
49         en <= '0';
50         rst <= '1';
51         wait for 50 ns;
52         rst <= '0';
53         en <= '1';
54         wait for 50 ns;
55         S <= "0001";
56

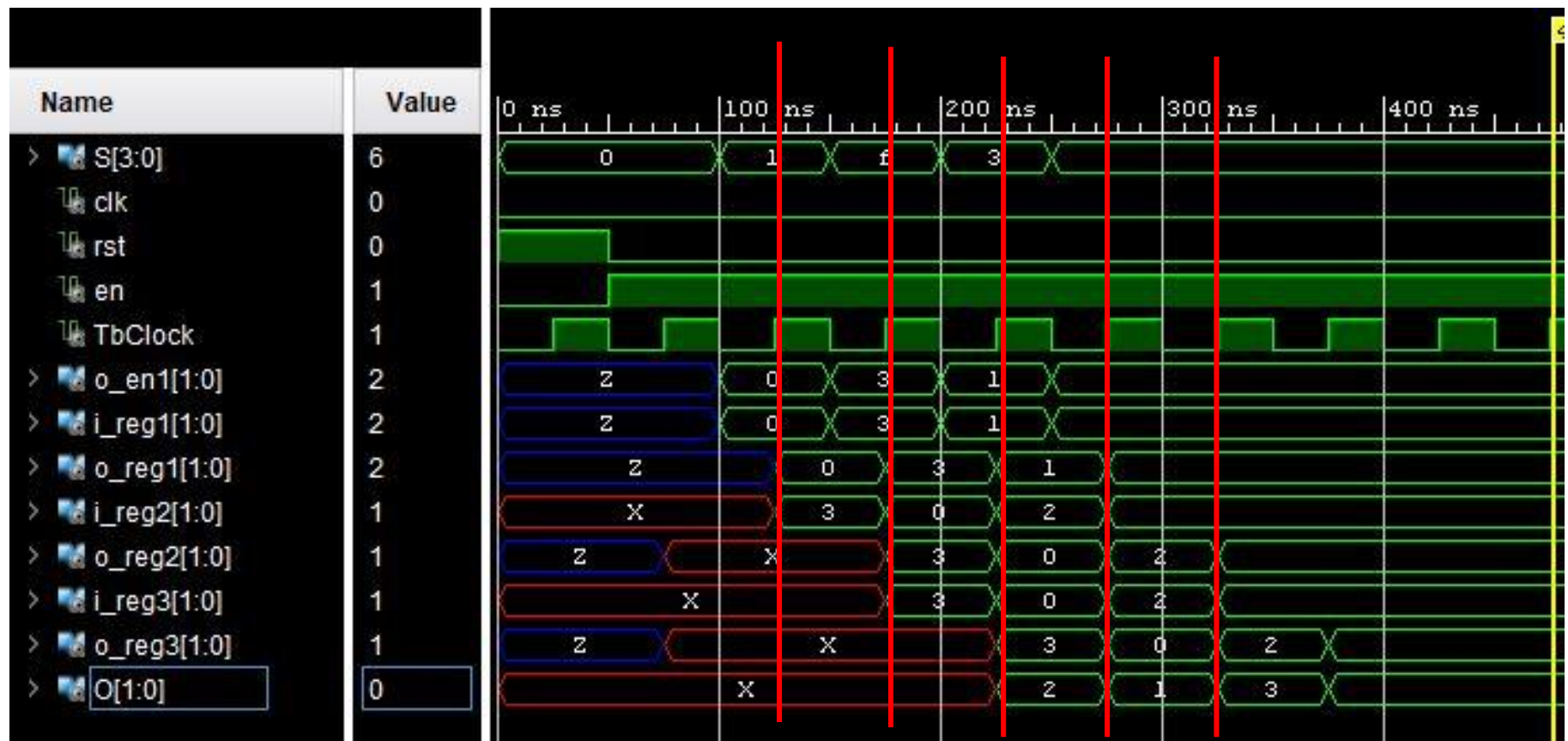
```

```

56
57         wait for 50 ns;
58         S <= "1111";
59         wait for 50 ns;
60         S <= "0011";
61         wait for 50 ns;
62         S <= "0110";
63
64         wait ;
65     end process;
66
67 end tb;

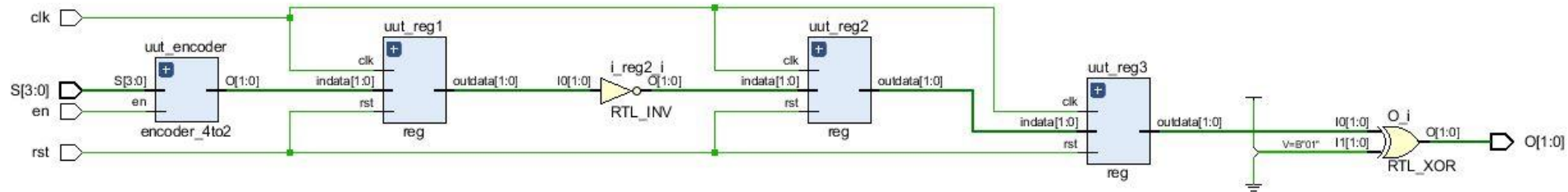
```

Simulation results



For more complicated entity or logic circuits, pipeline structure can help lower the complexity and accelerate the calculations. E.g. pipeline multiplier.

Schematic of the entity



Instructions:

- `clk` and `rst` connect to each register to control them in this entity.
- The calculation “AND 11” can be seen as making the output directly so that there is no RTL units.

Extended Reference

- <https://surf-vhdl.com/how-to-implement-pipeline-multiplier-vhdl/> pipeline multiplier in VHDL
- <http://www.csbio.unc.edu/mcmillan/Media/L10Spring2013.pdf> for binary multiplication

Tutorial - Remarks

- Please raise your hand if you have questions!
 - 9 course tutors + course Instructors
 - If you finish the tutorial quickly, please try different settings in the Vivado tool.
- You can also refer to the lecture notes on Canvas.
- Keep good habits in writing codes and avoid syntax errors.
- Add more functions for the pipeline structure. If you are interest in FPGA calculating algorithms, you can try to do multiplications in VHDL.