Sequential Logic Circuits

Learning Outcomes

At the end of this activity, you should be able to:

1. differentiate combinational and sequential elements in a processor;
2. implement sequential logic circuit elements in VHDL;

Contents

1 Resources

- Video: https://youtu.be/CzyXb_T-xgU.
- Source Codes: https://git.io/JU3al

2 Discussion

The discussion in this handout aims only to provide an outline of what is in the video lecture. It is recommended that you watch the video in its entirety.

Sequential elements are sometimes called memory elements because they store state information. The *output* of any memory element dependes on both the *input* and the *value stored* in it. An understanding of clocks is important in the study of sequential elements.

2.1 Clocks

Clocks determine when the current state of a sequential elements needs to be updated. It is a *signal* that transitions from low to high and high to low in a fixed cycle time or clock period. Figure 1 shows an example clock signal. It is a digital signal so it is represented by a square wave and the transition from low to high or high to low is abrupt unline in analog signals. These abrupt transitions are called *clock edges* which may be a *rising edge* (low to high) or *falling edge* (high to low). The *clock period* is the time to complete a cycle and the number of cycles per second is called the *frequency* in *hertz*. The higher the frequency the higher the number of cycles per second. We will use these parameters later when we go to the performance evaluation lab.

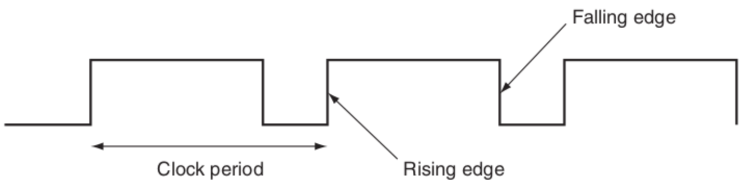


Figure 1: A clock signal showin the clock period, falling edge, and rising edge.

The updating of the state element is usually done at the edge of the clock to ensure that the signals will be valid. Figure 2 shows a comninational element sandwiched between two state elements. Observe that the updating of the state elements are done at the rising edge. Using the clock edge as marker allows the same state element to be the input and output of the combinational element without invalidating the signal.

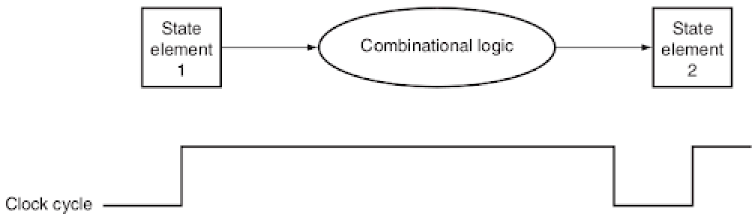


Figure 2: A combinational element is sandwiched between two state elements. The state elements are updated at the rising clock edge.

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY clocker_tb IS
END clocker_tb;
ARCHITECTURE behavior OF clocker_tb IS
   --100Mhz
   CONSTANT frequency: integer := 100e6;
   CONSTANT period : time := 1000 ms / frequency;
   SIGNAL clk : std_logic := '0';
BEGIN
   clk <= not clk after period / 2;
   -- do some stuff here using clk as input
END ARCHITECTURE;
```

2.2 Latches

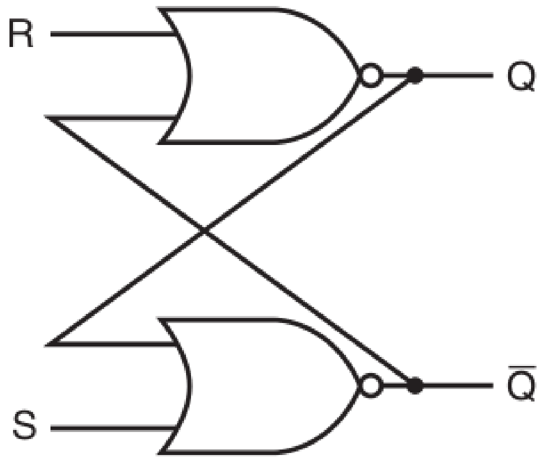


Figure 3: 2-to-1 Multiplexer.

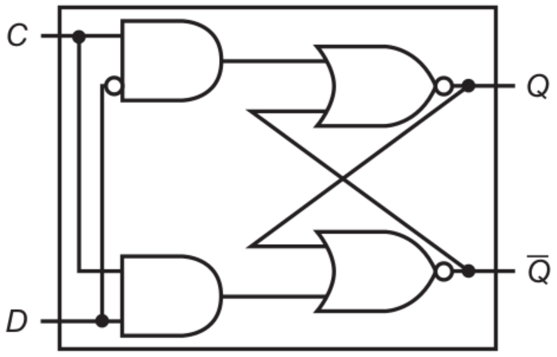


Figure 4: 2-to-1 Multiplexer.

2.3 Flip-Flops

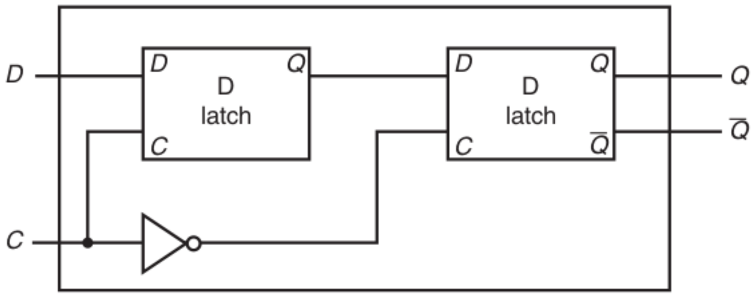


Figure 5: 2-to-1 Multiplexer.

2.4 Register Files

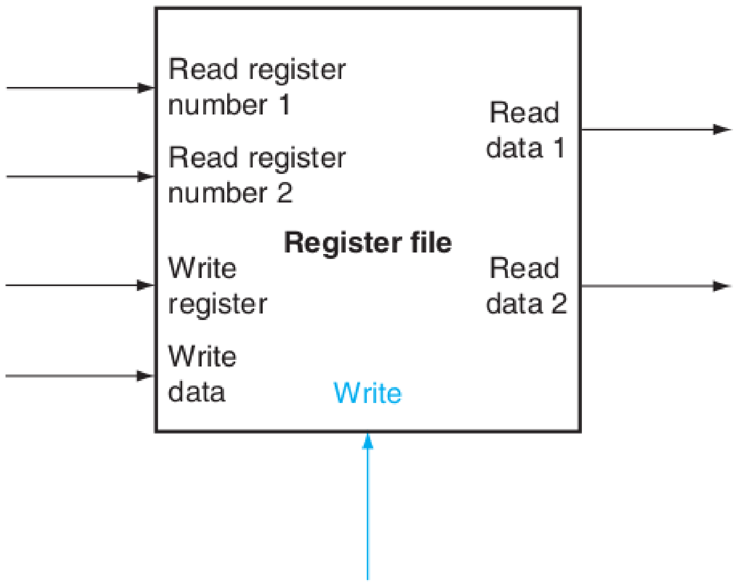


Figure 6: 2-to-1 Multiplexer.

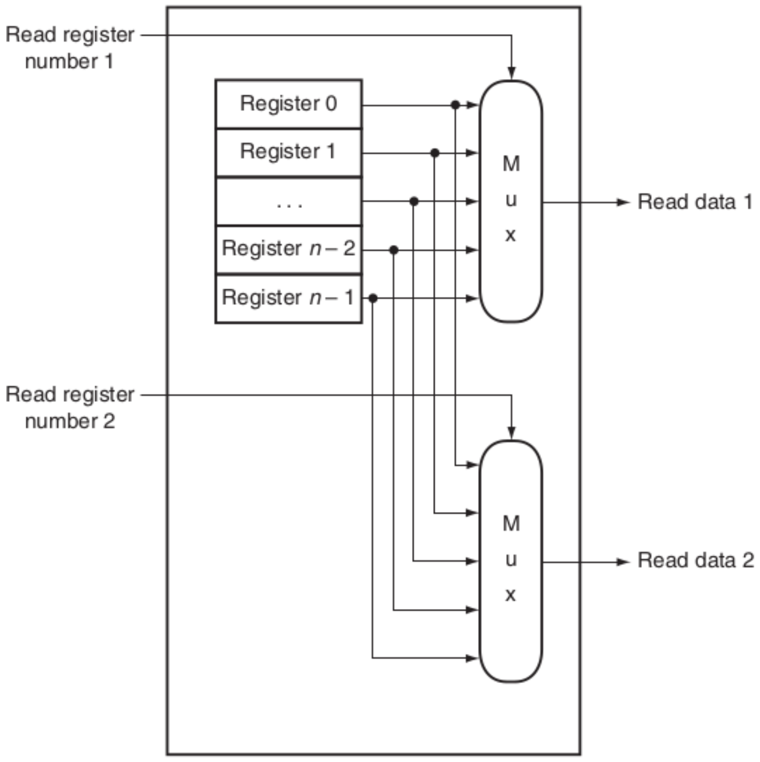


Figure 7: 2-to-1 Multiplexer.

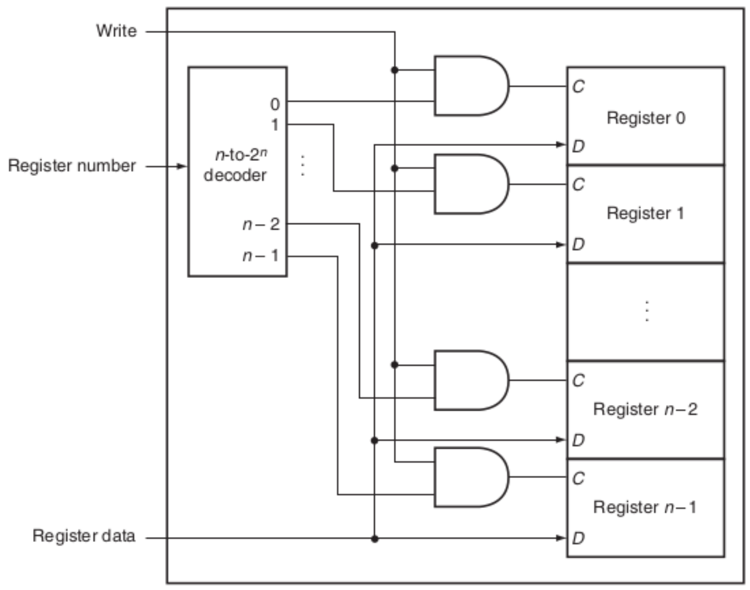


Figure 8: 2-to-1 Multiplexer.

2.5 Static RAM

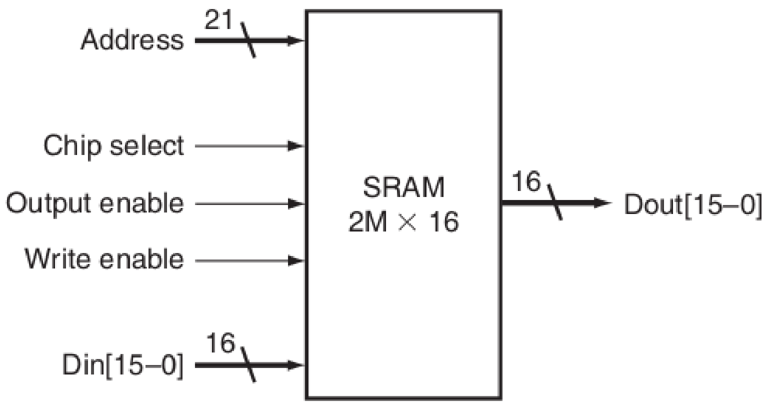


Figure 9: 2-to-1 Multiplexer.

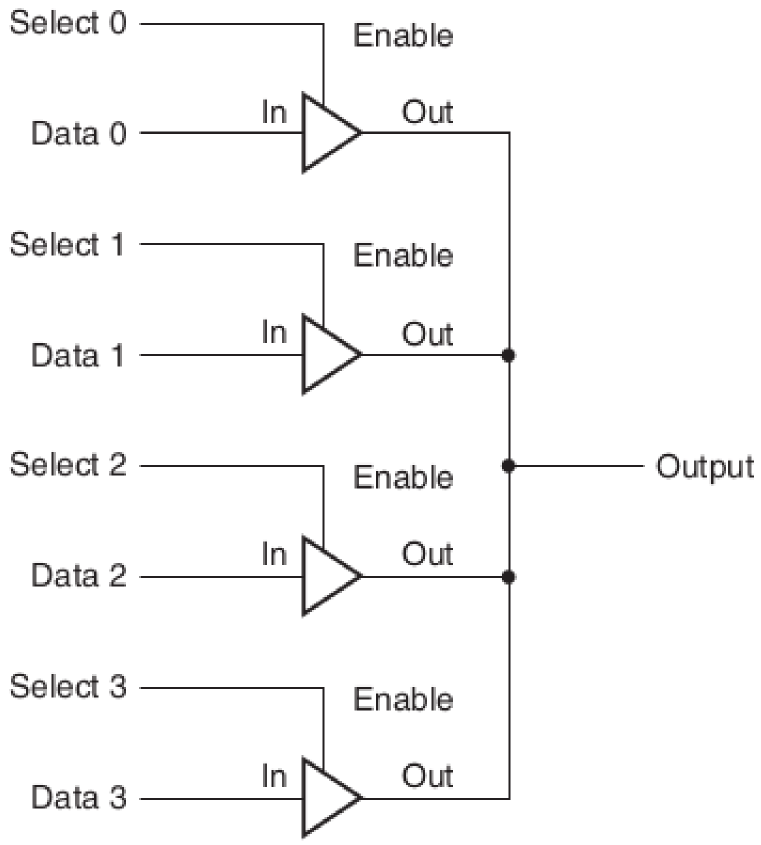


Figure 10: 2-to-1 Multiplexer.

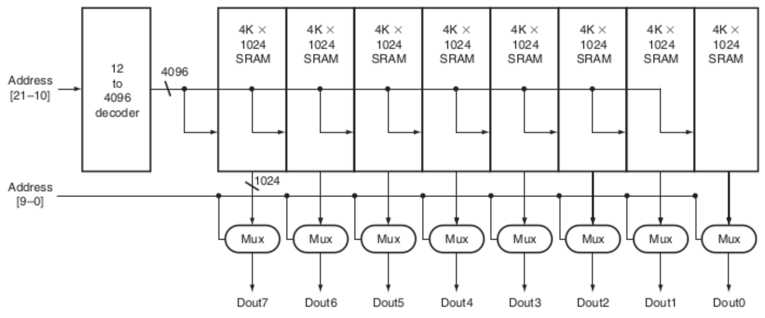


Figure 11: 2-to-1 Multiplexer.

2.6 Dynamic RAM

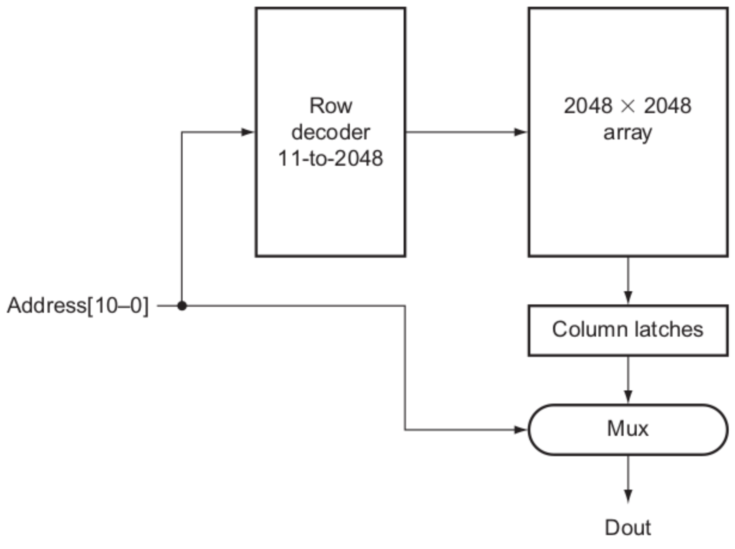


Figure 12: 2-to-1 Multiplexer.

2.7 Finite State Machines

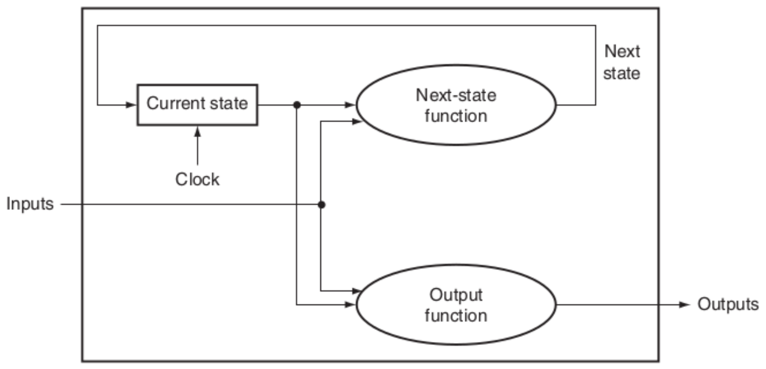
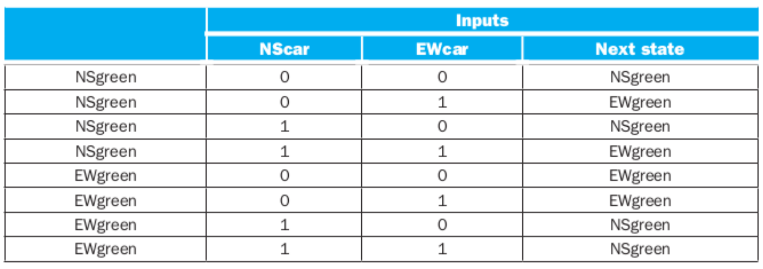
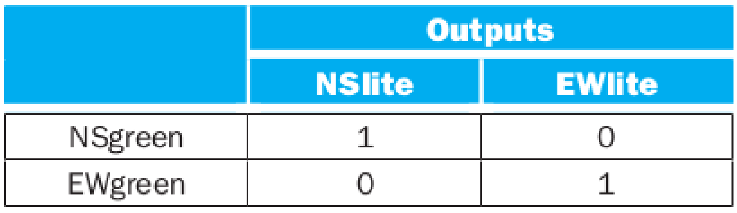


Figure 13: 2-to-1 Multiplexer.



	Inputs		
	NScar	EWcar	Next state
NSgreen	0	0	NSgreen
NSgreen	0	1	EWgreen
NSgreen	1	0	NSgreen
NSgreen	1	1	EWgreen
EWgreen	0	0	EWgreen
EWgreen	0	1	EWgreen
EWgreen	1	0	NSgreen
EWgreen	1	1	NSgreen

Figure 14: 2-to-1 Multiplexer.



	Outputs	
	NSlite	EWlite
NSgreen	1	0
EWgreen	0	1

Figure 15: 2-to-1 Multiplexer.

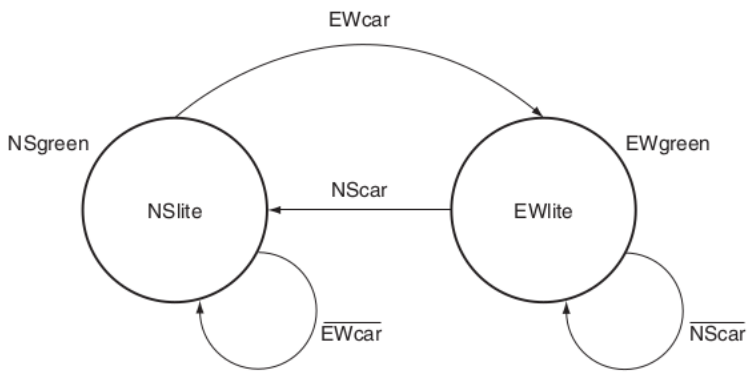


Figure 16: 2-to-1 Multiplexer.

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
---------------------------------
ENTITY mux_2to1 IS
   PORT (A, B, S: IN STD_LOGIC;
   C: OUT STD_LOGIC);
END mux_2to1;
--------------------------------

ARCHITECTURE pure_logic OF mux_2to1 IS
BEGIN
--- C <= (A AND NOT S) OR (B AND S);
   C <= A WHEN S='0' ELSE B;
END pure_logic;
```

3 Summary

We discussed some of the sequential elements that are useful in the design of a processor. We also showed the design and implementation of a simple traffic light system.

4 Learning Activities

Download the source codes for this lab then try experimenting by adding more test cases in the testbenches. Submit a PDF document that shows screenshots of your modifications and runs.

5 Deliverable

Your final deliverable for this lab is described in the accompanying exercise handout.

References

[1] David A. Patterson and John L. Hennessy. *Computer Organization and Design: The Hardware/Software Interface, ARM Edition.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, arm edition edition, 2017.