## Contents

1. CLAB2: Sequential Circuits 3
   1.1 Aims ................................................................. 3
   1.2 Sequential Logic ................................................. 3
   1.3 The RS Latch ....................................................... 4
   1.4 The D Latch and Gated D latch ............................... 8
   1.5 The D Flip-flop ..................................................... 10
1 CLAB2: Sequential Circuits

1.1 Aims

- Understand how sequential circuits are built and the underpinnings of the sequential paradigm.
- Learn to use ICARUS VERILOG to simulate hardware described by VERILOG and GTKWAVE to plot the results.
- Learn to use XILINX ISE WebPACK 9.2i to create hardware designs in schematics and VERILOG HDL (Hardware Definition Language).
- Learn to use VERILOG to study sequential logic circuits.
- Further investigate the synthesis of hardware in ISE WebPACK.

REMEMBER TO USE YOUR FLASH DRIVE TO SAVE ALL YOUR WORK AFTER THE LAB.

1.2 Sequential Logic

Sequential logic differs from combinational logic in that the output of the logic device is dependent not only on the present inputs to the device, but also on past inputs; i.e., the output of a sequential logic device depends on its present internal state and the present inputs. This implies that a sequential logic device has some kind of memory of at least part of its “history” (i.e., its previous inputs).

In this lab we study the basics of sequential circuits. You may recall that in this course all sequential devices are synchronous devices in which output changes update synchronously with the active edge of the clock. In order to understand how these circuits work and why they are advantageous we need to start at the very origin of sequential logic: memory devices constructed from purely combinational gates. In this lab we will look at how to describe and simulate sequential circuits in VERILOG and ISE WebPACK.

In lectures we learned how to represent combinational and synchronous sequential circuits in VERILOG. Fig[1] shows the code.
In this lab we investigate the properties of sequential devices in a bid to justify the representation of Fig 1. We also try to understand how asynchronous level driven memory devices can be unwittingly generated in VERILOG.

### 1.3 The RS Latch

A simple memory device can be constructed from combinational devices with which we are already familiar. By a memory device, we mean a device which can remember if a signal of logic level 0 or 1 has been connected to one of its inputs, and can make this fact available at an output. A very simple, but still useful, memory device can be constructed from a simple OR gate, as shown in Fig. 2.

Figure 1:

![Figure 1:](image1)

Figure 2:

![Figure 2:](image2)

Figure 3:

![Figure 3:](image3)

```verilog
/* **********************************/
always @(posedge clk or A) begin
    // Non-blocking IO
    X <= Y;
    Y <= A;
end

/* **********************************/
always @(A or B) begin
    // Blocking IO
    D = B;
    C = A;
end
```

Sequential code assignment at the posedge clk

Combinational code

All values settle, clock period >> settling time
Exercise 1: It is very important to understand this formative memory device. Do you see how it works. The device is all but useless: it is a “onceler”. Once the input goes ’1’, the output transitions to a ’1’ indefinitely.

Note that the output of the memory is used as one of the inputs; this is called feedback and is characteristic of programmable memory devices. (Without feedback, a “permanent” electronic memory device would not be possible.) The use of feedback in a device can introduce problems which are not found in strictly combinational circuits. In particular, it is possible to inadvertently construct devices for which the output is not determined by the inputs, and for which it is not possible to predict the output. A simple example is an inverter with its input connected to its output. Such a device is logically inconsistent; in a physical implementation the device would probably either oscillate from 1 to 0 to 1 · · · or remain at an intermediate value between logic 0 and logic 1, producing an invalid and erroneous output.

More complicated, stable, memory elements could be constructed using simple logic gates. In particular, simple, alterable memory cells can be readily constructed. One basic (but not often used in this form) memory device is the following, called an RS (reset-set) latch. It is the most basic of all the class of circuits which are called latches. A logic diagram for this device is shown in Fig. 4.

1Often the term flip-flop is synonymous with latch however in this course a latch will be taken to mean a level activated or asynchronous memory device whereas a flip-flop will mean a clock synchronous edge triggered memory device

2This particular incarnation uses a pair of NOR gates but a similar circuit can be constructed from NAND gates. This NAND version is referred to as the dual of the NOR RS latch
The symbol for the RS latch is shown in Fig. 4.

We can analyse the circuit shown in Fig. 4 in a straightforward manner, to determine the outputs $Q$ and $\overline{Q}$ for various inputs to $R$ and $S$; e.g., suppose we raise $S$ to logic 1, with
$Q$ at logic 0. Then $\overline{Q}$ must be 0, (the output of a NOR gate is 0 if any input is 1), and $Q$ must be 1. If $S$ is returned to 0, then $\overline{Q}$ remains 0 and $Q$ remains 1; i.e., the RS latch “remember” if $S$ was set to 1, while $R$ is 0. If $R$ is raised to logic 1 while $S$ is at logic 0, then $Q$ is set to logic 0, and $\overline{Q}$ is set to logic 1; i.e., the latch is reset. If both $R$ and $S$ are raised to logic 1, then both $Q$ and $\overline{Q}$ will be at logic 0. This output is inconsistent with the identification of $Q$ and $\overline{Q}$ as the two outputs, and therefore this combination of inputs for $R$ and $S$ should be avoided.

**Exercise 2: Analyse and write down the truth/transition table for the NAND based RS latch**

A more serious problem occurs if $R$ and $S$ change from logic 1 to logic 0 simultaneously. This situation is called a race condition. If both $R$ and $S$ are at logic 1, then $Q$ and $\overline{Q}$ are at logic 0. When $R$ and $S$ are both set to 0, then both $Q$ and $\overline{Q}$ should switch to logic 1. However, when they switch to logic 1, they should cause a switch back to logic 0 again, because if the logic 1 input to each NOR gate. If both NOR gates were identical, this would occur over and over again, indefinitely – an oscillation of the outputs $Q$ and $\overline{Q}$ from state 0 to 1 and back, with a period depending on the time delay for the NOR gate. In practice, one gate is a little faster than the other, and the final outcome depends on the relative speeds of the two gates. However, the final outcome cannot be predicted.

A state table for the RS flip flop is shown in Fig. 4, together with its a logic diagram and its circuit symbol. (The circuit symbol for a flip flop or latch is a rectangular block, with the inputs on the left and outputs on the right. Special inputs may be on the top and bottom.)

The VERILOG code that literally describes the NOR based RS-latch is shown as follows.

```verilog
module RSlatch_nor_comb(S, R, Q);
    input S;
    input R;
    output Q;
    wire Qa;
    wire Qb;

    assign Qa = ~(S | Qb);
    assign Qb = ~(R | Qa);
    assign Q = Qa;
endmodule
```

The following code shows a behavioural representation of the RS latch. Note that all possible combinations of $R$ and $S$ are contained in the always block and that the block is activated by changes in $R$ or $S.
module RSlatch_nor_seq(S, R, Q);

input S;
input R;
output Q;

wire R;
wire S;
reg Q;

always @(R or S) begin
    if(~S & ~R) Q = Q;
    if( S & ~R) Q = 0;
    if(~S & R) Q = 1;
    if( S & R) Q = 0;
end
endmodule

Exercise 3: Download and simulate this code in ICARUS VERILOG and GTKWAVE. Note that there is the one test bench TB_rslatch.v for both modules - you’ll need to comment out the unused instantiation. Compare the operation of the above two descriptions. What do you conclude?

1.4 The D Latch and Gated D latch

It is possible to create a latch which has no race condition, simply by providing only one input to a RS latch, and generating an inverted signal to present to the other terminal of the latch. In this case, the S and R inputs are always inverted with respect to each other, and no race condition can occur. The circuit for a D latch is shown in Fig. 6.

Fig. 7 shows the function of the gated D-latch compared to the D flip-flop (note that as yet we do not know how to build a D flip-flop!).

8
The D latch is used to capture, or “latch” the logic level which is present on the Data line when the clock input is high. If the data on the $D$ line changes state while the clock pulse is high, then the output, $Q$, follows the input, $D$. This effect can be seen in the timing diagram, Fig. 7. The D flip-flop, while a slightly more complicated circuit, performs a function very similar to the D latch. In the case of the D flip-flop, however, the rising edge of the clock pulse is used to “capture” the input to the flip flop. This device is very useful when it is necessary to “capture” a logic level on a line which is very rapidly varying. Figure 2.8 (b) shows a timing diagram for a D-type flip-flop. This type of device is said to be “edge triggered” – either rising edge triggered (i.e. a 0-1 transition) or falling edge triggered (i.e., a 1-0 transition) devices are available.

The VERILOG code that describes the gated D-latch is shown as follows.

```verilog
module GDlatch_nand_comb (D, Clk, Q);
  input D, Clk;

```
module GDlatch_nand_seq (D, Clk, Q);
  input D, Clk;
  output reg Q;
  always @ (D or Clk)
    if (Clk)
      Q = D;
endmodule

The following code shows a behavioural representation. Note the similarity that this has to the D flip-flop of Fig. 11

module GDlatch_nand_seq (D, Clk, Q);
  input D, Clk;
  output reg Q;
  always @ (D or Clk)
    if (Clk)
      Q = D;
endmodule

Exercise 4: Download and simulate this code in ICARUS VERILOG and GTKWAVE. Note that there is the one test bench `TB_gdlatch.v` for both modules - you’ll need to comment out the unused instantiation. Compare the operation of the above two descriptions. What do you conclude?

1.5 The D Flip-flop

Finally we can produce the circuit described by the code of the sequential circuit in Fig. 11

The D type flip-flop shown in Fig. 8
Figure 8: The D-type flip-flop or positive edge triggered Master-slave D flip-flop

For a positive-edge triggered master-slave D flip-flop, the first D latch is transparent during a high enable, and the second D latch is transparent during a low enable. Thus the full D flip-flop is never fully transparent. When the enable goes from low to high (0 to 1), the D input goes through the first (master) latch to the second (slave) latch. When the enable drops back to low (1 to 0), the output of the master latch is "locked", and the slave latch is transparent. When the enable goes back high (0 to 1), the slave latch will lock, and thus preserve the output until the next strobing of the enable (or clock).

CLEAR and PRESET can be used to force the output of the DFF to 0 or 1, regardless of the clock or D input values. When PRESET is LOW, Q is set to 1. When CLEAR is cleared or forced to 0. Note that CLEAR and PRESET CANNOT be forced to 0 at the same time.
Exercise 5: Try to understand what is going on in the circuit of Fig. 8. Draw a timing diagram to describe its operation. Explain how edge triggering occurs.

Exercise 6: Write VERILOG code using 'ASSIGNS' which describes the circuit of Fig. 8. Compare it to simulation output of the sequential circuit in Fig. 1. Does it work? What do you suggest to fix it?