

# Experiment #1: Random Sequence Counter

The A-Team  
Steve Comer  
Stuart Larsen

Implementation Due: 10 Sept 2012  
Report Due: 17 Sept 2012

---

Steven Comer

---

Stuart Larsen

## Table Of Contents

Introduction.....	3
Method.....	3
Results.....	9
Conclusion.....	9
Appendix A.....	11
Appendix B.....	12

## 1.0 Introduction

The first lab involved creating a 3-bit binary counter. The circuit needed to count a random string of numbers from 0 to 7, and repeat these numbers infinitely (or until power-off). The circuit needed to have a synchronous reset, and needed to be implemented using combinational logic along with 3 D Flip Flops (DFF). The output of the circuit needed to be displayed on 3 LEDs.

## 2.0 Method

### 2.1 Necessary Background Information

There are two main types of circuitry: combinational and sequential. Combinational logic circuits do not involve memory. Electricity flows like a stream through them. So once the input on one side changes and the clock ticks, the whole state updates like an equation. Sequential logic circuitry involves memory such as flip flops and latches which store the state of the system in memory. It is important to note that in combinational logic, a given input always produces the same output. The same is not true for sequential logic.

For our assignment, we used both combinational and sequential logic. The three flip flops made up the sequential logic component, while the logic block used to determine the next state was purely combinational.

Logic gates form the building blocks for combinational logic. The principal logic gates include the following: AND, OR, NOT, NAND, NOR. The diagram below (Figure 2.1) shows the graphical symbol, algebraic equation, and truth table for each of the major logic gates. Boolean expressions, those which evaluate to either "True" or "False", can be expressed by a combination of logic gates. Simple logic structures form the basis for more complex expressions which make a circuit diagram more readable.



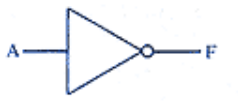


Name	Graphic Symbol	Algebraic Function	Truth Table															
AND		$F = A \cdot B$ or $F = AB$	<table><tr><th>A</th><th>B</th><th>F</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	A	B	F	0	0	0	0	1	0	1	0	0	1	1	1
A	B	F																
0	0	0																
0	1	0																
1	0	0																
1	1	1																
OR		$F = A + B$	<table><tr><th>A</th><th>B</th><th>F</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	A	B	F	0	0	0	0	1	1	1	0	1	1	1	1
A	B	F																
0	0	0																
0	1	1																
1	0	1																
1	1	1																
NOT		$F = \bar{A}$ or $F = A'$	<table><tr><th>A</th><th>F</th></tr><tr><td>0</td><td>1</td></tr><tr><td>1</td><td>0</td></tr></table>	A	F	0	1	1	0									
A	F																	
0	1																	
1	0																	
NAND		$F = (\overline{AB})$	<table><tr><th>A</th><th>B</th><th>F</th></tr><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table>	A	B	F	0	0	1	0	1	1	1	0	1	1	1	0
A	B	F																
0	0	1																
0	1	1																
1	0	1																
1	1	0																
NOR		$F = \overline{(A + B)}$	<table><tr><th>A</th><th>B</th><th>F</th></tr><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table>	A	B	F	0	0	1	0	1	0	1	0	0	1	1	0
A	B	F																
0	0	1																
0	1	0																
1	0	0																
1	1	0																

Figure 2.1: Combinational Logic Gates ([http://bmet.wikia.com/wiki/Logic\\_Gates](http://bmet.wikia.com/wiki/Logic_Gates))

Logic can be represented in many different forms, as seen by the table (Figure 2.2) below. The graphical symbol of each logic gate presents the simplest way to view the gates. Ports A and B represent the input to the logic gate, where port F represents the output of the logic gate. The algebraic function describes how the logic block actually works. The algebraic function is represented in Boolean algebra. The truth table shows all permutations of the input variables, and the corresponding output for each logic gate.

To create the circuit for this lab, a state transition diagram needs to be created. It takes 7 steps to get from the state transition diagram, to the circuit, or from the circuit to the state transition diagram. The 7 steps are as follows:

Table 1: Steps to Create State Diagram

Step	Name	Description
1	I.D. FF	Find the flip flops on the circuit.
2	Char. Eqn.	Write out the characteristic equations for D0 and D1.
3	Trans. Table	Map the input and outputs between states on the transition table.
4	I.D. Z	Figure out the output to each state
5	3+4	Combine steps 3 and 4 onto the same table
6	Name States	Give each unique state an name
7	State Trans.	Draw each as a node, and show the transition between each node depending

	Diagram	on the input and previous state
--	---------	---------------------------------

There are two main types of state machines: Mealy and Moore. The difference between the two lies in the output. In a Moore implementation, the output is based solely on the state, and not on the input. In a Mealy implementation, the output depends on the state as well as the input. Because of this, the state diagrams for each are notably different. The two images (Figures 2.3 & 2.4) below demonstrate the difference. The left diagram shows the state transition for a Mealy machine. In a Mealy machine, both the input and output are shown on the transition arrows. The right diagram shows the state transition for a Moore machine. The output value of the circuit is defined inside each state, with the input value still located on the transition arrows.

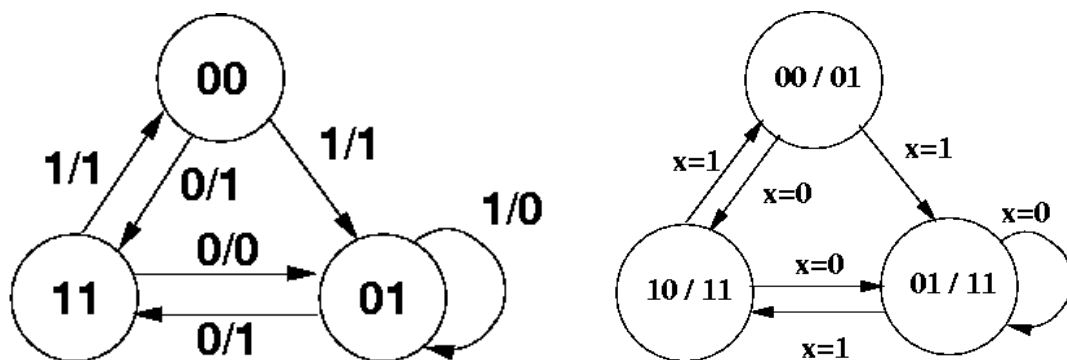


Figure 2.3 & 2.4: Mealy(left) and Moore(right) State Transition Diagrams.

(<http://www.cs.umd.edu/class/spring2003/cmsc311/Notes/Seq/fsm.html>)

The final piece of background information regards Karnaugh Maps, or K-Maps. K-Maps are used to simplify algebraic equations into simpler solution sets. The idea behind K-Maps is that generally the solution set to a problem is a group of specific cases that set the output to a certain state. The easiest way to represent this is to describe each case where the solution is true in their complete form. For example, let us assume that a captain of a baseball team needs to choose who he wants on his team. The captain judges the team based on the following attributes: { athletic, smart, determined, extroverted }. Let us assume that he only wants team members that are athletic and smart OR athletic and determined OR athletic and extroverted. By inspection it is deducible that the captain only wants members who are at least athletic. K-Maps simplify this process by putting each input value into a value, with the corresponding output value for each input set and allow you to see the relation between input states.

Below is an example K-map. The labeled inputs on the sides demonstrate the relation between input states. The diagram displays the changes between states in gray code. In gray code, the values between steps only switch one bit at a time. That's why the values change from 00 to 01 to 11 to 10, because in each step, only one bit is flipped. If the steps had gone 00 01 10 11, there would be two bit flips between the second and third steps, and so finding relationships between the steps would be harder.

To find the simplest solution to a K-map, circle all groups of one (if using a sum of products approach). As shown below, each group of 1's is circled. The circles can extend past the sides and wrap to the other side. Once that is done, determine the input terms that compose of the circles and sum them together to get the output. The solution for the below K-map is  $f = ABC + \sim C \sim D + \sim B \sim C$ .

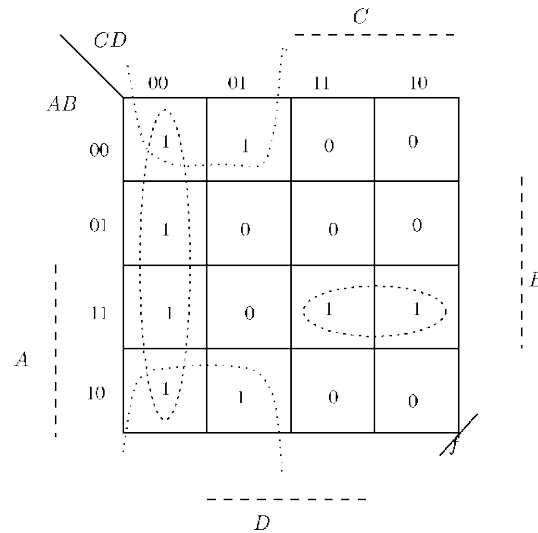
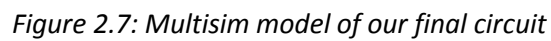


Figure 2.5: Example K-map (<http://users.cecs.anu.edu.au/~Matthew.James/engn2211-2002/notes/diglognode13.html>)

## 2.2 Procedure

Professor Vestal provided us with a sequence of random numbers at prior to starting the lab. From this we constructed the following state transition diagram. We recognized the diagram as Moore-type due to the fact that the output did not depend on the input. "Reset" provided the only input, and the current state determined the output, or next state.



After the simulation was build, we started building the circuit. We laid each piece out on the board and started connecting the pieces. Once the circuit was built, we created test cases and passed with flying colors.

### 2.3 Errors

During the design phase of the experiment we encountered a few problems. On Multisim we directly fed our function generator to an LED, and then to the clock pin on the Flip flop. The input clock pin on the Flip Flop has infinite impedance. Because of this no current was passing though the wire and the LED did not flash with each cycle of the clock. To solve this we moved the LED in parallel with the clock pin on the Flip Flop. The next problem we encountered dealt with a misconception of the assignment instructions. We didn't know that the circuit had to start with  $R = 1$ , because of this we spent time trying to figure out a way to make the state start at 3 without any previous input. The solution was simply to let  $R = 1$  at startup.

### 2.4 Tools Used

Only a few tools were used for the lab. Namely the computer for Multisim, the oscilloscope for debugging signals. The function generator generated our clock input. And lastly the logic analyzer was used to test our circuit.

## 3.0 Results

### 3.1 High Speed Operation

Although the function generator we used was unable to provide it, there was a limit to the possible Clock speed used in the circuit. This limit is based on the slowest, or critical, path in the circuit. The limit is calculated using the manufacturer-provided propagation delay for each chip that we used. The delays given assume a 5V input voltage and ideal conditions. For this reason, the theoretical delay calculated may not be the same as one determined by precise measurement of the operational circuit.

Chip	Part #	Delay (5V input)
Triple 3-input NOR	4025	60 ns
Quad 2-input OR	4071	45 ns
Quad 2-input AND	4081	45 ns
Quad 2-input XNOR	4077	110 ns
Hex Inverter	4069	40 ns
D Flip Flop	74LS175	20 ns

*Figure 3.1: Delay times for each Integrated Circuit chip (See Appendices for source)*

Figure 3.1 shows the propagation delay for each chip used in the circuit. Figure 2.7 shows the implementation of the circuit. It is important to note that the 3-input OR gate shown in the circuit diagram was implemented using two 2-input OR gates. Based on these sources, we



determined that the critical path of the circuit was the one leading to  $D_0$ , the flip flop closest to the bottom of the diagram. This path includes 3 OR gates, 1 AND gate, and 1 Flip Flop. Using Figure 3.1, the delay for this path is:  $3 \cdot 45 \text{ ns} + 45 \text{ ns} + 20 \text{ ns} = 200 \text{ ns}$ . Therefore, the Clock cycle time must be greater than 200 ns for the circuit to operate correctly. Inverting the cycle time provides the maximum Clock frequency, which is  $(1/200\text{ns})$  or 5.0 MHz. This proves that the theoretical delay is not the same as the measured delay because the circuit operated correctly at exactly the Clock frequency that theoretically should break it.

### 3.1 Other Observations

Another observation we made also relates to high speed operation. Although we used the reset input to determine that the circuit was operating properly at 5.0 MHz, the LEDs appeared to be constantly on. This demonstrates an important point. A very fast circuit can be constructed, but a fast circuit alone is not helpful if outside hardware (i.e. the LEDs) limits the usability of the device.

### 3.2 Testing Procedure

The testing procedure is important because it demonstrates the full capability of the circuit and proves that it works correctly for all input combinations. The steps of our test plan are detailed in the following list. We added an extra LED to the circuit to display the Clock state (LED on if Clock is high). The circuit was powered using a 5V DC power supply.

1. Power on circuit with reset off. Wait for one full sequence.
2. Power on circuit with reset on. Keep reset on for multiple clock cycles to show that reset persistent. Turn reset off. Wait for one full sequence.
3. Power on circuit with reset off. Turn reset on when circuit reaches second state. Wait for one full sequence.
4. Repeat step 3 for all remaining states, resetting at the next successive state each time before resetting and waiting for one full sequence.

### 3.3 Results

The experiment was successful. Our circuit performed correctly for all possible input combinations. The circuit actually exceeded expectations because the function generator used for testing was unable to produce a Clock speed fast enough to break our circuit. The maximum frequency for the function generator was 5.0 MHz. The theoretical Clock frequency required to prevent the circuit from functioning properly will be discussed in the following subsection.

## 4.0 Conclusion

The objective of this experiment was to implement a practical example of synchronous sequential logic, namely a random sequence counter. The goal was to build a 3-bit random sequence "counter". The

counter had 8 states, equating to the decimal values 0 through 7. This counter differed from an ordinary 3-bit counter in the sequence of the numbers, which in this case was a random permutation of the decimal values 0 through 7. Our specific sequence was [3, 1, 6, 4, 2, 0, 7, 5]. We successfully implemented the random sequence counter on our breadboard using six different Integrated Circuit (IC) chips and three D Flip-Flops (DFF). We exceeded the expectations in that our circuit worked for the maximum frequency provided by the function generator. We learned how to use Multisim to test our design prior to physical implementation. We also learned that it is important to thoroughly test a design before constructing it. In the case of the propagation delays and high speed operation, we learned that theoretical calculations often differ from real measurements.

## A. Appendix A

*Table 1: State Transition Table*

[illegible]

Comer, Larsen

## **B. Appendix B**