Digital Design Lab
ECN 315

Lab/Project
Lab 1A - Sequence Generator

Istvan von Fedak
(Partner: Jonathan Brooks, Josh Stoller)

May Zar Lin, TA

University of Miami
Jan 25 2017

Abstract

- The purpose of this lab was to design and implement a a sequence generator that would generate a sequence which had a minimum of 10 different states. We chose to implement the following sequence: 0, 6, 3, 5, 2, 15, 10, 14, 9, 12. Furthermore, our circuit had to contain a zero state and ensure that all invalid stages of the machine clear it back to zero. This is important in a design because it stabilizes the sequence — if there is a glitch in the sequence, it will always reset to its initial state.
- We implemented our circuit using two 7474 Dual Positive Edge Triggered D-FF, four 74151 8:1 multiplexers, a breadboard (provided by the TA), the necessary cables to connect the circuit and the ETS-700A Digital-Analog training System.
- After building and debugging the circuit, we were able to create a sequence generator with the desired sequence. Our group also gained a sound understanding on how to use breadboards, how to connect the circuits properly and how to implement proper cable management.
- Overall, a depth of understanding was acquired on logic design concepts. Building circuits using prebuilt micro-chips, understanding how to efficiently design prototypes on a breadboard and having a hands on experience were useful acquired skills within this lab.

Table of Contents

Overview

In order to design a sequence generator, on must go through various design steps. In this lab, we started with the state diagram. We listed our ten possible stages our sequence had. After having a clear picture of the different states in the sequence generator, we used a next state table to list all of the possible states in our circuit — making sure to list the state following the current state. Once all of the possible states were properly mapped, we converted the next state table into the transition table. This table is a binary representation of the next state table. Since the largest number in our sequence was 15, we needed to have 4 binary weights to represent each number. We decided to name these weights, A, B, C and D — D being the most significant digit. Once the transition table was properly filled in, we created the k-maps for the 4 multiplexers we were implementing — a multiplexer is a device that selects one of several analog or digital input signals and forwards the selected input into a single line. Once we used the k-maps to determine the input signals for our multiplexers, we started to build the circuit using both the 4 multiplexers and the 4 D flip flops on the provided breadboard. In order to facilitate the implementation of the circuit, we color coded the wires. Red was used for high voltage current (VCC), black was used for ground (GND), yellow was used for the clock(CK), blue was used to represent A, white was used to represent B, orange was used to represent C, light green was used to represent D and dark green was used to represent D complement (D').

Objectives

The main goal of this lab was to designing and implementing a sequence generator that would cycle through 10 different numbers — including zero ensured that all invalid stages restarted the sequence. Another thing that was gained from this lab was a sound understanding of the algorithmic implementation of logic design.

Equipment

Description	Chip Number	Quantity
Dual Positive Edge Triggered D-FF	7474	2
8:1 Multiplexer	74151	4

Description

When first designing a sequence generator, one must start with creating a state diagram. It is fairly simple, we started by listing the sequence we wished to generate (0, 6, 3, 5, 2, 15, 10, 14, 9, 12). Since our sequence's largest number was 15 we had to use 4 bits to represent the number

in binary. Once that was done, we converted the state diagram into the next state table. This table had all of the 16 possible combinations one could represent with 4 bits. We decided to name the four bits/weights A, B, C and D — D being the most significant digit. Because our circuit only used 10 numbers we had 6 numbers that we did not use, we were required to set their next state to zero. This would stabilize the circuit in the event of a glitch. After fully filling out the next state for all possible combinations, we moved our data to the transition table. This table is a binary representation of the next state table — it is a tool for filling in k-maps. Since we had to use 4 multiplexers — one representing each weight — that fed their output to 4 D flip flops, to create our sequence generator, 4 k-maps were used to determine both the select lines and input lines of each multiplexer. In other words, we used the k-maps to determine the inputs of each multiplexer. A, B, C were used as select lines and D, VCC, GND were used as input lines. Once all of the inputs for the multiplexers were known, we created a circuit diagram to help us map the layout of our sequence detector. Having the correct circuit diagram allow us to started building the sequence generator. We used the following materials provided by the lab: four 74151 micro-chips, two 7474 micro-chips, one breadboard, one wire cutter, and eight different colored wires. Each wire color corresponded to a specified signal (see the overview for more information on wire color-coding). After having the necessary materials we started building. We first connected the chips to the breadboard. We decided to have the multiplexers in center of the board so that it was easier to manage all of the select lines. The D flip flops were purposefully placed on either side, in order for them to have closer access to the multiplexers in the center — facilitating cable management. Then, the following connections were wired in chronological order on all chips: GND, VCC, CK, D, D', A, B and C. Since our circuit had good cable management, debugging only took a couple of minutes. Overall, designing and building the circuit was an insightful experience.

Specifications

We had to ensure that all invalid stages of the machine clear it back to zero.

Design Synthesis Or VHDL Code

Sequence: 0, 6, 3, 5, 2, 15, 10, 14, 9, 12, 0, 6, …

Transition Table:

PS	D	C	B	A	NS	QD	QC	QB	QA
0	0	0	0	0	6	0	1	1	0
1	0	0	0	1	0	0	0	0	0
2	0	0	1	0	15	1	1	1	1
3	0	0	1	1	5	0	1	0	1
4	0	1	0	0	0	0	0	0	0
5	0	1	0	1	2	0	0	1	0
6	0	1	1	0	3	0	0	1	1
7	0	1	1	1	0	0	0	0	0
8	1	0	0	0	0	0	0	0	0
9	1	0	0	1	12	1	1	0	0
10	1	0	1	0	14	1	1	1	0
11	1	0	1	1	0	0	0	0	0
12	1	1	0	0	0	0	0	0	0
13	1	1	0	1	0	0	0	0	0
14	1	1	1	0	9	1	0	0	1
15	1	1	1	1	10	1	0	1	0

A flip flop:

CBA	000	001	010	011	100	101	110	111
0	0	0	1	1	0	0	1	0
1	0	0	0	0	0	0	1	0
QA =	0	0	D'	D'	0	0	1	0

B flip flop:

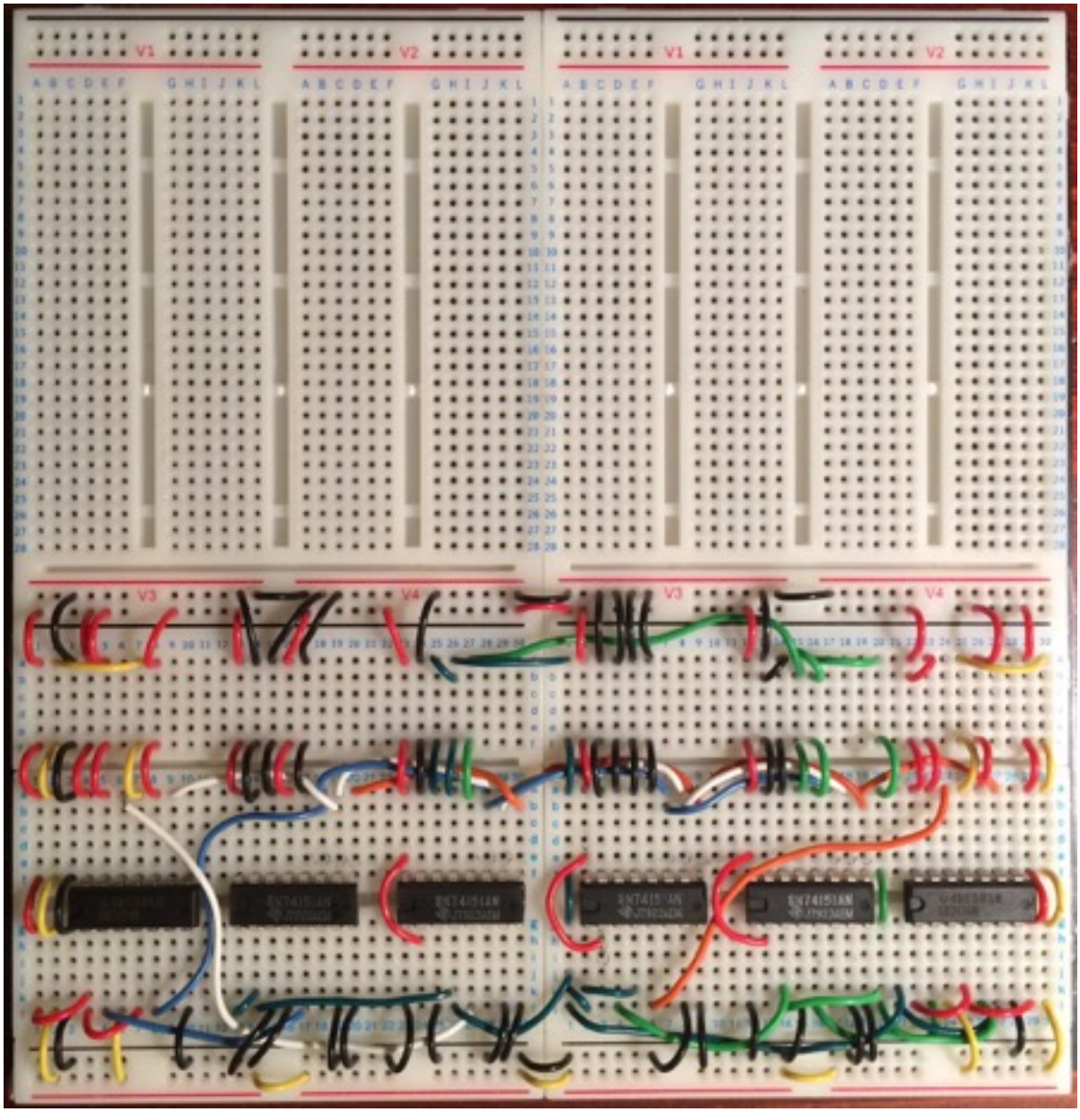
CBA	000	001	010	011	100	101	110	111
0	1	0	1	0	0	1	1	0
1	0	0	1	0	0	0	0	1
QB =	D'	0	1	0	0	D'	D'	D

C flip flop:

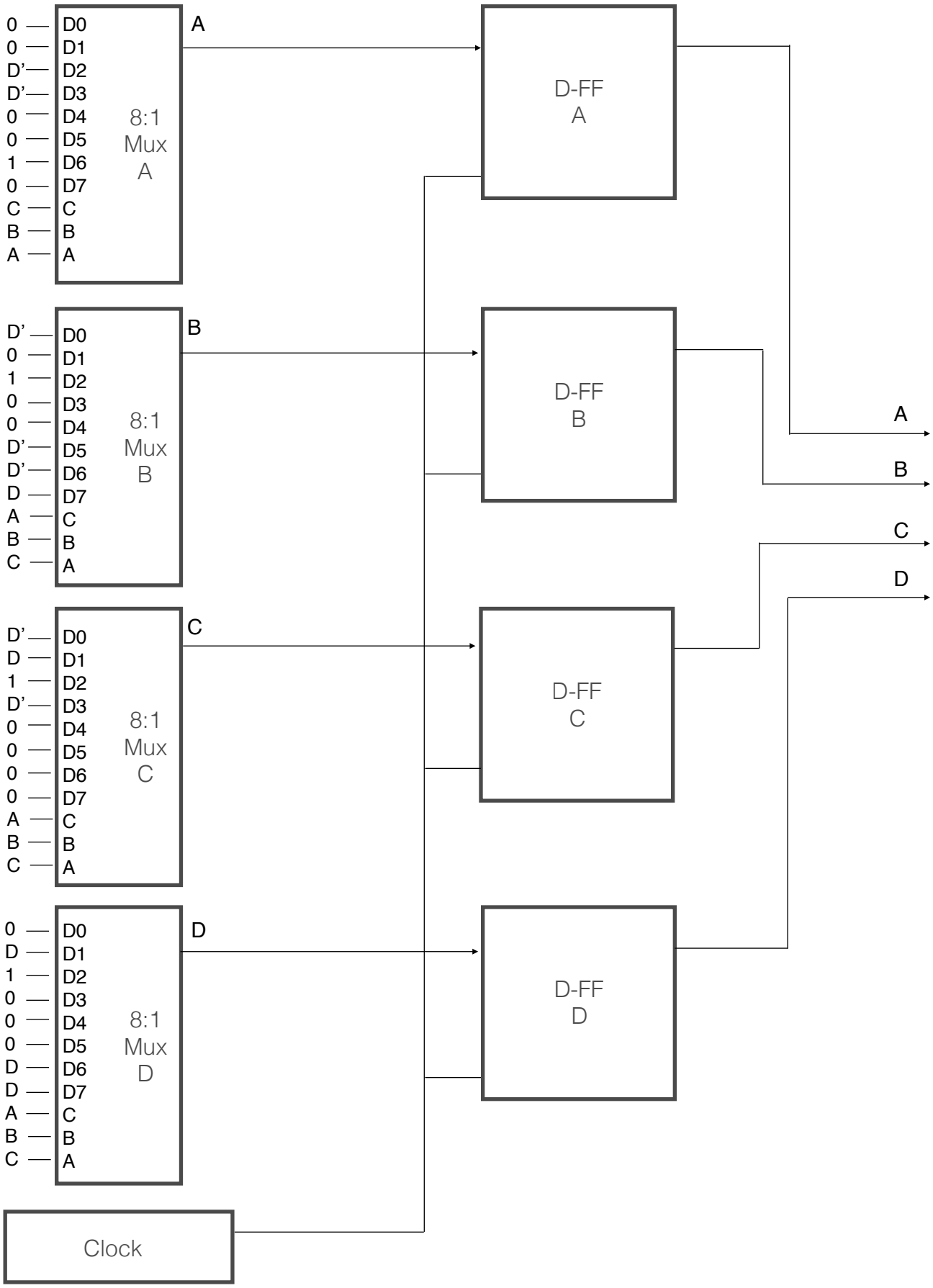
CBA	000	001	010	011	100	101	110	111
0	1	0	1	1	0	0	0	0
1	0	1	1	0	0	0	0	0
QC =	D'	D	1	D'	0	0	0	0

D flip flop:

CBA	000	001	010	011	100	101	110	111
0	0	0	1	0	0	0	0	0
1	0	1	1	0	0	0	1	1
QF =	0	D	1	0	0	0	D	D



Complete Logic Diagram



Results and Simulations

The results we obtained were as expected. We were able to generate a sequence generator with 10 states using logic design.

Conclusion

a. Our sequence generator was designed to have 10 different states. Any invalid stages would be set back to zero — stabilizing the sequence. We implemented our circuit using the following algorithmic steps: designing state diagram, implementing the next state table — having all possible combinations, mediate the information to the transition table, filling in the k-maps to determine the multiplexers select lines and input lines, creating the circuit diagram and building the circuit.
b. We encounter one problem when building the sequence generator. The input for one of the lines that belonged to the C multiplexer was not connected to GND, this caused the sequence to go to an invalid state. Once the error was corrected, the circuit followed its intended sequence.
c. One of the biggest limitation in our design had, was the use of a breadboard. Although they are a near perfect tool for prototype design, it had small layout flexibility.
d. I cannot stress this point enough! This lab served as a good refresher for the fundamental concepts of logic design.

Works Cited

https://www.courses.miami.edu/bbcswebdav/pid-7401813-dt-content-rid-10834491_1/courses/ECE315-E1-05939-1-20171/LAB1-A%20Sequence%20Generator.pdf