Digital Design Lab
ECN 315

Lab/Project
Lab 1 - Sequence Generator & Tone Generator

Istvan von Fedak
(Partner: Jonathan Brooks)

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, having a minimum of 10 different states, that would be converted into a sequence of different tones. The following sequence was implemented: 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 was important in the design because it stabilizes the sequence — if there is a glitch with the clock, 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, one 555 timer, one 74LS169 Modulo-N Counter, one breadboard (provided by the TA), one JK 7476 Flip Flop, the necessary cables needed 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. Once we implemented our sequence, we modified our circuit to generate a progression of different tones.
- Overall, a depth of understanding was acquired on logic design concepts. We gained a sound understanding on the use of breadboards, building circuits using prebuilt microchips, planning the most efficient way of placing components, creating a counter, calculating the proper value of resistors, understanding how frequency works, and performing adequate cable management.

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 current and next states in our circuit. Having everything properly mapped, we converted the next state table into the transition 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. We then created the k-maps for the 4 multiplexers we had to implement. After finishing the initial design, 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 A, white was B, orange was C, light green was D and dark green was D complement (D'). Once we had a working sequence generator, we started the design of the tone generator. We used a 555 timer in its stable state to generate a frequency of 16KHz with a 60% duty cycle. We then used a 74LS169 modulo N counter to achieve the following frequency:

$$F_{out} = \begin{cases} \frac{1}{2(N+1)} f_b \\ \frac{1}{2(16-N)} f_b \end{cases}.$$

Furthermore, we converted a 74LS76 J-K flip-flop into a T flip-flop and used it to divide the frequency output of the modulo N counter by two. By connecting the output of the J-K flip-flop, passing through a $47\mu F$ capacitor, to the speaker we were able to listen all the tones we were generating.

Objectives

The main goal of this lab was to designing and implementing a sequence generator and tone generator that would cycle through 10 different states — including zero ensured that all invalid stages restarted the sequence. This lab also served as an introduction to the algorithmic implementation of circuits using logic design.

Calculations

Given:

F_b =16kHz	[Frequency generated by 555 timer]
Duty cycle at 60%	[Percent time of clock at hight state (1)]
C = **10*nF***.	[Capacitor used by 555 timer]

Equations:

$T = 1/\text{frequency} = 62.5\mu s$	[Period of time between the high and low of the clock]
$t_H = (\text{Duty cycle}/100)*T = 37.5\mu s$	[Period of time clock is at high state(1)]
$t_L = T- tH = 25.0\mu s$	[Period of time clock is at low state(0)]
$R_B = t_L/(0.693*C) = 3.6k\Omega$	[Resistor B used by the 555 timer]
$R_A = (t_H/(0.693*C)) - R_B = 1.8k\Omega$	[Resistor A used by the 555 timer]

Equipment

Description

Description	Chip Number	Quantity
Dual Positive Edge Triggered D-FF	7474	2
8:1 Multiplexer	74151	4
Timer	555	1
Modulo-N Counter used a frequency generator	74LS169	1
JK Flip Flop used as a frequency divider	7476	1
Digital-Analog training System	ETS-700A	1

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 — stabilizing 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 allowed 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.

After creating the sequence generator, we started to design the circuit that would convert it to a tone generator. We decided to use the upper left quadrant of the bread board so that it was closer to the speaker on the Digital-Analog training System. The 555 timer was the first built component, making sure that it was in its stable stage. The output of sequence generator and used it as the input for the Modulo-N counter. We then used the output of the carry out pin in the 74LS169 chip as the baseline for our initial frequency. Passing it though a modified J-K flip-flop that acted as a T flip-flop to reduce the output frequency in half. We ran the output of the flip-flop through a 47 micro ohm capacitor and the speaker. The end result was a tone generator that would have 10 different states. Overall, designing and building the circuit was an insightful experience.

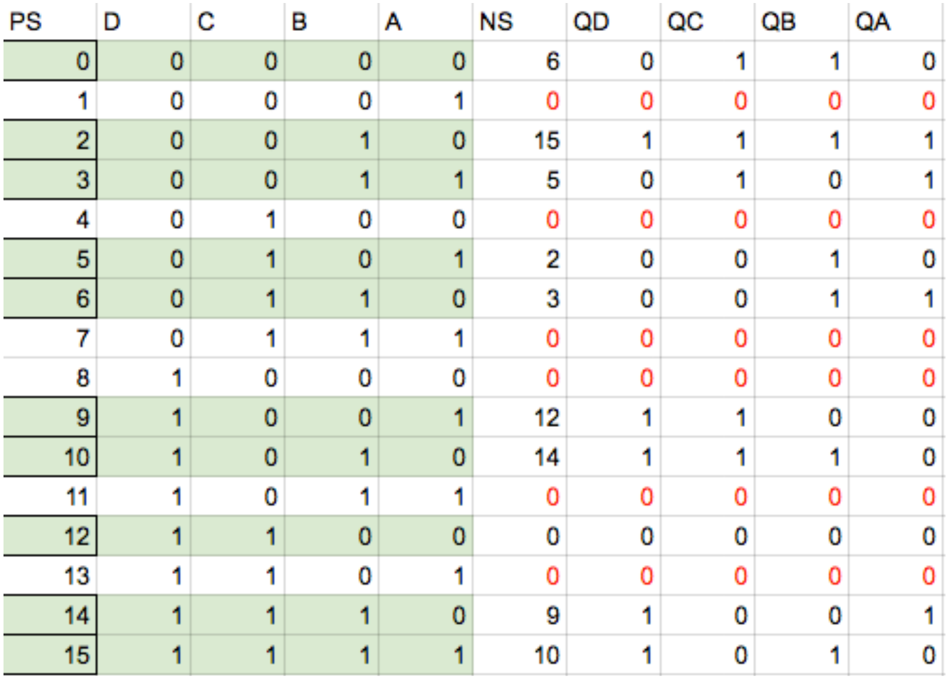
Specifications

We had to ensure that all invalid stages of the machine clear it back to zero. The 74LS169 as a frequency generator instead of a Modulo-N counter. And the J-K flip flop was repurposed to be used as a T flip-flop and divide the frequency in half.

Design Synthesis Or VHDL Code

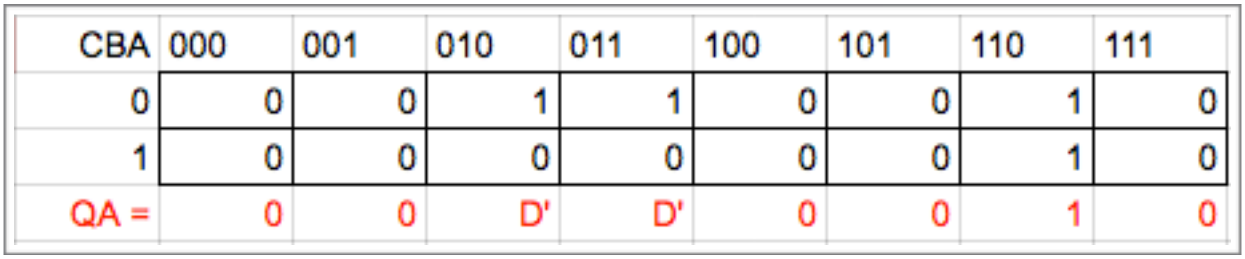
Sequence: 0, 6, 3, 5, 2, 15, 10, 14, 9, 12, 0, 6, …

Transition Table:



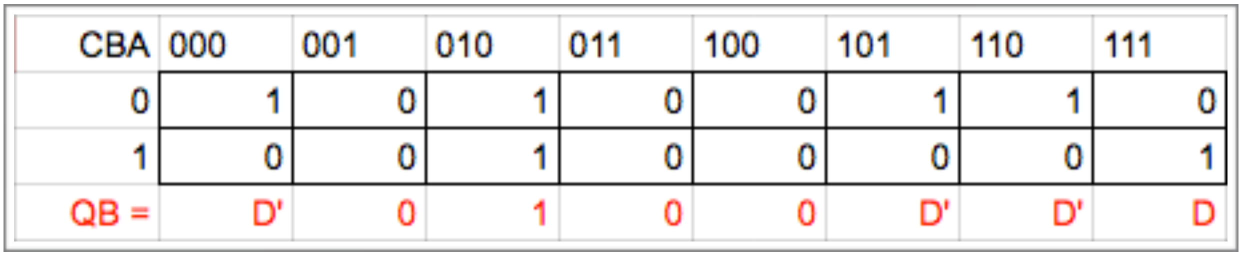
PS	D	C	B	A	NS	QD	QC	QB	QA
0	0	0	0	0	6	0	1	1	0
1	0	0	0	1	0	0	0	0	0
2	0	0	1	0	15	1	1	1	1
3	0	0	1	1	5	0	1	0	1
4	0	1	0	0	0	0	0	0	0
5	0	1	0	1	2	0	0	1	0
6	0	1	1	0	3	0	0	1	1
7	0	1	1	1	0	0	0	0	0
8	1	0	0	0	0	0	0	0	0
9	1	0	0	1	12	1	1	0	0
10	1	0	1	0	14	1	1	1	0
11	1	0	1	1	0	0	0	0	0
12	1	1	0	0	0	0	0	0	0
13	1	1	0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



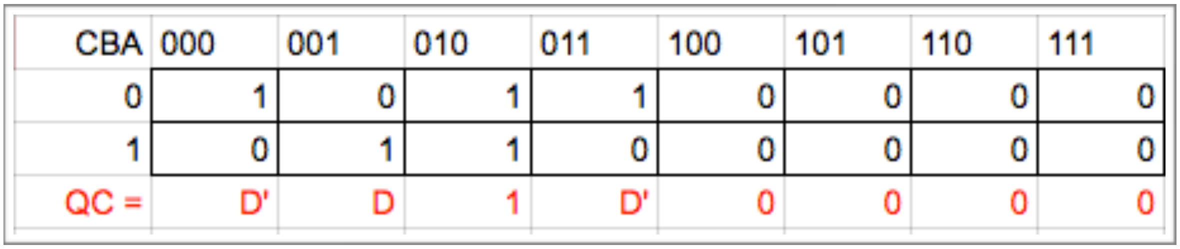
CBA	000	001	010	011	100	101	110	111
0	0	0	1	1	0	0	1	0
1	0	0	0	0	0	0	1	0
QA =	0	0	D'	D'	0	0	1	0

B Multiplexer:



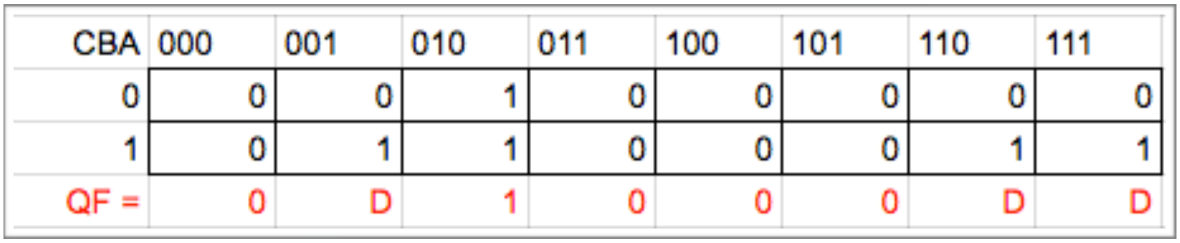
CBA	000	001	010	011	100	101	110	111
0	1	0	1	0	0	1	1	0
1	0	0	1	0	0	0	0	1
QB =	D'	0	1	0	0	D'	D'	D

C Multiplexer:

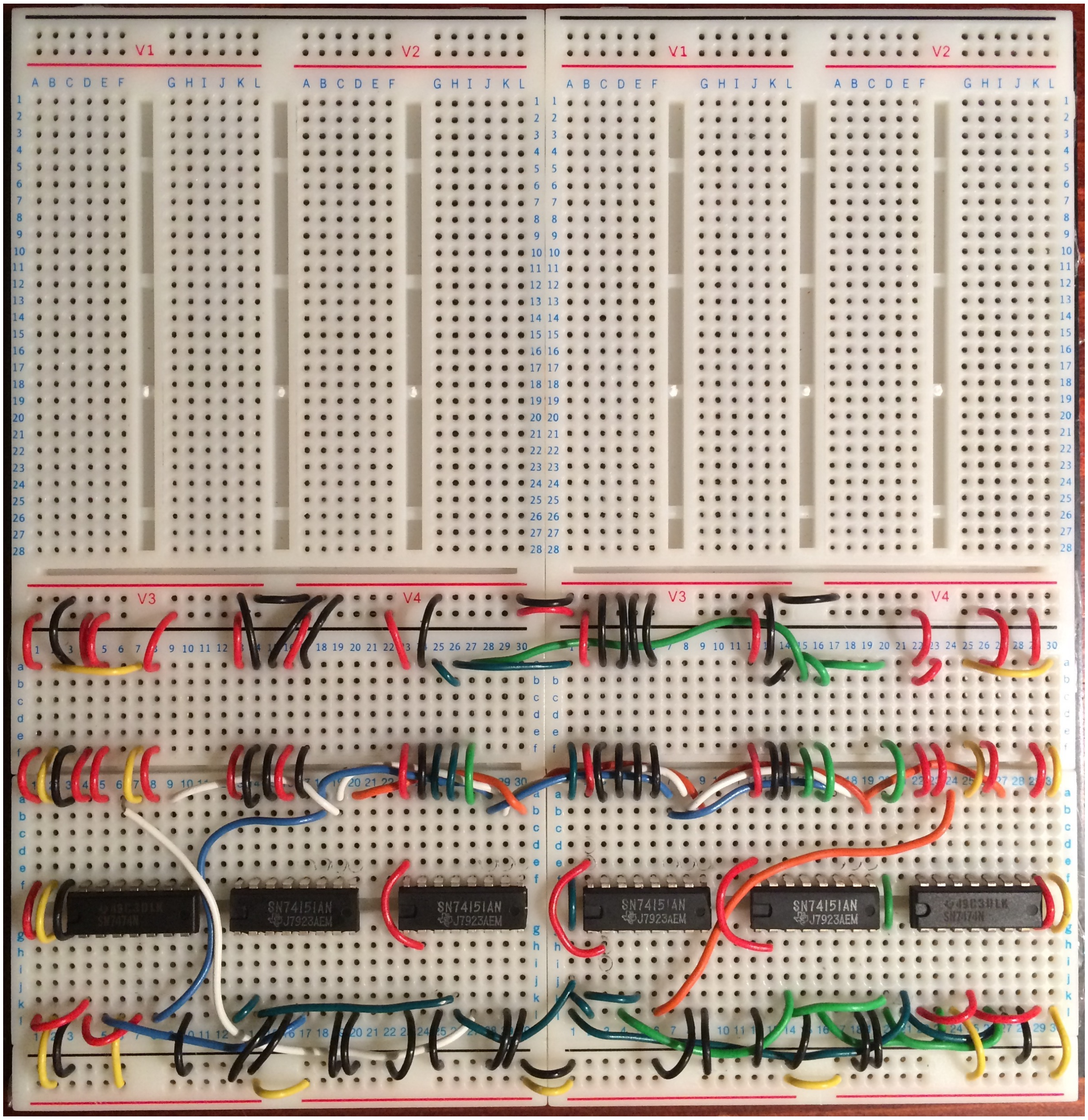


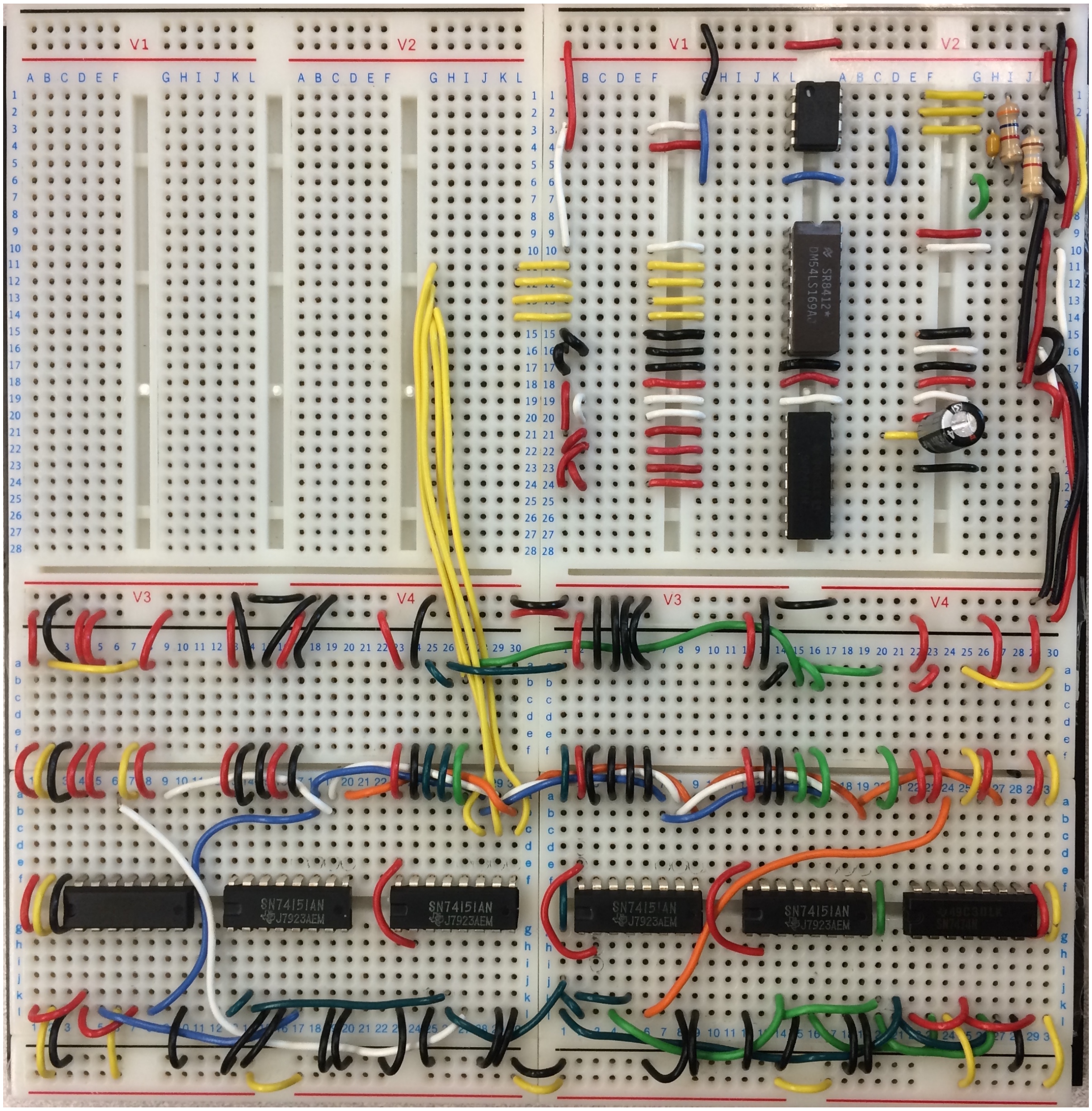
CBA	000	001	010	011	100	101	110	111
0	1	0	1	1	0	0	0	0
1	0	1	1	0	0	0	0	0
QC =	D'	D	1	D'	0	0	0	0

D Multiplexer:

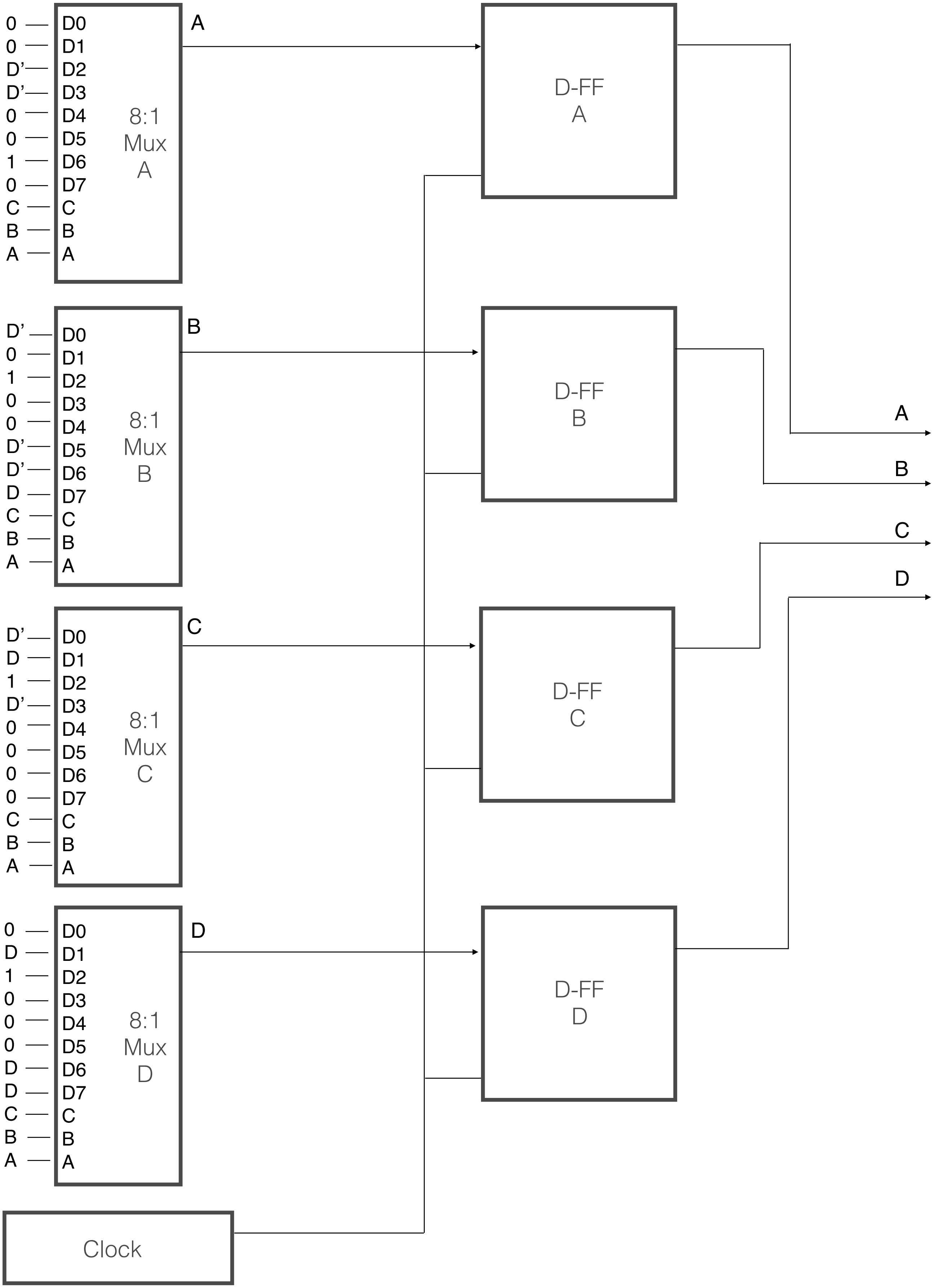


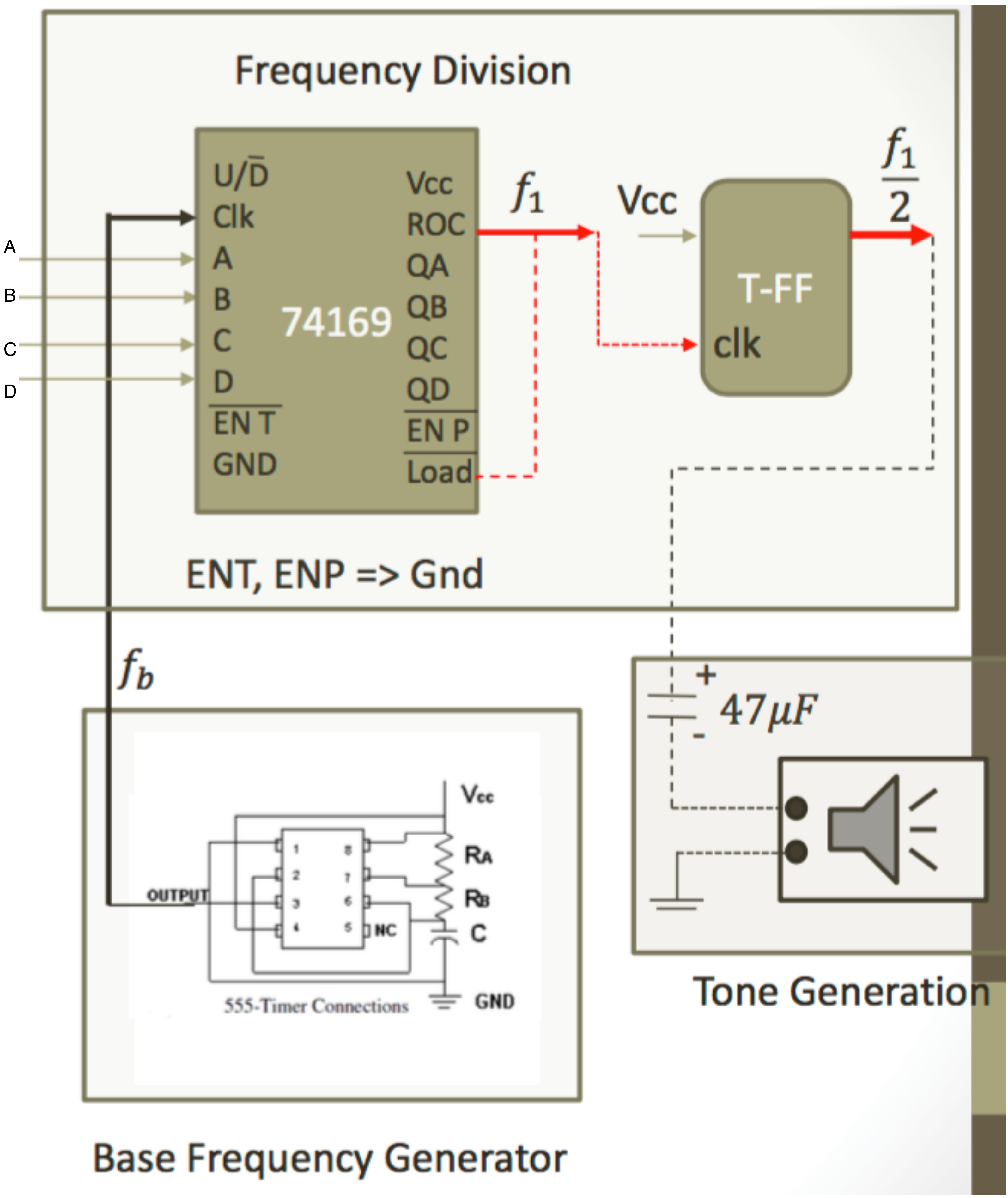
CBA	000	001	010	011	100	101	110	111
0	0	0	1	0	0	0	0	0
1	0	1	1	0	0	0	1	1
QF =	0	D	1	0	0	0	D	D




V1
V2
V1
V2
V3
V4
V3
V4
SR8412*
DM54LS169A
SN74151AN
J7923AEM
SN74151AN
J7923AEM
SN74151AN
J7923AEM
SN74151AN
J7923AEM

Complete Logic Diagram



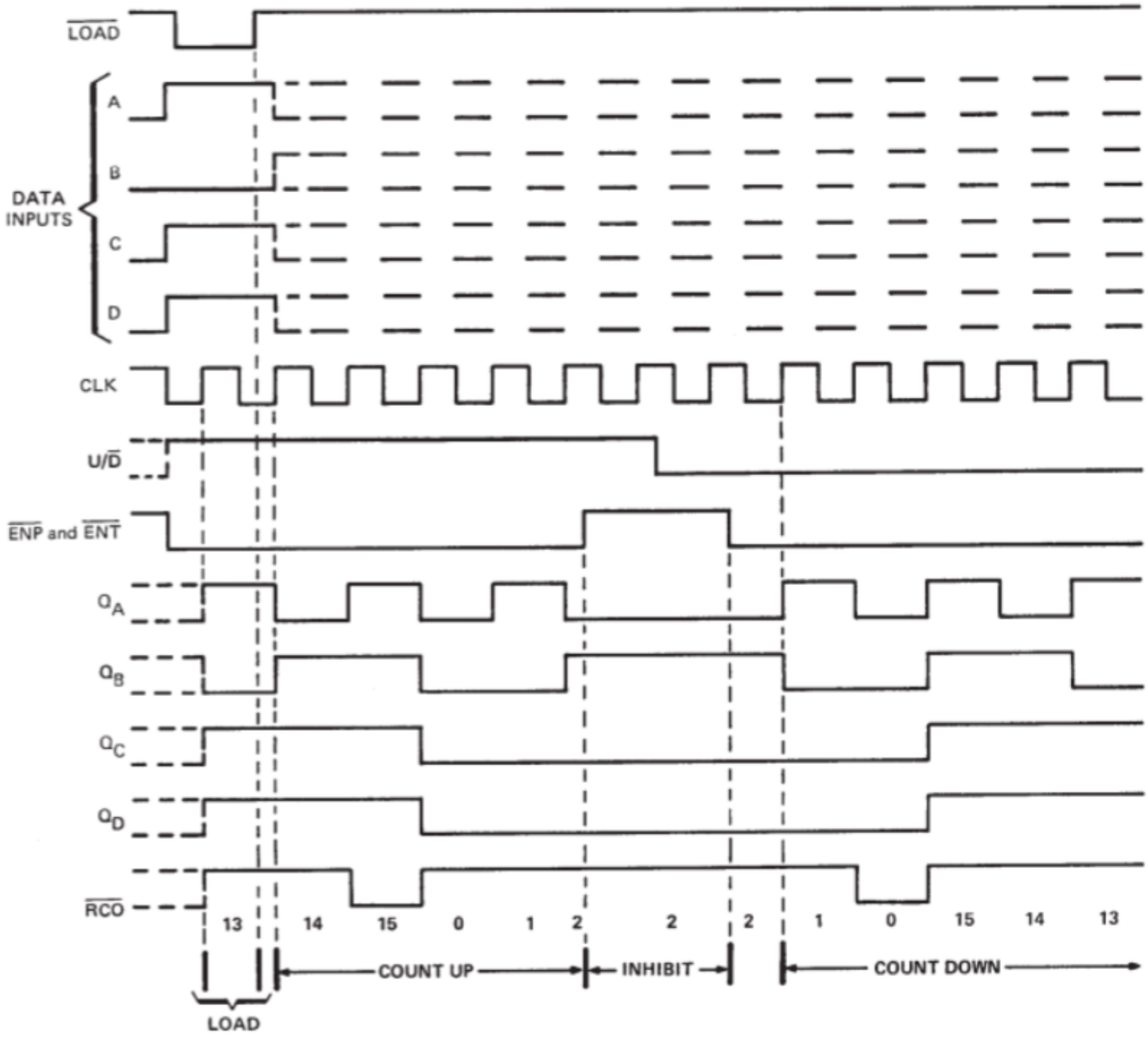


Frequency Division

ENT, ENP => Gnd

Base Frequency Generator

Tone Generation


LOAD
DATA INPUTS
A
B
C
D
CLK
U/D
ENP and ENT
Q_A
Q_B
Q_C
Q_D
RCO
13
14
15
0
1
2
2
2
1
0
15
14
13
COUNT UP
INHIBIT
COUNT DOWN
LOAD

Results and Simulations

The results we obtained were as expected. We were able to generate a sequence generator with 10 states using logic design. Furthermore, we were able to implement a timer and repurpose two micro-chips in order to convert our ten binary stages into different frequencies.

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. Once the sequence generator was implemented, we used the 555 timer to power a repurposed Modulo-N counter into a frequency generator. Dividing the frequency that came out of the carry out pin, though a frequency divider. This divider was composed of a modified J-K flip-flop that acted as a T flip-flop and decided the frequency by half. This frequency was then connected to speaker using an intermediary 47 micro farad capacitor.

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

https://www.courses.miami.edu/bbcswebdav/pid-7401813-dt-content-rid-10845609_1/courses/ECE315-E1-05939-1-20171/LAB1-B.pdf