CSC270—Circuits

Spring 2020—Week 5

Dominique Thiébaut
dthiebaut@smith.edu
Cartoon of the Day...

This is what learning logic gates feels like.

See, you just connect this 12 input reverse flip-flop to the controlled two-thirds adder, which resets the latches in the not-nand relay array, then loop back to odd-number inputs and reverse all your switches!

AND WHAT'S THAT DO?

SUBTRACTION.

Sent to us
By Prof. Kristen Dorsey

https://www.smbc-comics.com/comic/logic-gates
Feedback on Lab Reports

- “…as shown in Figure 1…” Use **upper case** when you refer to a figure number or table number.

- “…we measured **about** 5V…” Be precise and report the exact measurement! (see next slide)

- Do not forget to add pin numbers and part numbers to your diagrams.

- If you’re using a transistor explain how to identify the **base**, the **collector** and the **emitter**.

- Your report should have all the information needed to redo the lab.

- Be concise, thorough, and precise.
Template for Lab Reports

- Title
- List the **equipment used**
- Present **one section** for each experiment
  - What the **purpose** of the experiment is (1 to 2 sentences)
  - What the **setup** is (wiring diagram)
  - How the experiment is **conducted** (activate switches). You may show photos.
  - What the **observations** are. If quantities are measured, report on the values measured with the correct unit. If behavior is observed, show truth tables, or timing diagram, whatever is appropriate
  - **Summarize** the findings
About Voltage Measurements

Vcc can vary quite a bit!
• Minterms in Python
• Back to D-Flipflop (review)
• Exploring FSMs with D-Flipflops
• From word problem to FSM
• Exercise
• Moore vs Mealy Machines
def f( a, b, c ):
    '''f: boolean function of 3 variables'''
    return ( a and (not b) ) or c

def g( a, b, c ):
    '''g: boolean function of 3 variables'''
    return (not a) or (not b) or (not c)

def h( a, b, c ):
    '''h: boolean function of 3 variables'''
    return a or not b

def problem3( a, b, c, d ):
    '''problem3: boolean function from Homework 2, of 4 variables'''
    return not (a or b) and (not c) and (c or d)

print( "g(a,b,c) = Sigma( %s )"
       % ", ".join( [str(k) for k in minterms0f3(g)] ) )

print( "h(a,b,c) = Sigma( %s )"
       % ", ".join( [str(k) for k in minterms0f3(h)] ) )

print( "pb3(a,b,c,d) = Sigma( %s )"
       % ", ".join( [str(k) for k in minterms0f4(problem3)] ) )

main()
def f( a, b, c ):
    '''f: boolean function of 3 variables. Just pass the name of the function.'''
    return ( a and b ) or ( not a )

def g( a, b, c ):
    '''g: boolean function of 3 variables. Just pass the name of the function.'''
    return ( not a ) or ( not b )

def h( a, b, c ):
    '''h: boolean function of 3 variables. Just pass the name of the function.'''
    return a or not c

def problem3( a, b, c, d ):
    '''problem3: boolean function of 4 variables. Just pass the name of the function.'''
    return not ( a or not b ) or ( c and not d )

def mintermsOf4( func ):
    '''returns a list of minterms (as ints) for a boolean function of 4 variables. Just pass the name of the function.'''
    minterms = []
    for a in [ 0, 1 ]:
        for b in [ 0, 1 ]:
            for c in [ 0, 1 ]:
                for d in [ 0, 1 ]:
                    if func( a, b, c, d ) == 1:
                        minterms.append( a*8+b*4+c*2+d )
    return minterms

def main():
    print( "f(a,b,c) = Sigma( %s )" % , ".join( [str(k) for k in mintermsOf3(f)] ) )

    print( "g(a,b,c) = Sigma( %s )" % , ".join( [str(k) for k in mintermsOf3(g)] ) )

    print( "h(a,b,c) = Sigma( %s )" % , ".join( [str(k) for k in mintermsOf3(h)] ) )

    print( "pb3(a,b,c,d) = Sigma( %s )" % , ".join( [str(k) for k in mintermsOf4(problem3)] ) )

main()
def f(a, b, c):
    '''f: boolean function'''
    return (a and not b) or (a and b and c)

def g(a, b, c):
    '''g: boolean function'''
    return (not a) or (a and not b and c)

def mintermsOf4(func):
    '''returns a list of minterms (as ints) for a boolean function of 4 variables. Just pass the name of the function.'''
    minterms = []
    for a in [0, 1]:
        for b in [0, 1]:
            for c in [0, 1]:
                for d in [0, 1]:
                    minterms.append(func(a, b, c, d))
    return minterms

f(a,b,c) = Sigma( 1, 3, 4, 5, 7 )
g(a,b,c) = Sigma( 0, 1, 2, 3, 4, 5, 6 )
h(a,b,c) = Sigma( 0, 1, 4, 5, 6, 7 )
pb3(a,b,c,d) = Sigma( 1 )

print( "g(a,b,c) = Sigma( %s )"
    % ", ".join( [str(k) for k in mintermsOf3(g)] ) )

print( "h(a,b,c) = Sigma( %s )"
    % ", ".join( [str(k) for k in mintermsOf3(h)] ) )

print( "pb3(a,b,c,d) = Sigma( %s )"
    % ", ".join( [str(k) for k in mintermsOf4(problem3)] ) )

main()
Meet the D-Flipflop!
**Characteristic Table**

**Characteristic Tables**

A characteristic table defines the logical properties of a flip-flop by describing its operation in tabular form.

<table>
<thead>
<tr>
<th>Dt</th>
<th>Qt+1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Informal Exploration of an FSM (Automata)
What is the behavior of this circuit?
D-Flip-Flops in Computers

Exploring Finite State Machine with D-Flip-Flop

From Word Problem to FSM

Exercise

Moore vs. Mealy Machines
Can we do the opposite, start with a word problem, and end up with the schematics of the FSM?
Word Problem

- Design a Finite State Machine (FSM) that oscillates between 2 states, and outputs 1 in State 1 and 0 in State 0
D Flip-flop

<table>
<thead>
<tr>
<th>D</th>
<th>Qt+1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Exercise

Generate an FSM that controls 3 lights: Green, Yellow, and Red, and cycles through all 3, staying 1 second on Green, 1 second on Yellow, and 2 seconds on Red.

(Take good notes, you will have to redo all these steps soon, for a different problem)
4 States ==> 2 Flip-flops: Q1, Q0

<table>
<thead>
<tr>
<th>Tn</th>
<th>Tn+1</th>
</tr>
</thead>
<tbody>
<tr>
<td>S0</td>
<td>S1</td>
</tr>
<tr>
<td>S1</td>
<td>S2</td>
</tr>
<tr>
<td>S2</td>
<td>S3</td>
</tr>
<tr>
<td>S3</td>
<td>S0</td>
</tr>
</tbody>
</table>

\[
\begin{align*}
T_{n} & = \begin{bmatrix} 0 & 0 \\ 0 & 1 \\ 1 & 1 \\ 1 & 0 \end{bmatrix} \\
D_{1} & = Q_{0} \\
D_{0} & = Q_{1}'
\end{align*}
\]

\[
\begin{array}{c|c|c|c|c}
T_{n} & T_{n+1} & G & Y & R \\
\hline
Q_{1}Q_{0} & Q_{1}Q_{0} & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 \\
0 & 1 & 1 & 0 & 1 \\
1 & 1 & 0 & 0 & 1 \\
1 & 0 & 0 & 0 & 1 \\
\end{array}
\]
A very quick and dirty way to check a sequencer... This sequencer activates a G, Y, and R light system s.t. G is on for 1 cycle, followed by Y for 2 cycles followed by R for 1 cycle. Then the whole cycle repeats

```python
Q1 = 0
Q0 = 0

def NOT( a ):
    return 1 - a

for step in range( 20 ):
    # the Q1 and Q0 outputs go through combinational logic to generate the new values of D1, D0, and the outputs G, Y, R...
    D1 = Q0
    D0 = NOT( Q1 )
    G = NOT( Q1 ) & NOT( Q0 )
    Y = Q0
    R = NOT ( G | Y )

    # show the stable circuit signals
    print( "Q1Q0 = %d %d | GYR = %d %d %d" % ( Q1, Q0, G, Y, R ) )

    # wait for the next clock tick
    # (the user presses Enter)
    input( "> " )

    # as soon as the clock has ticked,
    # D1 and D0 get latched in the flipflops
    # and Q1 and Q0 reflect the values captured.
    Q1 = D1
    Q0 = D0
```

http://www.science.smith.edu/dftwiki/index.php/CSC270_GYRSsequencer.py
Recap
Exercise to Do in Pairs

Generate an FSM that controls 3 lights: Green, Yellow, and Red, and cycles through all 3, staying 1 second on Green, 1 second on Yellow, and 1 second on Red.

Different from previous exercise!
We can easily introduce **input signals** that will modify the behavior of the FSM.
Controllable FSM
Moore Machine

Input

Clock

Combinational

Flip-Flops

Output