we pick transitions that limit the number of bit flipping.

We don't care what Green or Red do in the transient states (S5, S6, & S7).

3 FFS. $f_{\text{req}} = \frac{1}{105} = 0.1 \text{ Hz}$

<table>
<thead>
<tr>
<th>$Q_2 \ Q_1 \ Q_0$</th>
<th>$Q_2 \ Q_1 \ Q_0 \ R$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 0</td>
<td>0 0 1</td>
</tr>
<tr>
<td>0 0 1</td>
<td>0 1 0</td>
</tr>
<tr>
<td>0 1 0</td>
<td>0 1 0</td>
</tr>
<tr>
<td>0 1 1</td>
<td>0 1 0</td>
</tr>
<tr>
<td>1 0 0</td>
<td>1 0 0</td>
</tr>
<tr>
<td>1 0 1</td>
<td>1 0 0</td>
</tr>
<tr>
<td>1 1 0</td>
<td>0 0 0</td>
</tr>
<tr>
<td>1 1 1</td>
<td>0 0 1</td>
</tr>
</tbody>
</table>

$D_2 \ D_1 \ D_0$
\[ D_2 = \overline{Q_2} \cdot \overline{Q_1} \cdot \overline{Q_0} \]

\[ D_1 = \overline{Q_0} \cdot (Q_2 \oplus Q_1) + Q_0 \cdot (Q_2 \oplus Q_1) + Q_0 Q_1 \]

\[ = Q_0 \oplus (Q_2 \oplus Q_1) + Q_0 Q_0 \]

\[ D_0 = \overline{Q_2} \cdot \overline{Q_1} + Q_2 Q_0 \]

\[ R = \overline{Q_1} \cdot \overline{Q_0} \]

\[ C = \overline{R} \]

[ ] For explanations, see listing!
# Homework 5
# Problems 1 and 2
# CSC 270
# 2012
# D.T.
# for problem #2, since we associated Red ON with the state where
# Q0, Q1, and Q2 are 0, then all we need to do is clear D2, D1, and
# D0 when the command is 1. So we AND the functions of Problem 1 that
# generate D2, D1, and D0 with NOT( cmd ). No analysis necessary!
#
def NOT( a ):
    return 1 - a

def Problem1():
    Q2 = 1
    Q1 = 0
    Q0 = 0

    for step in range( 20 ):
        # the Q1 and Q2 outputs go through combinational logic to generate the new val
        # uses
        # of D1, D2, and the output G, Y, R...
        D2 = NOT( Q2 ) & Q1 & NOT( Q0 )
        D1 = ( Q0 & Q1 ) | ( Q2 ^ Q1 ) ^ Q0
        D0 = NOT( Q2 ) & NOT( Q1 ) | ( Q2 & Q0 )
        R = NOT( Q1 ) & NOT( Q0 )
        G = NOT( R )

        # show the stable circuit signals
        print( "%d% d% d% | GR = %d% d% % ( Q2, Q1, Q0, G, R )" )

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

        # as soon as the clock has ticked, D1 and D2 get latched in the flipflops
        # and Q1 and Q2 reflect the values captured.
        Q0 = D0
        Q1 = D1
        Q2 = D2

    def Problem2():
        Q2 = 0
        Q1 = 0
        Q0 = 0

        for cmd in [0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,0,0,0,1,1]:
            # the Q1 and Q2 outputs go through combinational logic to generate the new val
            # uses
            # of D1, D2, and the outputs G, Y, R...
            D2 = NOT(cmd) & ( NOT( Q2 ) & Q1 & NOT( Q0 ) )
            D1 = NOT(cmd) & (( Q0 & Q1 ) | ( Q2 ^ Q1 ) ^ Q0 )
            D0 = NOT(cmd) & ( NOT( Q2 ) & NOT( Q1 ) | ( Q2 & Q0 ) )
            R = NOT( Q1 ) & NOT( Q0 )
            G = NOT( R )

            # show the stable circuit signals
            print( "%d% d% d% d% | GR = %d% d% % ( cmd, Q2, Q1, Q0, G, R )" )

    main()

# Problem 1
-------------------
Q2Q1Q0 = 1 0 0 | GR = 0 1
Q2Q1Q0 = 0 1 0 | GR = 1 0
Q2Q1Q0 = 1 1 0 | GR = 1 0
Q2Q1Q0 = 0 0 0 | GR = 0 1
Q2Q1Q0 = 0 0 1 | GR = 1 0
Q2Q1Q0 = 0 1 1 | GR = 1 0
Q2Q1Q0 = 1 0 1 | GR = 1 0
Q2Q1Q0 = 1 1 1 | GR = 1 0
Q2Q1Q0 = 0 0 1 | GR = 0 1
Q2Q1Q0 = 0 1 0 | GR = 1 0
Q2Q1Q0 = 1 0 0 | GR = 1 0
Q2Q1Q0 = 1 1 0 | GR = 1 0
Q2Q1Q0 = 0 0 0 | GR = 0 1
Q2Q1Q0 = 0 0 1 | GR = 1 0
Q2Q1Q0 = 0 1 1 | GR = 1 0
Q2Q1Q0 = 1 0 1 | GR = 1 0
Q2Q1Q0 = 1 1 1 | GR = 1 0
Q2Q1Q0 = 0 0 0 | GR = 0 1
Q2Q1Q0 = 0 0 1 | GR = 1 0

# Problem 2
-------------------
cmd Q2Q1Q0 = 0 0 0 0 | GR = 0 1
cmd Q2Q1Q0 = 0 0 0 1 | GR = 1 0
cmd Q2Q1Q0 = 0 0 1 1 | GR = 1 0
cmd Q2Q1Q0 = 0 0 1 0 | GR = 1 0
cmd Q2Q1Q0 = 0 1 0 0 | GR = 1 0
cmd Q2Q1Q0 = 0 1 0 1 | GR = 1 0
cmd Q2Q1Q0 = 0 1 1 0 | GR = 1 0
cmd Q2Q1Q0 = 0 1 1 1 | GR = 1 0
cmd Q2Q1Q0 = 0 0 0 1 | GR = 1 0
cmd Q2Q1Q0 = 0 0 0 0 | GR = 1 0
cmd Q2Q1Q0 = 0 0 1 0 | GR = 1 0
cmd Q2Q1Q0 = 0 0 1 1 | GR = 1 0
cmd Q2Q1Q0 = 0 1 0 1 | GR = 1 0
cmd Q2Q1Q0 = 0 1 1 0 | GR = 1 0
cmd Q2Q1Q0 = 0 1 1 1 | GR = 1 0

cmd Q2Q1Q0 = 0 0 0 0 | GR = 0 1