April 28, 2013

Designing a Divide-by-Three Logic Circuit

http://www.crbond.com

Purpose

This document provides a detailed description of each step in the design of an efficient *divide-by-three* logic circuit. The methods used are those presented in the document titled "Advanced Logic Design Techniques in Asynchronous Sequential Circuit Synthesis", C. Bond, 2002, available from the author at the above website.

Several unique signal requirements are imposed on the design, and the previously published methods are freely adapted to yield a solution which is in some sense optimal.

Before proceeding, be advised that most logic designers, when confronted with a need for a divide-by-three circuit, will opt for a design which uses flip-flops or shift registers and supporting logic. The thought of designing from gate level up is usually not the first one that comes to mind. Building circuits with higher level elements generally speeds the design process and simplifies troubleshooting. That approach is completely workable and really needs no defense.

Nevertheless, gate level designs offer opportunities for minimization, optimization and control that may recommend them over higher level design methods. Here is one solution.

Problem Statement

For this divider we are given a clock signal which is a simple, symmetrical square wave, T. This clock is the only input to the circuit. The output of the circuit consists of three, symmetric overlapping square waves whose frequency is one third that of the input. The circuit will be implemented with NAND gates, although the problem statement could be satisfied with other devices.

Constructing the Flow Table

The design process begins with the construction of a flow table expressing the problem statement in simplified form, as in Table 1.

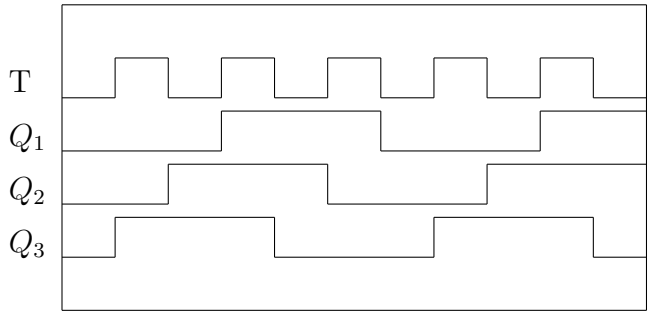


Figure 1: Divide-by-Three Timing Waveforms

T		
0	1	$Q_{1,2,3}$
(**1**)	2	0 0 0
3	(**2**)	0 0 1
(**3**)	4	0 1 1
5	(**4**)	1 1 1
(**5**)	6	1 1 0
1	(**6**)	1 0 0

Table 1: Flow Table for Divide-by-Three

This table illustrates some design techniques which exploit properties of the particular problem. For example, the required output states yield unique combinations for each row of the table. These can be directly equated to internal variables, as will be shown later. Stable states are indicated by parentheses.

There are six rows in the table, indicating the presence of three internal variables,[1] so the assignment of output values to internal states is direct. No row merging will be required using the given flow table.

Replacement of the enumerated stable state entries in the table by the corresponding secondary variables results in Table 3. Note that the structure of

[1]Recall that 2^n states require n internal variables.

	T	
$y_1\ y_2\ y_3$	0	1
0 0 0	(**1**)	2
0 0 1	3	(**2**)
0 1 1	(**3**)	4
1 1 1	5	(**4**)
1 1 0	(**5**)	6
1 0 0	1	(**6**)

Table 2: Secondary Assignments for Divide-by-Three

the original flow table simplifies the mapping.

	T	
$y_1\ y_2\ y_3$	0	1
0 0 0	(000)	001
0 0 1	011	(001)
0 1 1	(011)	111
1 1 1	110	(111)
1 1 0	(110)	100
1 0 0	000	(100)

Table 3: Flow Table with Boolean State Entries

Deriving the Circuit Equations

The above table is not only a complete problem statement, it is also a complete solution description. The circuit behavior is defined by the changes in secondary variable values following motion from cell to cell in the table.

As a first step toward deriving the circuit equations, split the table into individual elements, one for each secondary variable.

In these tables we have abandoned the special notation for stable states. They are not necessary in the derivation of equations, but may (will) become useful in optimizing the preliminary solutions or in analyzing and eliminating critical races and hazards.

In keeping with the conventions used in the previously cited document, the devices which correspond to secondary variables are identified with upper case labels, such as Y_n, and the outputs associated with those devices are identified with lower case labels, such as y_n.

	T	
$y_1\ y_2\ y_3$	0	1
0 0 0	0	0
0 0 1	0	0
0 1 1	0	1
1 1 1	1	1
1 1 0	1	1
1 0 0	0	1

Y_1

	T	
$y_1\ y_2\ y_3$	0	1
0 0 0	0	0
0 0 1	1	0
0 1 1	1	1
1 1 1	1	1
1 1 0	1	0
1 0 0	0	0

Y_2

	T	
$y_1\ y_2\ y_3$	0	1
0 0 0	0	1
0 0 1	1	1
0 1 1	1	1
1 1 1	0	1
1 1 0	0	0
1 0 0	0	0

Y_3

Table 4: Split Internal State Tables

The equations are taken directly from Table 4 by simply picking up all 1's, as is done with any combinational logic table. For our design, the equations for the secondary variables are:

$$Y_1 = y_1 \cdot y_2 + T \cdot y_1 + T \cdot y_2 \quad (1)$$
$$Y_2 = y_2 \cdot y_3 + \overline{T} \cdot y_2 + \overline{T} \cdot y_3 \quad (2)$$
$$Y_3 = \overline{y_1} \cdot y_3 + T \cdot \overline{y_1} + T \cdot y_3 \quad (3)$$

Clearly, there are many ways to regroup the terms in each equation, depending on the need for emphasizing common sub-expressions or minimization. This will be taken up in the next section. For now, note that the minterm redundancy in this representation supports smooth transitions along rows and columns.

Implementing the Circuit

A straightforward reduction of the equation minterms to NAND gates will be demonstrated. This implementation only succeeds if a few auxiliary signals which are not part of the original specification are available. Such signals may require additional gates.

Direct implementation of the minterms of the equations for variable Y_1 leads to a simple circuit, such as in Figure 2. No special implementation techniques are involved in this circuit.

Note that the output device, Y_1 (upper case), is equivalent to the the signal y_1 (lower case). The distinction is purely for convenience and there is a one-to-one correspondance between every device and its output. Furthermore, the use of y_n for signal identification is simply a convention. In fact, Y_1 (or

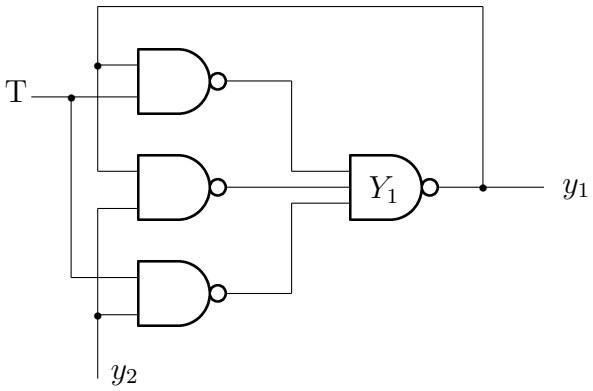


Figure 2: Direct Implementation of Y_1

y_1) is identified with the signal Q_1 in the problem statement. The variables Y_2 and Y_3 can be treated in the same way. For the (almost) complete circuit,

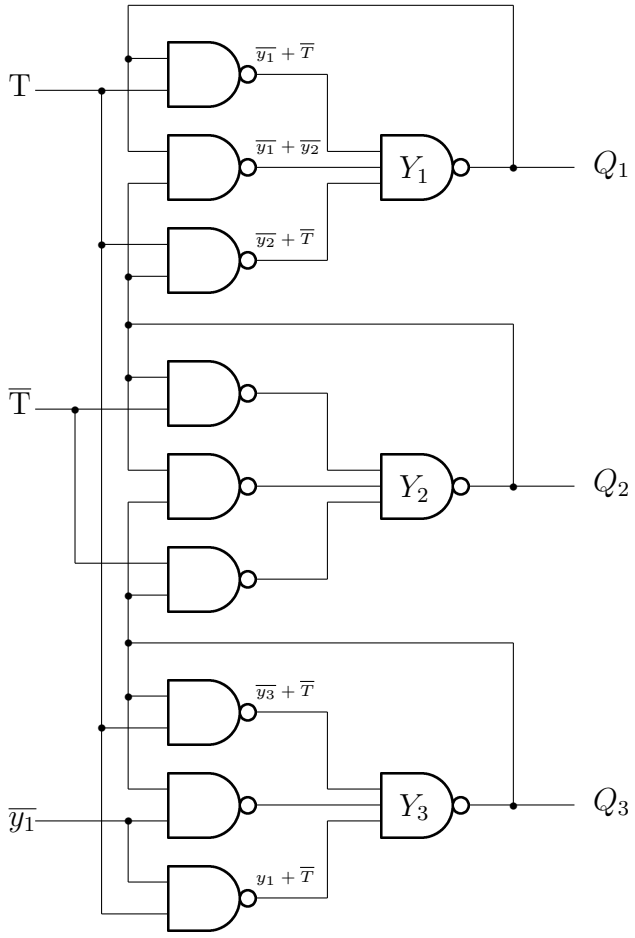


Figure 3: First (Partial) Implementation of Divide-by-Three

we have Figure 3, above.

We refer to this as a 'partial' implementation because of the presence of two signals which were not part of the original specification, $\overline{T}$ and $\overline{y_1}$. These signals could, of course, be generated immediately by simply connecting inverters to T and Y_1. A full analysis reveals that the circuit will function properly if those devices are included for this purpose.

Note that if the user wishes to implement the circuit in a PLD, simply entering the equations into the source file are all that ia necessary. Implementing the equations with other devices or gates is possible. Optimization of any initial implementation is dependent on specific and can be managed using techniques presented in the above cited paper.

Optimization

The original version of this paper had a section on optimization of the above schematic. Unfortunately, the resulting schematic contained errors so a second release deleted that section. In this third revision, the correctly optimized circuit is included. The optimization consists of eliminating the auxiliary signals, $\overline{\mathrm{T}}$ and $\overline{y_1}$, by canceling literals with available signals. The result is shown in Figure 4

Note that cancellation introduces races and hazards that can cause a malfunction. It is imperative to trace the signal paths to assure proper operation. It also generally increases the delay between a trigger event and the following stable state. Such increases limit the maximum frequency the circuit can handle. This can be determined by tracing the signal flow paths or by simulation.

Further Considerations

Although the circuit as it stands fully meets the stated requirements, there are other issues a conscientious designer should consider. For example, we have not made device fan-out an issue in the specification or the design. The viability of our result should be confirmed for the logic family in use.

Another issue is the existence of 'forbidden' states. There are eight combinations of Q_1, Q_2 and Q_3 possible, but we have only used six of them – relegating the others, by omission, to *don't cares*. That leaves two states unaccounted for. A conservative design would include the examination of the behavior of the circuit in the event an unknown state occurred, and assure that the circuit would move into a know state in all cases. This is an exercise for the reader and can be done with the methods in the cited paper.

Finally, a master reset or initialization of the circuit may be desirable. Again,

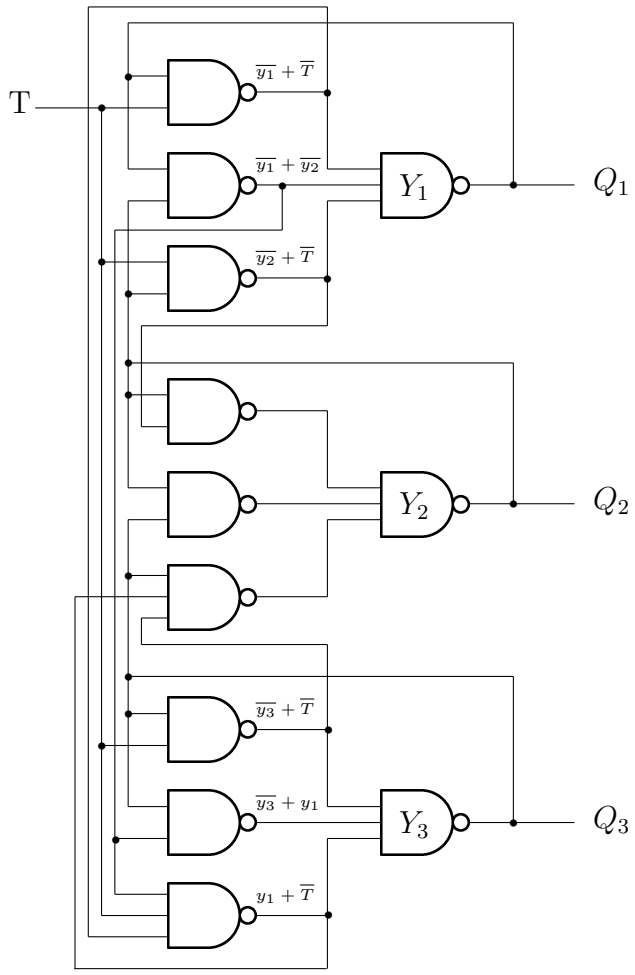
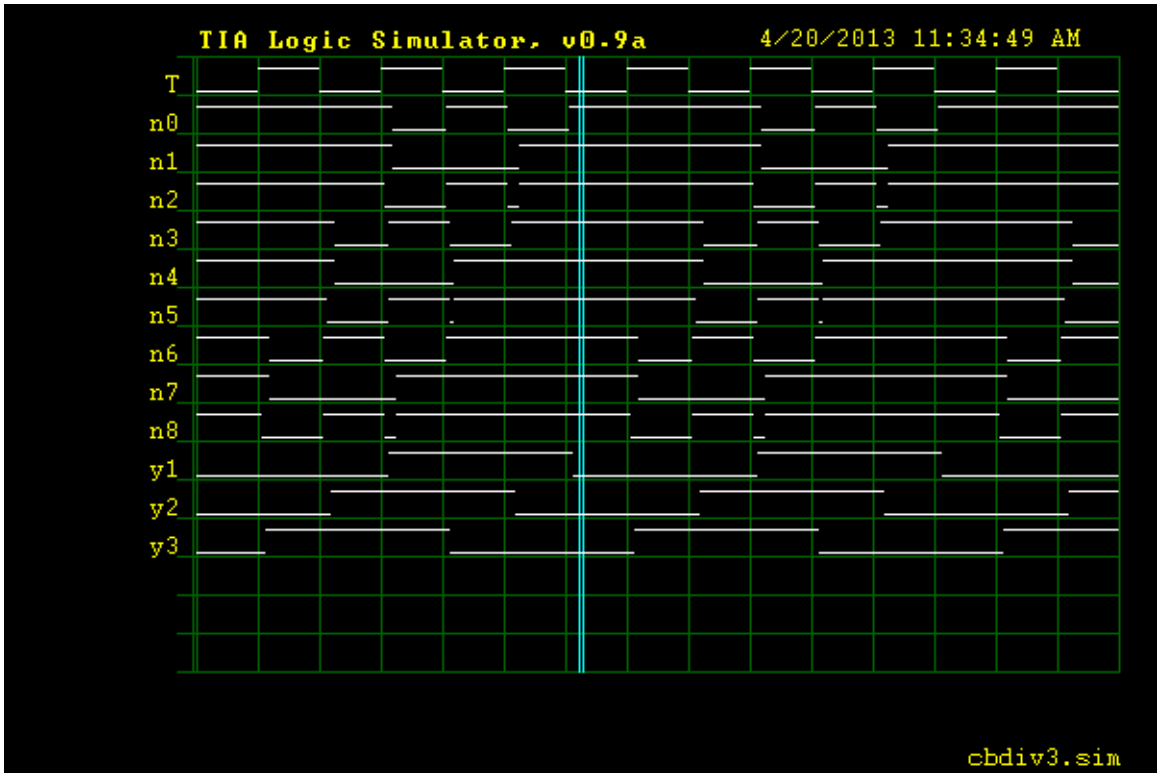


Figure 4: Optimized Implementation of Divide-by-Three

the cited paper shows methods for 'forcing' a multistate circuit into a known state and guidelines for determining the least number of connections required.

Simulation

Here you will find a screen shot of a simulation of the circuit. The input 'T' and outputs of each gate are shown.


TIA Logic Simulator, v0.9a
4/20/2013 11:34:49 AM
T
n0
n1
n2
n3
n4
n5
n6
n7
n8
y1
y2
y3
cbdiv3.sim