July 17, 2005

Designing a Data Synchronizer Logic Circuit

http://www.crbond.com

Purpose

One of the most familiar of the logic blocks used in complex logic designs is the 'D' flip-flop. This device transfers the value of a binary signal input to its output under control of a system clock. Generally, one edge of the clock is designated as the triggering event. Once the trigger has occurred and the input signal has been transferred to the output, the output is held until the next trigger event.

In this document we will design a data synchronizer, which is similar to the 'D' flip-flop, except that the trigger occurs on *both* edges of the clock.

Such a device may have applications in one of the following areas:

- Wide band data aligner,
- Noise reducer,
- Correlator,
- Low-pass filter.

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. Familiarity with the contents of that document are essential for understanding the material.

Flow Table

Since the problem statement is simple and straightforward, we can immediately construct a suitable flow table.

This flow table includes eight rows and, if no row merging is done, would require three secondary variables to implement. Merging is possible, and as shown in cited document, provides a means to reduce the number of secondary (internal) variables required. Also shown in the cited document

DC				
00	01	11	10	Q
(**1**)	2	–	4	0
1	(**2**)	3	–	0
–	2	(**3**)	8	0
1	–	7	(**4**)	0
(**5**)	2	–	8	1
1	(**6**)	7	–	1
–	6	(**7**)	8	1
5	–	7	(**8**)	1

Table 1: Primitive Flow Table for Data Synchronizer

are design examples which benefit by avoiding this reduction, or even by expanding the number of rows.[1]

Row merging is typically done in such a way that the output values can be identified with one of the secondaries. This often, but not always, simplifies the implementation. A first cut at row merging observing this rule is shown in Table 2.

	DC				
$y_1\, y_2$	00	01	11	10	Q
0 0	(**1**)	2	7	(**4**)	0
0 1	1	(**2**)	(**3**)	8	0
1 1	(**5**)	2	7	(**8**)	1
1 0	1	(**6**)	(**7**)	8	1

Table 2: Merged Flow Table with Secondary Assignments, (v.1)

This merged table is identified as a *first version*, (v.1), in anticipation of another version to be presented later. In this table, the secondary y_1 is identified with the output Q.

Replacing the numbered entries with their secondary state equivalents yields Table 3, where the stable states are identified by the enclosing parentheses.

The next step in the design would be to derive the circuit equations from the split maps taken from Table 3. However, the author has found that

[1]The overriding lesson is that simplification or reduction rules should be applied when they provide some improvement. But be cautioned that there are situations where they are not appropriate.

	DC				
$y_1\ y_2$	00	01	11	10	Q
0 0	(00)	01	10	00	0
0 1	00	(01)	(01)	11	0
1 1	(11)	01	10	(11)	1
1 0	00	(10)	(10)	11	1

Table 3: Completed Flow Table for Data Synchronizer, (v.1)

this particular mapping has suffered from the merging chosen. That is, the attempt to merge so that the output is identified with one of the secondaries has complicated the design. Because of this, an alternative approach in which the merging does not support a direct identification of the output with one of the secondaries is preferred. The cost of this choice is that the output must be decoded from the inputs and secondaries.

An alternative merged flow table is shown if Table 4. Note that the output values cannot be directly equated with either of the secondaries.

	DC				
$y_1\ y_2$	00	01	11	10	Q
0 0	(**1**)	(**2**)	3	4	0
0 1	(**5**)	2	(**3**)	8	?
1 1	5	6	(**7**)	(**8**)	1
1 0	1	(**6**)	7	(**4**)	?

Table 4: Merged Flow Table with Secondary Assignments, (v.2)

Replacing the enumerated values with the equivalent secondary combinations gives Table 5.

	DC			
$y_1\ y_2$	00	01	11	10
0 0	(00)	(00)	01	10
0 1	(01)	00	(01)	11
1 1	01	10	(11)	(11)
1 0	00	(10)	11	(10)

Table 5: Completed Flow Table for Data Synchronizer, (v.2)

Deriving the Equations

From Table 5 we construct a map for each secondary.

	DC			
$y_1\, y_2$	00	01	11	10
0 0	0	0	0	1
0 0	0	0	0	1
1 1	0	1	1	1
1 0	0	1	1	1

Y_1

	DC			
$y_1\, y_2$	00	01	11	10
0 0	0	0	1	0
0 1	1	0	1	1
1 1	1	0	1	1
1 0	0	0	1	0

Y_2

Table 6: Completed Maps for Data Synchronizer Secondaries, (v.2)

The corresponding equations are:

$$Y_1 = y_1(D + C) + D \cdot \overline{C} \quad (1)$$
$$Y_2 = y_2(D + \overline{C}) + D \cdot C \quad (2)$$

It might be instructive to compare the simplicity of these equations with that of the equations derived from the merged flow table of version 1. There is a substantial difference. Of course, we now have to decode the output with additional logic, and until that is done an overall estimate of the merits of each method isn't possible.

Decoding the Output

Finding equations for the required output signal is begun by constructing a 'skeletal' flow table.[2] Such a table provides a blank map to be filled in according the the requirements of the signal.

Here is the skeletal flow table for the problem at hand. Only the stable state entries have been represented, as the output is only defined for these states.

The decoding is done by filling in each stable state entry with its corresponding output value, taken from the original flow table. The unstable states are treated as *don't cares*.

An equation for output Q is then:

$$Q = y_1 \cdot C + y_1 \cdot y_2 + y_2 \cdot \overline{C} \quad (3)$$

[2]See the cited paper for details.

	DC			
$y_1\, y_2$	00	01	11	10
0 0	(**1**)	(**2**)		
0 1	(**5**)		(**3**)	
1 1			(**7**)	(**8**)
1 0		(**6**)		(**4**)

Table 7: Skeletal Flow Table for Data Synchronizer, (v.2)

	DC			
$y_1\, y_2$	00	01	11	10
0 0	(0)	(0)		
0 1	(1)		(0)	
1 1			(1)	(1)
1 0		(1)		(0)

Table 8: Output Decoder Flow Table for Data Synchronizer, (v.2)

This simple, and possible redundant, representation of the output signal will be implemented in the next paragraphs.

Implementation

We have chosen to implement the equations derived from the second version of the merged flow table and the output decoder in Eq: 3. The result is shown in Figure 1.

It is possible that this implementation harbors critical races which may interfere with its expected behavior. In particular, we note that the output is allowed to change following a change in the clock (C) input. But the output circuit decodes combinations of y_1 and y_2, which also may change following a clock change. Hence, we should determine whether it is necessary to allow y_1 and y_2 to settle before applying C to the output decoder. This can be determined by methods shown in the cited paper, and is left as an exercise for the reader.

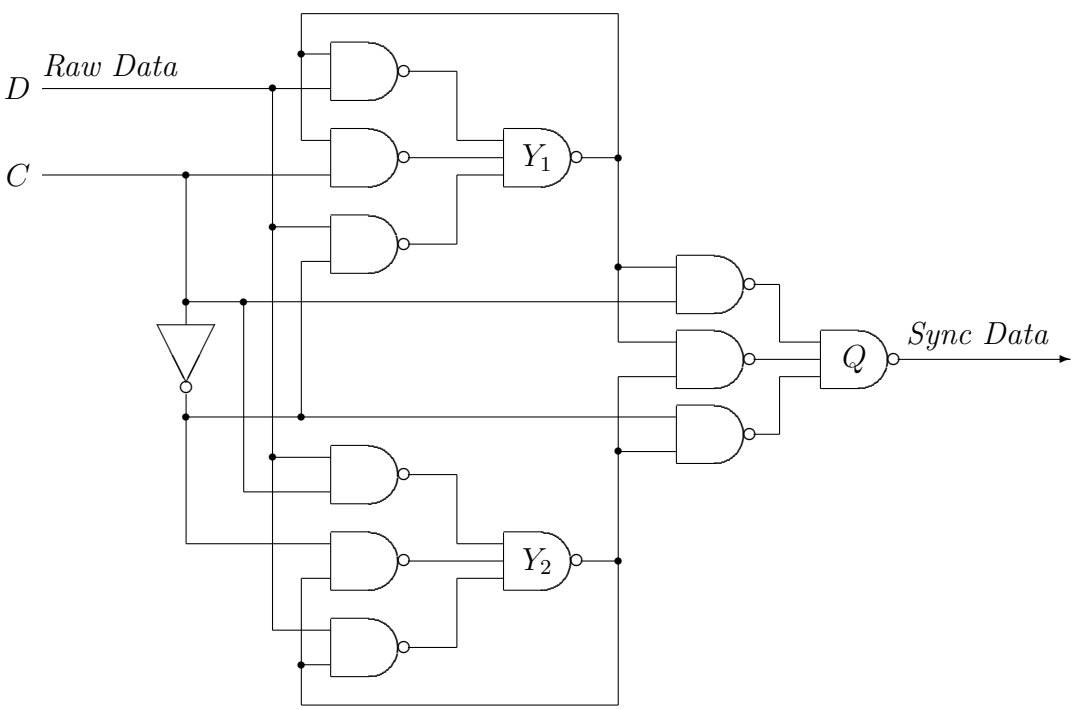

Figure 1: Data Synchronizer