Report Number: WUCSE-2005-43

2005-09-16

Synthesis of Control Elements from Petri Net Models

J. R. Cox and D. Zar

Methods are presented for synthesizing delay-insensitive circuits whose behavior is specified by Petri net models of macromodular control elements. These control elements implement five natural functions used in asynchronous system design. Particular attention is paid to modules requiring mutual exclusion where metastability must be carefully controlled.

2005-43

Synthesis of Control Elements from Petri Net Models

Authors: Cox, J. R.; Zar, D. M.

September 16, 2005

Abstract: Methods are presented for synthesizing delay-insensitive circuits whose behavior is specified by Petri net models of macromodular control elements. These control elements implement five natural functions used in asynchronous system design. Particular attention is paid to modules requiring mutual exclusion where metastability must be carefully controlled.

Synthesis of Control Elements from Petri Net Models

J. R. Cox and D. Zar

Department of Computer Science and Engineering
Washington University
St. Louis MO 63130

Abstract - Methods are presented for synthesizing delay-insensitive circuits whose behavior is specified by Petri net models of macromodular control elements. These control elements implement six natural functions used in asynchronous system design. Particular attention is paid to modules requiring mutual exclusion where metastability must be carefully controlled.

Index Terms – arbitration, asynchronous systems, circuit synthesis, combinational hazards, macromodular control elements, metastability, mutual exclusion, Petri nets, reachability graphs.

1.0 Introduction

The use of Petri net models of asynchronous systems is well known [1], [2], [3], [4]. However, application to a substantial system design encounters a state-space explosion that renders synthesis through Petri net reachabililty graphs problematic. The approach considered here creates delay-insensitive (DI) modules that can be composed to produce endlessly scalable systems. These modules include the macromodular control elements developed by Clark and Molnar [5], [6], [7]. They form a natural set whose functionality is easily understood and sufficiently complete when coupled with DI processors to have been the basis for a wide variety of substantial systems [8]. Since the development of macromodules more than 30 years ago, integrated circuit technology has shrunk the size required to fabricate such modules by 4 or 5 orders of magnitude and increased their speed 1 or 2 orders of magnitude. Perhaps the time has come to reexamine the role that this modular approach might play in system-on-chip (SoC) designs that are fabricated on multi billion transistor integrated circuits.

This paper demonstrates how a designer can proceed from a Petri net model of a control element to the layout of an integrated circuit. Of particular interest is the synthesis of those parts of the control elements that require mutual exclusion, a phenomena that manifests itself in a Petri net model as conflict [1]. Special care is also required in all asynchronous circuits to prevent transient errors in combinational circuit outputs due to unanticipated component delays. These transient errors are known as *hazards* and techniques to produce hazard-free implementations of these control elements are explored.

2.0 Macromodular Control Elements

Figure 1 is adapted from [6] and sketches a portion of a macromodular system including processing elements (rectangles), data paths between them (heavy lines) and sequencing paths (light lines) between control elements (circles). The original macromodular systems operated asynchronously and their elements and interconnecting pathways observed DI design rules externally. However internally, correct behavior of these elements was dependent upon proper arrangement of their component delays.

The data paths in the original systems were multi conductor cables that included control signals that accompanied the data paths to insure their correct receipt at the destination. These *escort* signals allowed the source to remove the data only after it had been safely received at the destination.

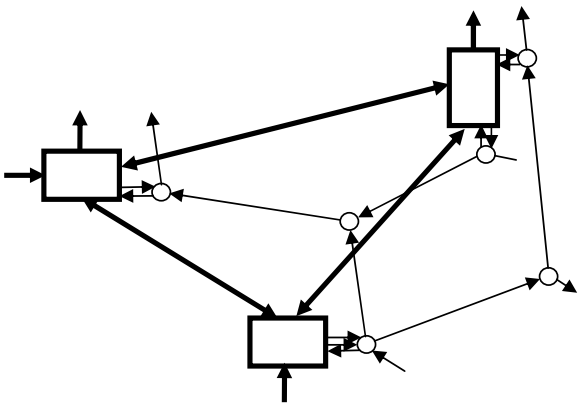


Figure 1. Processing and sequencing networks (adapted from [6])

In SoC applications these cables would likely be replaced by DI FIFOs to improve performance over long chip distances and to maintain DI behavior. Escort signals can manage the DI characteristics and acknowledgments by using control elements embedded in the data paths.

The macromodular control elements discussed here include the ***B***ranch, ***R***endezvous, ***D***ecision, ***M***erge, ***C***all and ***I***nterlock. These control elements utilize transition signaling, with each logic level transition, whether up or down, representing a control signaling event.

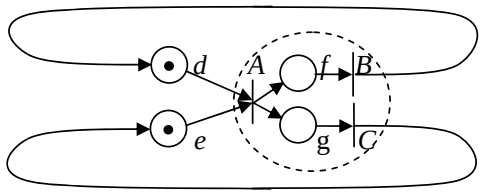
The precise definition of the behavior of these elements is most easily specified by Petri net models. There are, however, slight differences in some of the definitions presented below from the original ones in [6], [7]. The ***I*** element described below has only a single completion line from each half of the called processor whereas the original elements

featured dual completion lines. This simplification does not decrease the expressive power of the set, as will be discussed below. In addition, the ***D*** element below serves a more theoretical purpose as a modeling tool rather than as a physical device. It selects one of two outputs randomly whereas the decision element in [7] was actually built and selects one of two outputs depending on an input level.

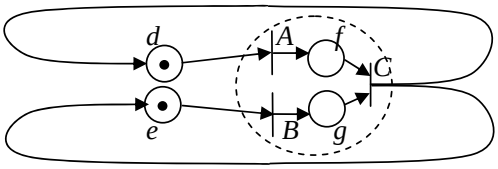
3.0 Petri Net Models

Models of the control elements must also include a model of the environment in which they function. This is necessary since certain disallowed sequences of input events could lead to unwanted behaviors. These environmental models must provide exactly the allowed input sequences, no more and no less.

3.1 Branch and Rendezvous Elements



***Branch* (*B*)**



***Rendezvous* (*R*)**

Figure 2. Petri net models of two simple control elements and their environments

Within the dashed circles of Figure 2 are shown the Petri net models of the ***B***ranch and ***R***endezvous control elements. The Petri net models of their environments are as simple as possible and are shown outside the dashed circles. The transitions at the input (output) of the elements receive (generate) signaling events from (for) the environment. The reachability analysis [1] of these nets can be obtained from the initial marking M_0 and the production rules in Pr.1. Note that the Petri net transitions[1] are identified with capital letters and the Petri net places with lower case letters. The reachability graphs for the ***B*** and ***R*** elements show that the two nets are live and safe, but they will not be displayed here. In the next section an example reachability graph will be shown for the ***R*** element.

[1] Context will usually distinguish Petri net transitions from logic level transitions, but occasionally full names will be used to assist the reader.

Branch	***Rendezvous***	
$M_0 = de$	$M_0 = de$	
$A: de \rightarrow fg$	$A: d \rightarrow f$	Pr. 1
$B: f \rightarrow d$	$B: e \rightarrow g$	
$C: g \rightarrow e$	$C: fg \rightarrow de$	

The output maps, needed for synthesis, can be calculated directly and automatically from the basic Petri net descriptions given by their initial markings and production rules. The vertex and edge sets that underlie the reachability graphs are calculated along the way to producing these output maps. Petri net properties such as liveness and safety are also determined as a byproduct of the output map calculations. The ***B*** and ***R*** elements are complementary. They often appear together, for example in the parallel processing of two sequences of jobs in processors ***P1*** and ***P2***, as shown in Fig. 3.

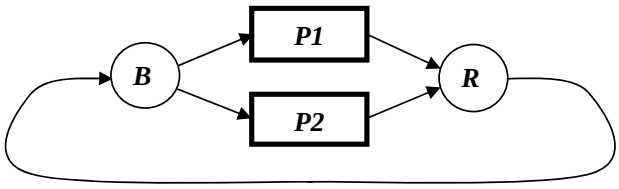
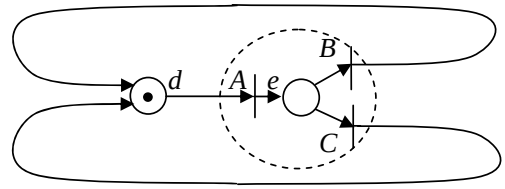


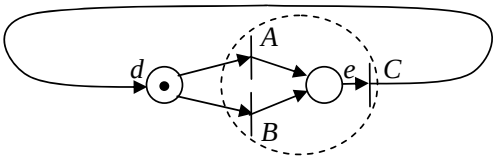
Figure 3. Example of the use of the ***B*** and ***R*** elements in parallel processing

The lines connecting the elements represent wires which are modeled by individual places connected to the input and output transitions of the control element models of Fig. 2. Tree connections of ***B*** and ***R*** elements allow the formations of branch and multiple input rendezvous elements with an arbitrary number of outputs and inputs, respectively.

3.2 Decision and Merge Elements



Decision (D)



***Merge* (*M*)**

Figure 4. Petri net models of two simple control elements, ***D*** and ***M***.

The reachability analysis of the ***D*** and ***M*** elements in their environments (Fig. 4) also shows liveness and safety, which can be determined from the initial markings and production rules given in Pr. 2.

Decision (D)	***Merge (M)***
$M_0 = d$	$M_0 = d$
$A: d \rightarrow e$	$A: d \rightarrow e$
$B: e \rightarrow d$	$B: d \rightarrow e$
$C: e \rightarrow d$	$C: e \rightarrow d$

Pr. 2

The ***D*** and ***M*** elements form a similar complementary pair although the ***D*** element is more of a theoretical construct than a physical circuit. It can be used in Petri net models of a system to replace a ***P***rocessing element that, depending on the outcome of a computation, yields two or more outputs as in Fig. 5.

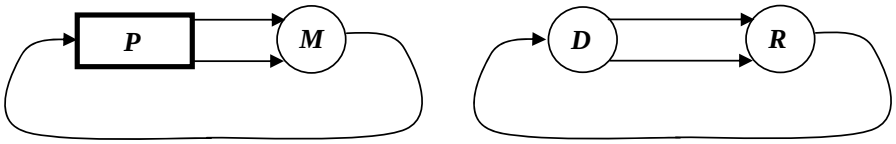


Figure 5. Replacement of a ***P*** element by a ***D*** element for high-level behavior analysis.

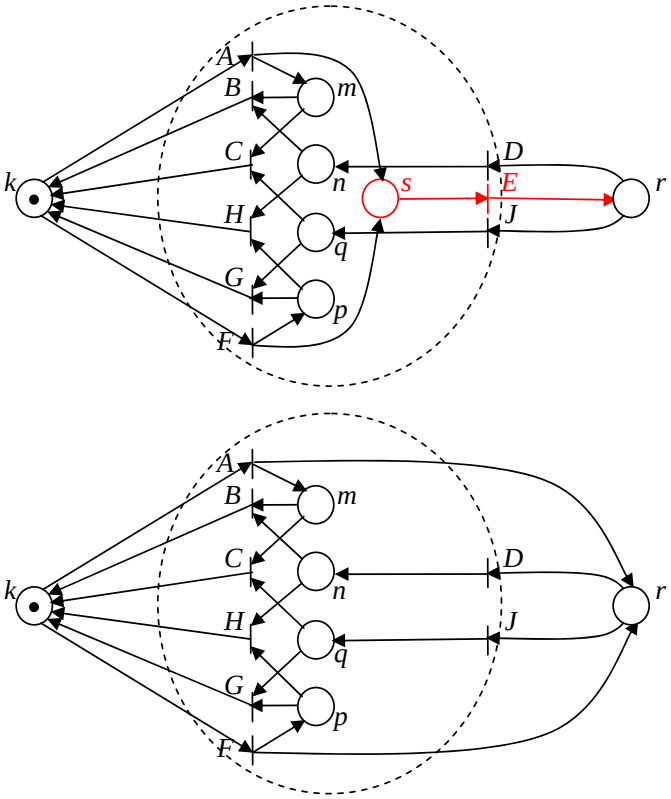
Here, for the purpose of high-level behavioral analysis, the details of the computation performed by ***P*** element can be suppressed by virtue of the fact that the ***D*** element randomly chooses one or the other output. Conflict exists in the Petri net model of the ***D*** element (Fig. 4) at transitions *B* and *C* and suggests the possibility of metastability if a physical device were to be built. As with the ***B*** and ***R*** elements, inputs and outputs can be expanded to an arbitrary number by constructing a tree of ***D*** and ***M*** elements.

3.3 Call Element

The Petri net model for the ***C***all element, shown in the upper part of Fig. 6, has 4 inputs and 5 outputs and is substantially more complex than the previous four elements. The production rules for the ***C*** element[2] follow in Pr. 3. The purpose of the ***C*** element is to provide access for two or more processors to a shared processor and, upon completion of use of that shared resource, return control to the calling processor.

The initial marking M_0 and the left-hand set of production rules for the ***C*** element and its environment, shown in Pr. 3, can demonstrate that the Petri net is both live and safe. These rules can be simplified somewhat by eliminating place *s* and transition *E* (in red) while connecting the outputs of transitions *A* and *F* directly to place *r* instead of *s*. These changes are shown in the right-hand set of productions rules in Pr. 3 and in the lower part of Fig. 6. These changes leave the behavior of the model unchanged except for the absence of the firing of output transition *E* during the execution of the Petri net.

[2] Our ***C*** element should not be confused with the Muller C-element which corresponds to our ***R*** element.



Call (C)

Figure 6. Petri net model of the Call element.

Elimination of *E* makes the arrangement of transitions symmetric about the horizontal centerline of the Petri net. The firings of *E* can be subsequently inferred by connecting an ***M*** element's inputs to transitions *A* and *F*. The elimination of this one transition leaves eight transitions in the Petri net and reduces the number of possible distinguishable input/output states from 512 to 256, a valuable simplification.

Call (C)

$M_0 = k$	$E : s \to r$	$M_0 = k$		
$A : k \to ms$	$F : k \to ps$	$A : k \to mr$	$F : k \to pr$	
$B : mn \to k$	$G : pq \to k$ =>	$B : mn \to k$	$G : pq \to k$	Pr. 3
$C : mq \to k$	$H : np \to k$	$C : mq \to k$	$H : np \to k$	
$D : r \to n$	$J : r \to q$	$D : r \to n$	$J : r \to q$	

The typical use of the ***C*** element[3] is shown in Fig. 7 and allows processors ***P1*** and ***P2*** to share processor ***P3***. ***P1*** and ***P2*** must not request ***P3*** at the same time since ***C*** can handle only one request at a time. ***P3*** will produce an output on either one of the two completion lines, but not both. The ***C*** element will transmit these alternative completion signals to the calling processor. Coordination between ***P1*** and ***P2*** will determine which processor requests ***P3*** next.

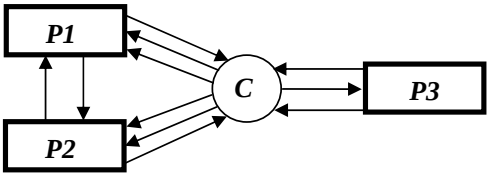


Figure 7. Processors ***P1*** and ***P2*** share the use of processor ***P3*** through a ***C*** element.

The original set of macromodular control elements included two kinds of call elements, one with a single completion line and one with two as defined here. Obviously, the dual-completion ***C*** element can play the role of a single-completion element by tying off one input and ignoring one of the outputs connected to each calling element.

3.4 Interlock Element

Deleted: 4

The Petri net model of the Interlock ***I*** in Fig. 8 is also substantially more complex than the models of the first four control elements. It has 4 inputs and 4 outputs and involves conflict in the firing of transitions *F* and *G*. M_0 and the productions rules shown in Pr. 4 can be used to show that the ***I*** element and its environment are both live and safe. Furthermore, in contrast to the ***C***, the ***I*** element provides arbitration of concurrent requests for the use of a shared resource as shown in Fig. 9. This introduces mutual exclusion and the possibility of metastability, a topic to be discussed in Section 4.3.

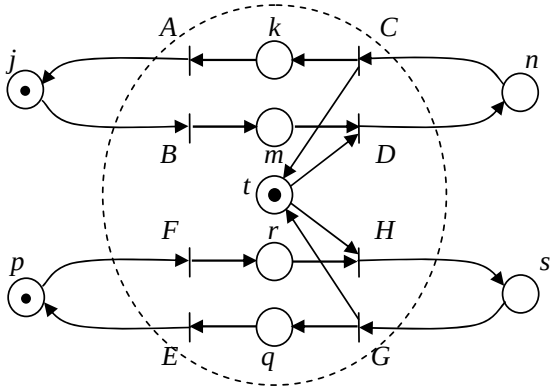


Figure 8. Petri net model of the Interlock element and its environment.

[3] Note that the ***C*** element is distinguished from the *C* transition by the former being in boldface.

Interlock (I)

$M_0 = jpt$

$A: k \rightarrow j$	$E: q \rightarrow p$	
$B: j \rightarrow m$	$F: p \rightarrow r$	Pr. 4
$C: n \rightarrow kt$	$G: s \rightarrow qt$	
$D: mt \rightarrow n$	$H: rt \rightarrow s$	

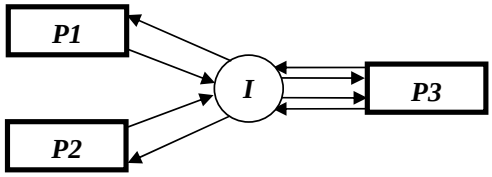


Figure 9. Interlock element arbitrating concurrent requests from ***P1*** and ***P2*** for use of ***P3***.

The ***I*** element in Fig. 9, in contrast to the original macromodular Interlock element, receives only a single completion line from each half of ***P3***. That is, the original Interlock receives dual-completion lines from both the upper and lower halves of ***P3***. In addition, the upper half of the original Interlock outputs dual-completion lines to ***P1*** and the lower half outputs dual completion lines to ***P2***. Therefore, in the original Interlock one of two completion signals will always be returned to the calling processors. The ***I*** element defined in Pr. 4 cannot, by itself, provide this functionality.

However, by combining an ***I*** with two ***C*** elements we get, as shown in the dashed box of Fig. 10, the same functionality as the original Interlock element.

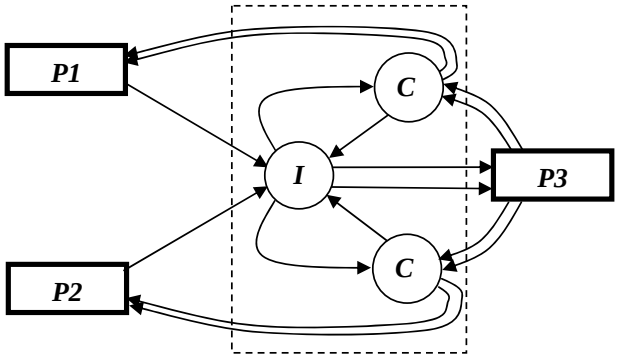


Fig. 10. Creation of a dual-completion Interlock element from an ***I*** and two ***C*** elements.

Both the ***C*** and ***I*** elements can be composed to form versions with an arbitrary number of inputs or outputs, just as was the case for the ***B***, ***R***, ***D*** and ***M*** elements. A Call element with greater than two inputs can be formed from a tree of two-input ***C*** elements. In a fairly obvious manner two-input ***I*** and ***C*** elements can be connected in tandem to produce

an arbitrating call element. Such an element is a building block that can be used in a tree structure to produce an arbitrating call with any number of inputs.

Thus, the six control elements ***B***, ***R***, ***D***, ***M***, ***C*** and ***I*** have the same functionality and same expressive power as provided by the original macromodule control elements [6], [7]. In the next section we will describe a methodology for synthesizing four of these, the ***B***, ***R***, ***M*** and ***C*** elements. The method for synthesizing the ***D*** and ***I*** elements will be deferred until the following section where conflict in the Petri net model is examined; this complication leads to a requirement for providing a mutual exclusion element in the circuit.

4.0 Synthesis Techniques

Although the ***B***, ***R***, and ***M*** control elements are quite simple and can be synthesized from Petri net models by a number of techniques [9], [10], [3], [4], we propose to use a highly automated method in order to manage the complexity of the ***C*** element. The first step for any of these elements stems from the fact that a signaling event modeled by the firing of a Petri net transition can be either a rising logic level or a falling logic level. The Petri net production rules for these six elements must be augmented to distinguish rising from falling logic level transitions. After that step we will present an example synthesis of the ***R*** element. That example should make clear the how the output table results are obtained for the remaining five elements.

4.1 Logic Level Encoding

Each of the Petri net transitions that represent input or output events can be expanded to encode the direction of the logic level transition. This step requires the expansion from one to two transitions and the addition of two places. For example, in Fig. 11 a Petri net transition *A* is expanded to include a down transition to zero A_0 and an up transition to one A_1. New places a_0 and a_1 are added to encode the resulting state of the logic variable. The places *b* and *c* are in the original Petri net and remain unchanged. In some cases there may be more than one place input to or output from the expanded transition, but in either case these places remain unchanged.

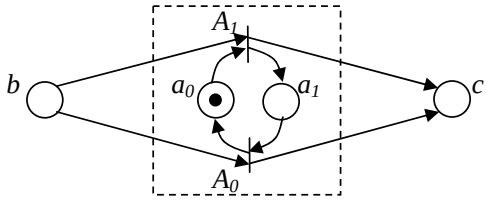


Figure 11. Expansion of a Petri net transition to encode the logic level state.

To see how this works we apply this expansion to the *A*, *B* and *C* transitions in the production rules for the ***R*** element and show the results in Pr. 5.

Rendezvous (R)

$$
\begin{array}{lcll}
M_0 = de & & M_0 = a_0 b_0 c_0 de & \\
A: d \to f & \Rightarrow & A_0 : a_1 d \to a_0 f & A_1 : a_0 d \to a_1 f \\
B: e \to g & & B_0 : b_1 e \to b_0 g & B_1 : b_0 e \to b_1 g \\
C: fg \to de & & C_0 : c_1 fg \to c_0 de & C_1 : c_0 fg \to c_1 de
\end{array}
$$

Pr. 5

With a careful choice of labels for Petri net transitions and places the expansion is straight forward. Algorithms to obtain the reachability set and the edge set for the reachability graph are also straight forward. Any one of several available programs can produce the desired results, but we chose to construct a Mathematica program in order to be able to extend its functionality to the calculation of tables for each of the control element's new outputs, given the present state of all of its inputs and outputs.

4.2 Reachability Graphs and Output Tables

The reachability set for ***R*** given by the output of our Mathematica program is

```
{{0,{a0,b0,c0,d,e}},{1,{a0,b0,c1,f,g}},
 {2,{a0,b1,c0,d,g}},{3,{a0,b1,c1,e,f}},
 {4,{a1,b0,c0,e,f}},{5,{a1,b0,c1,d,g}},
 {6,{a1,b1,c0,f,g}},{7,{a1,b1,c1,d,e}}}
```

where the octal coding of the binary logic levels abc_8 uniquely identifies each of the reachable markings. Using these identifiers the edges of the reachability graph become

```
{{0,4},{0,2},{1,0},{2,6},{3,1},
 {4,6},{5,1},{6,7},{7,3},{7,5}}
```

in which each edge is directed from the first to the second member of the tuple identifying the edge. From this Mathematica result the reachability graph can be drawn as shown in Fig. 12.

The output *c* of the ***R*** element is the least significant digit of the binary coded marking and changes whenever transition *C* fires. The red edges between markings 6 and 7 and between markings 1 and 0 represent the firing of transition *C* and, thus an output logic level change in *c*. Thus, markings 6 and 1 represent unstable states of the logic circuit to be synthesized. All the remaining markings represent stable states of the circuit.

Another Mathematica program has been written to list the new circuit state for each of its 8 present states. It uses only the set of reachable markings and the set of edges connecting them. The truth table results are shown in Table 1.

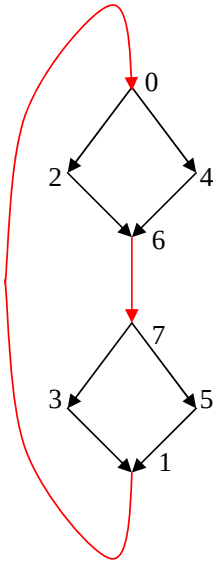


Figure 12. Reachability graph for the ***R*** element.

Rendezvous Output	***c′***
$abc_8 = abc_2$	
0 = 000	0
1 = 001	0
2 = 010	0
3 = 011	1
4 = 100	0
5 = 101	1
6 = 110	1
7 = 111	1

Table 1. Output of the Rendezvous element.

The new value c' of output c that follows each unstable state is shown in red in Table 1. The equation for the output variable c' can be synthesized from Table 1 by use of a Karnagh map or a synthesis program such as Espresso [11]. In the case of the ***R*** element, the equation for the output variable c' is given in Eq. 1.

$$c' = ab + ac + bc \qquad \text{Eq. 1}$$

The above procedure can be repeated for each of the other remaining three control elements that are conflict free. The computed results are summarized Eq. 2 through Eq. 4 below.

Branch (B)

$$M_0 = de \quad A: de \to fg \quad B: f \to d \quad C: g \to e \quad \Rightarrow \quad M_0 = a_0 b_0 c_0 de$$

$$A_0 : a_1 de \to a_0 fg \qquad A_1 : a_0 de \to a_1 fg$$

$$B_0 : b_1 f \to b_0 d \qquad B_1 : b_0 f \to b_1 d$$

$$C_0 : c_1 g \to c_0 e \qquad C_1 : c_0 g \to c_1 e$$

Pr. 6

Branch Output	***b′***	***c′***
$abc_8 = abc_2$		
0 = 000	0	0
1 = 001	0	0
2 = 010	0	0
3 = 011	0	0
4 =100	1	1
5 = 101	1	1
6 = 110	1	1
7 = 111	1	1

Table 2. Outputs of the Branch element.

$$b' = a$$

$$c' = a$$

Eq. 2

Merge (M)

$$M_0 = d \quad A: d \to e \quad B: d \to e \quad C: e \to d \quad M_0 = a_0 b_0 c_0 d$$

$$A_0 : a_1 d \to a_0 e \qquad A_1 : a_0 d \to a_1 e$$

$$B_0 : b_1 d \to b_0 e \qquad B_1 : b_0 d \to b_1 e$$

$$C_0 : c_1 e \to c_0 d \qquad C_1 : c_0 e \to c_1 d$$

Pr. 7

Merge Output	***c′***
$abc_8 = abc_2$	
0 = 000	0
1 = 001	0
2 = 010	1
3 = 011	1
4 = 100	1
5 = 101	1
6 = 110	0
7 = 111	0

Table. 3. Output of the Merge element.

$$c' = a \oplus b$$

Eq. 3

Call (C)

$M_0 = k$		$M_0 = a_0 b_0 c_0 d_0 f_0 g_0 h_0 j_0 k$		
$A: k \to mr$		$A: a_1 k \to a_0 mr$	$A: a_0 k \to a_1 mr$	
$B: mn \to k$		$B: b_1 mn \to b_0 k$	$B: b_0 mn \to b_1 k$	
$C: mq \to k$		$C: c_1 mq \to c_0 k$	$C: c_0 mq \to c_1 k$	
$D: r \to n$	=>	$D: d_1 r \to d_0 n$	$D: d_0 r \to d_1 n$	Pr. 8
$F: k \to pr$		$F: f_1 k \to f_0 pr$	$F: f_0 k \to f_1 pr$	
$G: pq \to k$		$G: g_1 pq \to g_0 k$	$G: g_0 pq \to g_1 k$	
$H: np \to k$		$H: h_1 np \to h_0 k$	$H: h_0 np \to h_1 k$	
$J: r \to q$		$J: j_1 r \to j_0 q$	$J: j_0 r \to j_1 q$	

In Table 4 the hex coding of the present state can be read from the left hand column for *abcd* and the top row for *fghj*. For example, the initial mark state M_0 is located in the upper left corner at 00_{16} and produces output $b' = 0$. The output $b' = 1$ (in red) at $26_{16} = 0010\ 0110_2$ results from a current state in which $b = 0$. The new output $b' = 1$ ultimately leads to the stable state $66_{16} = 0110\ 0110_2$.

Note that in Table 4 the hex digits a, b, c, d, e, f are never in italics, but the output variables *a, b, d, e, f, g, h, j* are always in italics. Also note that there are many *don't care* cells in Table 4 and as a result many coverings are possible leading to different minterm forms and logic equations other than the one for b' shown in Eq. 4. No don't care cells occur in the ***B***, ***R***, ***D*** and ***M*** elements so there is only one minterm form for their output variables. There are several ways to simplify the ***C*** element minterm forms and we have chosen those that include several XOR functions because of their apparent simplicity.

Call Output b' $fghj_{16}$ =>	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
$abcd_{16} = abcd_2$																
0 = 0000	0		0		0	0			0	0				0		0
1 = 0001			0	0		0		0	0		0				0	0
2 = 0010	0	0					1					1	0	0		
3 = 0011		1					0	0			0	0	1			
4 = 0100	0						1	1			1	1		0		
5 = 0101	1	1						0			0		1	1		
6 = 0110			1	1	1		1			1		1			1	1
7 = 0111		1		1	1	1			1	1			1		1	
8 = 1000	0	0						1			1		0	0		
9 = 1001	1						0	0			0	0		1		
a = 1010		0		0	0	0			0	0			0		0	
b = 1011			0	0	0		0			0		0			0	0
c = 1100			1	1		1		1	1		1				1	1
d = 1101	1		1		1	1			1	1				1		1
e = 1110		0					1	1			1	1	0			
f = 1111	1	1					0					0	1	1		

Table 4. Output b' of the ***C***all element and its covering.

$$
\begin{aligned}
b' &= d \cdot \bar{h} \cdot \overline{(f \oplus g)} + \bar{d} \cdot h \cdot (f \oplus g) + b \cdot (a \oplus c) \\
c' &= \bar{g} \cdot j \cdot \overline{(f \oplus h)} + g \cdot \bar{j} \cdot (f \oplus h) + c \cdot (a \oplus b) \\
e' &= a \oplus f \\
g' &= \bar{c} \cdot j \cdot \overline{(a \oplus b)} + c \cdot \bar{j} \cdot (a \oplus b) + g \cdot (f \oplus h) \\
h' &= \bar{b} \cdot d \cdot \overline{(a \oplus c)} + b \cdot \bar{d} \cdot (a \oplus c) + h \cdot (f \oplus g)
\end{aligned}
\qquad \text{Eq. 4}
$$

The output equations synthesized by Espresso have six terms each. Using XOR gates, the Espresso result can be reduced to 3 terms as shown in Eq. 4. The result for b' is shown by the colored outlines in Table 4. The four column pairs and the two row pairs with colored outlines correspond to the six terms in the Espresso results. The yellow, green and blue rectangles correspond to the first, second and third terms, respectively, in the expression for b' shown in Eq. 4.

The logic equations for c', g' and h' can be obtained in a similar fashion from Espresso results. Alternatively, by using the symmetry properties of the Petri net for the ***C*** element, these other results can be obtained from b'. For example, c' can be obtained from b' by swapping the input variables: $b \Leftrightarrow c$, $g \Leftrightarrow h$ and $d \Leftrightarrow j$. The logic equation for e' in Eq. 4 is just the same as that for the ***M*** in Eq. 3, but with the substitutions: $f \Rightarrow b$ and $e' \Rightarrow c'$. The bottom two equations can be obtained from the top two by swapping: $a \Leftrightarrow f$, $b \Leftrightarrow g$, $c \Leftrightarrow h$ and $d \Leftrightarrow j$.

4.3 Conflict Management

The remaining two of the set of macromodular control elements are the ***D*** and the ***I***. Both Petri net models contain conflict, in which two transitions may be simultaneously enabled resulting in a mutually exclusive firing of one of the two. If these two transitions are live, execution of the Petri net will lead to an unpredictable sequence of firings.

4.3.1 Decision Element

We examine the ***D*** element first since it is simpler and can set the pattern for the more complicated ***I*** element. The logic equation for the ***D*** will be determined by the same procedure as was used in the previous section and is presented below in Eq. 5.

	Decision (D)		
$M_0 = d$	$M_0 = a_0 b_0 c_0 d$		
$A : d \to e$	$A_0 : a_1 d \to a_0 e$	$A_1 : a_0 d \to a_1 e$	
$B : e \to d$	$B_0 : b_1 e \to b_0 d$	$B_1 : b_0 e \to b_1 d$	
$C : e \to d$	$C_0 : c_1 e \to c_0 d$	$C_1 : c_0 e \to c_1 d$	

Pr. 9

Decision Output	***b′***	***c′***
$abc_8 = abc_2$		
0 = 000	0	0
1 = 001	**1**	**0**
2 = 010	**0**	**1**
3 = 011	1	1
4 = 100	**1**	**1**
5 = 101	0	1
6 = 110	1	0
7 = 111	**0**	**0**

Table 5. Output of a Decision element

Examination of the reachability graph for the ***D*** element shows two edges directed away from each of the unstable states (1, 2, 4, 7): one causing b' to be different than b and the other causing c' to be different from c. The mutual exclusivity of changes in b' and c' expressed in the reachability graph must be realized in the circuit to be synthesized. We proceed by developing the logic equations for b' and c' independently of each other.

$$b' = a \oplus c$$
$$c' = a \oplus b \qquad \text{Eq. 5}$$

Map for *b′c′*	*bc* = 00	*bc* = 01	*bc* = 11	*bc* = 10
a = 0	00	**10**	11	**01**
a = 1	**11**	01	**00**	10

Table 6. Karnaugh map for outputs $b'c'$ of ***D*** element

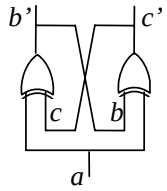


Figure 13. ***D*** element circuit.

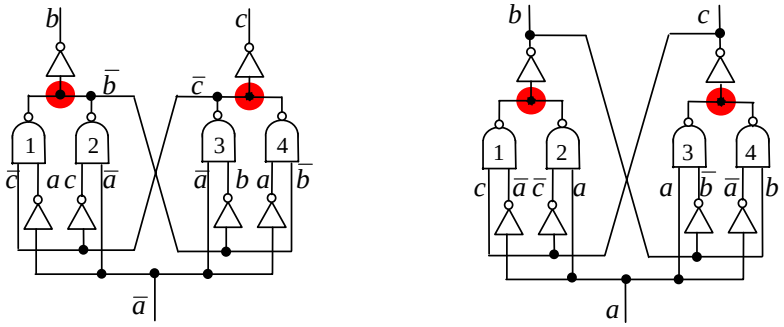
There is cross coupling between outputs b' and c' as can be seen in Eq. 5, the Karnaugh map (Table 6) and the synthesized circuit (Fig. 13). This cross coupling provides the mutual exclusivity required by the Petri net, but the subsequent transient behavior of the circuit must be examined carefully.

Unstable states are shown in bold red in Table 6. Instead of always settling to a stable state, as was the case for all the previous control elements, the ***D*** element appears to oscillate between two pairs of states *abc* =100 ⇔ 111 and *abc* = 010 ⇔ 001. The Karnaugh map does not tell the whole story, however, since it does not reflect the analog nature of the two cross-coupled XOR gates.

In Fig. 14 two realizations of the ***D*** element are shown. Figure 14(a) feeds back the outputs of the NAND gates directly to the inputs while Fig. 14(b) buffers the output of the NAND gates before feeding them back to the inputs[4]. The red dots indicate that the

[4] For brevity and because the loop has been closed the primes denoting the open loop values have been suppressed in Fig. 14 and thereafter.

output of each pair of NAND gates is joined in a wired OR gate. Both of these realizations lend themselves easily to CMOS circuits. In the realizations in [12] the delays through the input inverters are, in the non-inverted inputs, offset by CMOS transmission gate pairs that are always turned on, but provide equivalent delay.

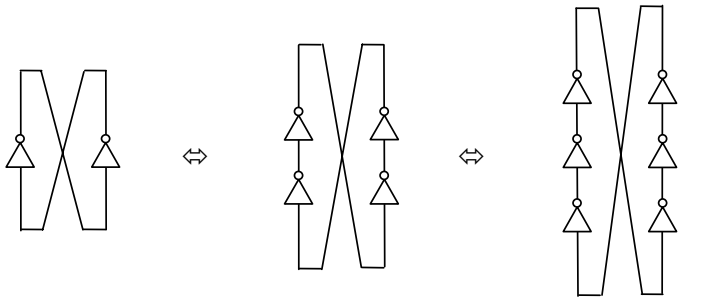


(a) (b)

Figure 14. Two realizations of the ***D*** element using inverters and NAND gates.

In all of the states shown in Table 6 either NAND gate 1 or 2 of Fig. 14 (a) is enabled by the input and never both: gate 1 for $a = 1$ and gate 2 for $a = 0$. A similar mutual exclusion exists for NAND gates 3 and 4. The situation is similar, but reversed, in Fig. 14 (b). Figure 15 shows three abstractions that cover all the stable states.

The cross-tied inverters of Fig. 15 produce a bistable circuit. As input *a* toggles between 0 and 1, the circuit of Fig. 14(a) toggles between the abstractions in Fig. 15(a) and Fig. 15(b) while Fig. 14(b) toggles between the abstractions in Fig. 15(b) and Fig. 15(c). This alternation of abstractions occurs because the switching between NAND gates 2 and 1, switches an input inverter in and out of the feedback path.



$abc \in \{101, 110\}$ $abc \in \{000, 011\}$ $abc \in \{101,110\}$

(a) (b) (c)

Figure 15. Abstraction of stable states of the circuits of Fig. 14.

Figure 15 shows the abstraction for the four stable states and for the two circuits of Fig. 14. We can reason about the transition from one stable state to another by looking at just one side of the abstractions in Fig. 15. For example, examine the left-hand inverters in

Fig. 15(a) and Fig. 15(b). Start with the state $a = 0$ and $\bar{b}\bar{c} = 00$. Then make $a = 1$. The enabled NAND gate moves from 2 to 1 in Fig. 14(a) and the inverter is no longer in the $\bar{c}$ feedback path.

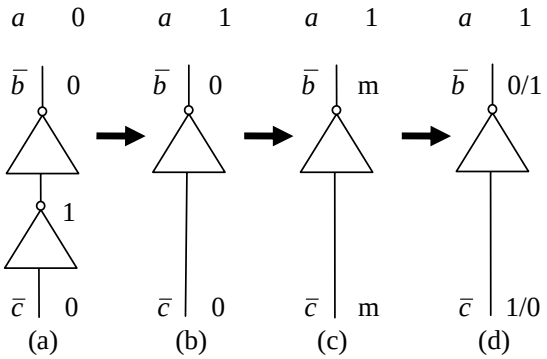


Figure 16. Progression from one stable state to another: *abc* = 011 => 101 or 110.

Figure 16(a) corresponds to the left-hand inverter pair of Fig. 15(b), but Fig. 16(b) through Fig. 16(d) corresponds to the single left-hand inverter in Fig. 15(a). In Fig. 16(b) the circuit capacitances briefly maintain the feedback voltages $\bar{b}$ and $\bar{c}$ unchanged. However, in Fig. 16(c) these voltages decay to intermediate values in the metastable region. Eventually, in Fig. 16(d) the circuit stabilizes in one of the two available stable states.

For $a = 1$, the inverter shown in Fig 16(b) through Fig. 16(d) is part of the two inverter loop shown in Fig. 15(a). This is the familiar bistable circuit or flipflop whose output waveforms proceed exponentially from the metastable state to a stable state and are comparable to those first observed in [13] for the flipflop. Swamy [12] shows waveforms exhibiting this behavior that were the result of a SPICE simulation of a CMOS circuit. The Appendix develops a simplified linear analysis of the inverter rings shown in Fig. 14 yielding results for the general case and verifying the reasoning presented in Fig. 16.

Figure 16 starts from *abc* = 011 and shows the progression to the two possible stable states *abc* = 101 or *abc* = 110. There is also a similar sequence starting from *abc* = 000 and terminating in the same two stable states. It can be inferred from Fig. 16 by complimenting the values of *a*, $\bar{b}$ and $\bar{c}$ in Fig. 16.

Starting from the stable state *abc* = 101 we see a different behavior in Fig. 17. When the enabled NAND gate moves from gate 1 to 2, the number of inverters in the loop grows from two as in Fig. 15(a), to four as in Fig. 15(b).

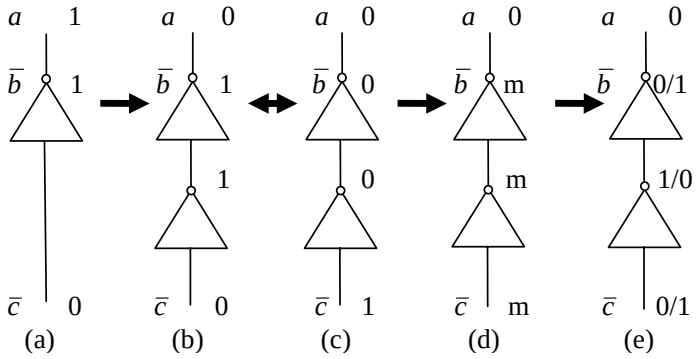


Figure 17. Progression from one stable state to another: *abc* = 101 => 000 or 011.

In this case an oscillation appears between Fig. 17(b) and Fig. 17(c). Neither of these states is stable as seen by the red entries in Table 6. The duration of oscillation will depend upon circuit parameters. In Fig. 17(d) the difference between outputs $\bar{b}$ and $\bar{c}$ will grow from a trivial amount present in the previous two oscillatory states to an appreciable amount. However, this difference is still in the metastable region. Finally in Fig. 17(e) the outputs leave the metastable region and settle into one of the two allowed stable states. This interpretation of the sequence of events is supported by a SPICE simulation [12] and the development in the Appendix. Starting from *abc* = 110 produces a sequence complementary to Fig. 17.

In the Appendix we show that the difference between the output logic voltage levels *b* and *c* is an exponentially growing function of time with a time constant proportional to C_s/g_t where C_s is the stray capacitance at the output of each inverter and g_t is the magnitude of the transconductance of each inverter in the middle of its switching region. Furthermore, the behavior of the circuit in Fig. 14(b) can be analyzed in a similar fashion with the inverter ring with Fig. 15(c) taking the place of that of Fig. 15(a). In contrast to the analysis of the circuit in Fig 14(a), oscillation as shown in Fig 17 takes place for all four unstable states.

The construction of a ***D*** element has been proposed [14] for the purpose of generating random binary digits. Chaney has dubbed this device the Perhapsatron. However, it appears quite difficult to prove lack of bias for the stream of digits generated by a real device of this sort. Even with the addition of substantial circuitry beyond the cross-coupled XORs the Perhapsatron is likely to still have some opportunity for bias. It is probably best to view the ***D*** element as a theoretical device useful in modeling and as a stepping stone to the ***I*** element.

4.3.2 Interlock Element

Like the ***D*** element, the Petri net model of the ***I*** element (Fig. 8) displays conflict. For the ***I*** element it is between the firing of transitions *D* and *H* which share inputs from place *t*.

Interlock (I)

$M_0 = jpt$	$M_0 = a_0 b_0 c_0 d_0 e_0 f_0 g_0 h_0 jpt$		
$A: k \to j$	$A_0 : a_1 k \to a_0 j$	$A_1 : a_0 k \to a_1 j$	
$B: j \to m$	$B_0 : b_1 j \to b_0 m$	$B_1 : b_0 j \to b_1 m$	
$C: n \to kt$	$C_0 : c_1 n \to c_0 kt$	$C_1 : c_0 n \to c_1 kt$	
$D: mt \to n$	$D_0 : d_1 mt \to d_0 n$	$D_1 : d_0 mt \to d_1 n$	Pr. 10
$E: q \to p$	$E_0 : e_1 q \to e_0 p$	$E_1 : e_0 q \to e_1 p$	
$F: p \to r$	$F_0 : f_1 p \to f_0 r$	$F_1 : f_0 p \to f_1 r$	
$G: s \to qt$	$G_0 : g_1 s \to g_0 qt$	$G_1 : g_0 s \to g_1 qt$	
$H: rt \to s$	$H_0 : h_1 rt \to h_0 s$	$H_1 : h_0 rt \to h_1 s$	

There are four input logic levels (*b*, *c*, *f*, *g*) and four output logic levels (*a*, *d*, *e*, *h*). The output level *d′* can be determined from the output map in Table 7 or calculated directly from the set of vertices and edges of the reachability graph. Then by using Espresso the logic equation for *d′* in Eq. 6 can be determined. The output level *h′* can be determined in a similar fashion.

There is symmetry between the outputs maps for *d′* and *h′*: one is the transpose of the other. This occurs because the logic variables are arranged so that the first four (*abcd*) pertain to the upper Interlock channel and the second four (*efgh*) pertain to the lower channel. In fact, one output can be obtained from the other by the following variable swaps: *a* ⇔ *e*, *b* ⇔ *f*, *c* ⇔ *g*, *d* ⇔ *h*.

Call Output d′ $efgh_{16}$ =>	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
$abcd_{16} = abcd_2$																
0 = 0000	0				0	0		0	0		0	0				0
1 = 0001																
2 = 0010																
3 = 0011																
4 = 0100	1				**1**	0		1	1		0	**1**				1
5 = 0101	1				1			1	1			1				1
6 = 0110																
7 = 0111	1				1	1		1	1		1	1				1
8 = 1000	0				0	0		0	0		0	0				0
9 = 1001																
a = 1010	0				0			0	0			0				0
b = 1011	0				**0**	1		0	0		1	**0**				0
c = 1100																
d = 1101																
e = 1110																
f = 1111	1				1	1		1	1		1	1				1

Table 7. Output *d′* of Interlock element and its covering.

$$
\begin{aligned}
a' &= c \\
d' &= b \cdot \overline{(g \oplus h)} + c \cdot (g \oplus h) \\
e' &= g \\
h' &= f \cdot \overline{(c \oplus d)} + g \cdot (c \oplus d)
\end{aligned}
\qquad \text{Eq. 6}
$$

The logic levels a' and e' are obvious from an examination of Fig. 8, the Petri net of the ***I*** element. The logic level d' can be understood from a realization that if *g* and *h* are at the same logic level, there must not be a token in place *s* of the Petri net. Thus, d' can follow *b* and the request of the upper channel can be granted. However, if *g* and *h* are different logic levels, *c* and *d* must be the same, preventing place *n* from being marked and deferring the grant until logic levels *g* and *h* are the same. Similar reasoning can be applied to logic level h'.

The covering for d' shown in color in Table 7 corresponds to the first (blue) and the second (green) terms in Eq. 6. Remember that the covering for h' is the transpose of that for d'.

Conflict can occur only when the shared resource is free and places *n* and *s* are empty, a circumstance that happens when logic levels $c = d$ and $g = h$. In addition, requests must be pending in both the upper and lower channels requiring that places *j* and *p* be free of a token and thus, $a \neq b$ and $e \neq f$. Finally, it must be true that the places *k* and *q* be free of a token so that $a = c$ and $f = h$. These three conditions must hold when there is conflict and are true only at the four states that are marked in Table 7 with a bold red digit, they are states 44, 4b, b4 and bb. At each one of the four states the circuit defined by Eq. 6 must resolve the conflict. So we are interested the states immediately following 44, 4b, b4 and bb. The only logic variables that can change following these states are *d* and *h*. According to the reachability graph, the changes in *d* or *h* must be mutually exclusive just as was the case between *b* and *c* for the ***D*** element. Table 8 lists the possible next states determined by the covering of Eq. 6.

Conflict at	dh = 00	dh = 01	dh = 11	dh = 10
44	**11**	01	**00**	10
4b	00	**10**	11	**01**
b4	00	**10**	11	**01**
bb	**11**	01	**00**	10

Table 8. Next states for the ***I*** element following the four conflict states.

As can be seen from Table 8 the next states following 44 and bb are identical to those for the ***D*** element shown in Table 6 for $a = 1$. Similarly, the next states following 4b and b4 are identical to those for the ***D*** element for $a = 0$. Thus, we can expect the same kind of behavior following a conflict state as described for the ***D*** element in Section 4.3.1. Since the ***I*** element can be realized in a sum of products form, it can be constructed in an analogous fashion to either Fig. 14(a) or 14(b). In this case, the d' and h' outputs will

each be composed of four, three-term products. When taken together under conflict conditions these two logic arrays become inverter rings with n = 2, 4 or 6 as shown in Fig. 15. The behavior of these rings will be as predicted in the equations given in the Appendix.

4.4 Hazards-Free Control Elements

Transient errors can occur in the outputs of a combinational circuit due to unintended component delays. These errors are called combinational hazards and must not occur in the control elements since they can produce spurious transitions in element outputs. The complexity of the ***C*** and ***I*** elements rule out manual methods of analysis and argue for a systematic approach. A ternary logic test is described by Unger [15] that can be automated based on the results of the reachability graphs obtained from the control element Petri nets.

Unger postulates a third logic value midway between the traditional binary logic values of 0 and 1. The added value is chosen to be ½ and is inserted in the sequence of input logic values between 0 and 1 for a positive going change and between 1 and 0 for a negative going change. Ternary truth tables for each of the basic logic functions (AND, OR, XOR) can be established as shown in Fig. 18.

a \ *b*	0	½	1
0	0	½	1
½	½	½	1
1	1	1	1

OR

a \ *b*	0	½	1
0	0	0	0
½	0	½	½
1	0	½	1

AND

a \ *b*	0	½	1
0	0	½	1
½	½	½	½
1	1	½	0

XOR

Figure 18. Ternary truth tables for three basic logic functions

The NOT function swaps 1 ⇔ 0 and leaves the ½ unchanged as can be seen for the special case NOT(a) = XOR(a, 1). The XNOR is just the XOR with the swap: 1 ⇔ 0.

The logic level ½ should not be viewed as a precise value, but rather as an indicator that the logic level is between 0 and 1 and its exact value is uncertain. It could equally well be represented as a “?” except that such a symbol is less convenient for automation.

No matter the configuration of the gates in a combinational circuit the presence or absence of a hazard is conserved so long as the resulting map covering remains unchanged. A hazard may or may not result in transient errors in a physical circuit depending on the values of its inherent, unintended circuit delays. A hazard, the potential for a transient error, exists depending only on the nature of the circuit’s map covering. In different physical circuit realizations different arrangements of inherent delays may be required for a hazard to be become a transient error. For example, a three-input AND and a cascade of two, two-input ANDs having the same truth table may be quite different in the values of delay through the various paths in the circuit. If a hazard exists in a circuit

containing the three-input AND, it also exists in the circuit containing the equivalent two-input ANDs, but the delay values that would cause a transient error may differ substantially.

The reachability graph of the Petri net of a control element displays all the allowed sequences of states for the combinational circuit that implements that control element. In fact, we only need to examine the paths between stable states since no hazard can occur while the circuit is in a stable state. In many cases a single edge in the reachability graph carries the circuit from one stable state to the next. For unstable states it is necessary to examine a path that consists of, at most, two edges. This path starts with a stable state, passes through an unstable state and terminates at a second stable state. We approach the application of this method in two steps: first analyze the control elements without conflict and then those with conflict.

4.4.1 Hazard Test for Conflict-Free Control Elements

The ***B***, ***M***, and ***C*** elements can have only a single input change at a time. The ***B*** element is trivial since it consists of merely two buffers between its single input and its two outputs. So long as the input transitions are properly spaced, no hazard can exist. The ***M*** element has no feedback path and since only one input can change at a time, it is easy to see no hazard can exist for this element, too.

The ***R*** element can have concurrent transitions on its two inputs, but it is much simpler than the ***C*** element and so we apply the ternary hazard test to the ***R*** as an example of the method. The method also applies to the ***B*** and ***M*** elements, but the results are unsurprising and are not discussed here. The ***D*** and ***I*** elements are discussed in the next section on conflict.

Examining the reachability graph of the ***R*** element shown in Fig.12, we see that states 0, 2, 3, 4, 5, and 7 are stable and only states 1 and 6 are unstable. However, since concurrent input transitions can occur, it is possible to pass directly from state 0 to 6 and then to 7. Similarly, it is possible to pass form state 7 to1 and then to 0.

Path	M0(abc)	M1i(abc)	M1f(abc)	M2i(abc)	M2f(abc)
{0,4}	000	½00	½00	100	100
{0,2}	000	0½0	0½0	010	010
{1,0}	001	001	000	000	000
{2,6,7}	010	½10	½1½	11½	111
{3,1,0}	011	0½1	0½½	00½	000
{4,6,7}	100	1½0	1½½	11½	111
{5,1,0}	101	1½1	1½½	00½	000
{6,7}	110	110	111	111	111
{7,3}	111	½11	½11	011	011
{7,5}	111	1½1	1½1	101	101
{0,6,7}	000	½½0	½½½	11½	111
{7,1,0}	111	½½1	½½½	00½	000

Table 9. Ternary test for hazards for the ***R*** element.

Table 9 shows the sequence of states that occur for each of the possible paths in the reachability graph between two stable states. The red states (1, 6) are unstable and the paths that begin there are shown for completeness since they appear in the set of reachability graph edges. However, they also appear in the four paths that begin at 2, 3, 4 and 5 where the behavior of the circuit can be examined more comprehensively. Likewise, they appear in the last two paths that describe concurrent inputs. Henceforth, we ignore the two rows that originate in unstable (red) states.

In column M0 we see the binary marking corresponding to the initial state of the logic levels *a*, *b* and *c*. In column M1i the inputs *a* and *b* have initiated their change from the standard values and one or the other takes on the value ½ to signify this event. In the last two rows these inputs both change to ½. Note that the output *c* in column M1i is given by Eq. 1 with inputs from the M0 column. In column M1f the output *c* is either unchanged or the logic value ½ has propagated to the output. Column M1f is the final value reached by the output that was initiated in column M1i. This final value is computed in the ternary test by a fixed-point function.

In column M2i the next change is initiated by setting the input logic levels to their final values. The output logic level is the same as that achieved in column M1f. Finally, the fixed-point process is repeated one more time in M2f and the final stable state is reached for those cases with their path through an unstable state.

For comparison with [15], identify column M1i with Step 1 in Procedure 4.3A and M1f with Steps 2, 3, and 4. Column M2i should be identified with Step 1 in Procedure 4.3B and M2f with Steps 2 and 3.

Are there any hazards represented in Table 9? The answer is no, if the progression of the output logic level is monotonically increasing or decreasing to 1 or 0, respectively, producing a final and stable state. There must not be a turning point nor an unanticipated final state. The existence of a turning point can be detected by computing the slope of *c* from M0 through M2f. There is no change in *c* between M0 and M1i nor is there a change in *c* between M1f and M2i. The remaining differences are reported in Table 10.

Path	M1i(*c*)-M1f(*c*)	M2i(*c*)-M2f(*c*)
{0,4}	0	0
{0,2}	0	0
{2,6,7}	½	½
{3,1,0}	-½	-½
{4,6,7}	½	½
{5,1,0}	-½	-½
{7,3}	0	0
{7,5}	0	0
{0,6,7}	½	½
{7,1,0}	-½	-½

Table 10. Slope of logic level output *c* of the ***R*** element.

It can be seen by inspection that the slope of *c* is flat, upward or downward with no instance of a turning point. Thus, the ***R*** element is hazard-free.

The process outlined in Tables 9 and 10 has been automated with a Mathematica program and applied to the other conflict-free elements. Note that the product of the two columns of Table 10 will always be non-negative for hazard-free control elements. Results show that in addition to the ***R***, the ***B*** and ***M*** elements are also hazard-free.

However, computer analysis of the ***C*** element does exhibit hazards. Since the reachability graph has 160 edges, it cannot be handled with certainty by manual methods. The two counter-examples shown in Table 11 are sufficient to demonstrate that hazards exist.

Path	M0(a..j)	M1i(a..j)	M1f(a..j)	M2i(a..j)	M2f(a..j)
$\{5,4,0\}_{16}$	00000101	00000010½	00½00½0½	00½00½00	00000000
$\{5,15,17\}_{16}$	00000101	000½0101	0½½½0½½1	0½½10½½1	0½½10½½1

Table 11. Ternary test counter-examples showing hazards in two ***C*** element paths.

In the path {5,4,0} output logic level *c* has the pattern 00½½0 showing a reversal in slope indicating a hazard. In path {5,15,17} each of the output logic levels *b*, *c* and *h* have the pattern 00½½½ indicating the possibility of metastability. The output logic level *g* has a similar pattern that starts from 1 instead of 0. In each case the inherent delays must have specific values for these hazards to appear.

A different covering of the "don't cares" in Table 4 that leads to a different truth table might remove the hazards, but finding that covering is a daunting task. A more tractable approach is to insert delays in the feedback paths as suggested by Molnar [10] and sketched in Fig. 19.

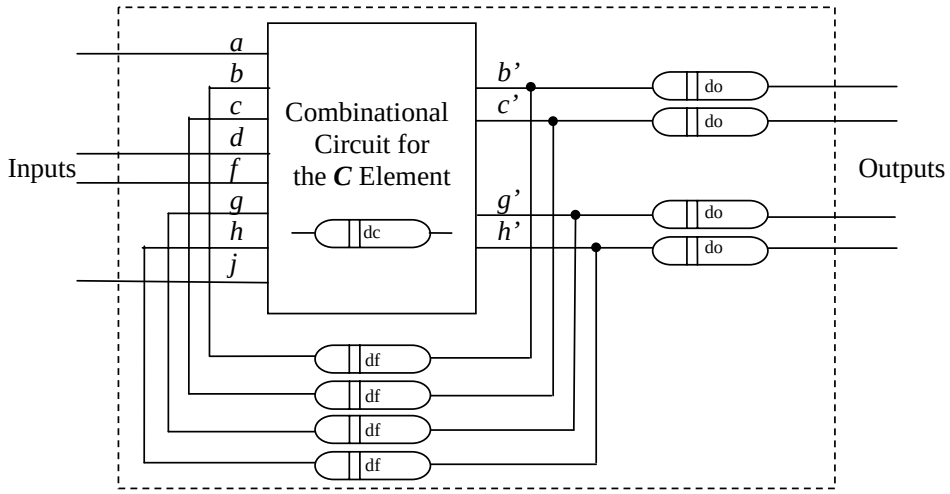


Figure 19. Delays for hazard elimination in the ***C*** element.

Let *dc* be the maximum possible delay in the propagation of any change through the combinational circuit and *df* be the minimum possible delay inserted in the feedback path. Then, the inequality $df > dc$ ensures that no output change can arrive at the input before the outputs are stable. Subsequently, when any changed output has been passed through the delay *df* to the input, the circuit will lock into the next stable state. This behavior is demonstrated in Table 12 where the two counter examples of Table 11 are repeated with the delay *df* inserted in the circuit as shown in Fig. 19.

Path	M0(a..j)	M1(a..j)	M2(a..j)	M3(a..j)	M4(a..j)
$\{5,4,0\}_{16}$:I	00000101	0000010½	00000100	00000½00	00000000
$\{5,4,0\}_{16}$:O	00__10	00__10	00__½0	00__00	00__00
$\{5,15,17\}_{16}$:I	00000101	000½0101	00010101	000101½1	00010111
$\{5,15,17\}_{16}$:O	00__10	00__10	00__1½	00__11	00__11

Table 12. Ternary test examples for ***C*** element with delays.

Because the input logic levels are different from the output levels as a result of the insertion of the delays, two rows (labeled I and O) are included for each path. In the M0 column the fed back inputs are the same as the outputs so the circuit is stable. In column M1 the input change begins and appears in the output in column M2 along with the completion of the input change. There is no need for the M1f column in this case because there is no feedback as yet. In column M3 the feedback of the output change begins and is completed in column M4. Again there is no need for the M2f column because of the absence of immediate feedback. Column M4 shows that the fed back inputs are the same as the outputs so the circuit is again stable, but now in the new state.

As can be seen by inspection of the output values, all the output slopes are zero or monotonic. Thus, no hazards exist in these two cases. Computer analysis of all 160 cases verifies that the ***C*** element circuit, with delays inserted as in Fig. 19, is hazard-free.

4.4.2 Hazard Test for Control Elements with Conflict

Because of the circuit behavior discussed in Section 4.3, we anticipate hazards in the control elements whose Petri nets contain conflict. Our approach is to identify the reachability graph paths that demonstrate conflict and test for hazards in the remaining paths between stable states. If none exist, attention can be focused on the conflict states.

We first examine the ***D*** element because of its simplicity and use what we learn there for our study of the ***I*** element. The stable states of the ***D*** element are 0, 3, 5 and 6 as can be

Path	M0(abc)	M1i(abc)	M1m(abc)	M1m'(abc)	M2m(abc)	M2m'(abc)
{0,4}	000	½00	½½½	½½½	1½½	1½½
{3,7}	011	½11	½½½	½½½	1½½	1½½
{5,1}	101	½01	½½½	½½½	0½½	0½½
{6,2}	110	½10	½½½	½½½	0½½	0½½

Table 13. Ternary test for hazards in the ***D*** element.

seen from Table 5 and shown in the Path column of Table 13. The remaining four states (1, 2, 4, 7) are unstable.

Table 13 for the ***D*** element differs markedly from Table 9 for the ***R*** element because columns M1m and M1m′ show a persistent metastable state. Even when input logic level *a* completes the transition to a standard level in column M2m, the metastable condition of the outputs persists. Only the paths originating in stable states are reported in Table 13. Those starting in unstable states oscillate as described in Section 4.3.1 and then, because of the physics of the circuit, settle into one of the four stable states. This final phase of the circuit's behavior cannot be predicted by the ternary analysis.

As suggested in columns M1m′ and M2m′, repeatedly applying the logic function for the ***D*** element given in Eq. 5 to the entries in column M1m and M2m leaves the result unchanged with both output logic levels at ½. This violates the requirement for termination of the ternary test in an expected stable state. Thus, the circuit for the ***D*** element contains a metastability hazard.

Each one of the four unstable states is a result of conflict in the Petri net model of the ***D*** element and all the paths between stable states pass through one of these four states. The ternary analysis predicts these hazards, but we already knew there was trouble in these unstable states.

There are 112 edges in the reachabilty graph for the ***I*** element. Of these the ternary test warns of metastability hazard in 32 cases and a combinational hazard in 8 cases. The remaining 72 edges prove to be hazard-free.

For both the ***D*** and ***I*** elements, elimination of the metastability hazards associated with conflict is necessary. The mutual exclusion circuit originated by Seitz [9], later modified for CMOS implementation [4], [16], provides just the right detector for metastability. When the output of the metastability detector is deasserted, the outputs of the ***D*** or ***I*** circuits can be clocked into output flip-flops thereby hiding the hazards associated with conflict and its resultant possibility of metastability. This solution to the problem of metastability hazards has been described in detail and successfully simulated for the ***I*** element [12].

The combinational hazards that appear in the 8 cases can be removed by augmenting the logic equations as follows:

$$\begin{aligned} a' &= c \\ d' &= b \cdot \overline{(g \oplus h)} + c \cdot d \cdot (g \oplus h) + b \cdot c \cdot d \\ e' &= g \\ h' &= f \cdot \overline{(c \oplus d)} + g \cdot h \cdot (c \oplus d) + f \cdot g \cdot h \end{aligned} \qquad \text{Eq.7}$$

The added terms have been determined through manual trial and error and unfortunately have not, at this time, had the benefit of algorithmic calculation. The ternary test of these equations is, of course, automated, but the added terms have been chosen by hand.

4.4.3 Elimination of System Hazards

Combinational and metastability hazards internal to the control element are not our only concern. A system of control elements can contain feedback loops causing a change in an input before the effects of the previous input change have settled to their final value. The output delay *do* shown in Fig. 19 can be used to prevent this hazard.

Consider a control element with a self-loop, a direct connection from the output of a control module to its input. A self-loop is unlikely to be used in practice, but it is the worst case for this kind of system hazard. Then the condition [10] for hazard-free composition of a system of control elements is:

$$do \geq dc + df \qquad \text{Eq. 8}$$

where *do* is the minimum value for the output delay and *dc* and *df* are the maximum values of the circuit and feedback delays, respectively. Note that control elements like the ***R*** , ***B, M, D*** and ***I,*** that are hazard-free without a feedback delay, allow *do* to satisfy the inequality in Eq. 8 with $df = 0$. However, the ***C*** element expressed by the logic of Eq. 4 will presently require a non-zero value of *df*. Perhaps a future augmentation of the logic of Eq. 4 will remove this constraint.

5.0 Conclusions

The design of a set of delay-insensitive control elements has been presented. This design proceeds from the Petri net model of the element's desired behavior. Automation of the process is used to manage the complexity, both of the logic equations that define the circuit realization and of the ternary test for hazards. A method for the management of hazards in the more complex ***C*** element is described. Control elements whose Petri net models contain conflict are shown to have the potential for metastability hazards. This particular problem must be managed with a metastability detector circuit since the duration of metastability is unknown and unbounded.

The use of these delay-insensitive control elements in association with a restartable crystal clock [17] can produce blended systems of clocked processor cores coupled by a set of standard clockless elements. These systems can be composed to produce endlessly scalable systems that are free of all hazards including synchronizer failures. These results suggest it is time to reexamine the role that this modular approach might play in system-on-chip (SoC) designs fabricated on multi billion transistor integrated circuits.

6.0 Future Work

A test system composed of two restartable clocks, two processing elements, two ***I*** elements and two connecting delay-insensitive FIFOs has been planned and will be fabricated soon. This system will use both ***R*** and ***M*** elements in the construction of the DI FIFOs. It should demonstrate the effectiveness of the blended clocked and clockless methodology. Later systems must be developed to test, in practice, the unlimited scalability of this approach.

Presently the ***C*** element requires feedback delays to be hazard-free. There are a number of algorithms for the synthesis of hazard-free circuits [4] that can be applied to the output map for the ***C*** element to eliminate the need for these delays and increase the ***C*** element's speed of operation.

The output delays shown in Fig. 19 are required to forestall the creation of hazards through the interconnection of control elements within a system. These delays are sized for the worst case of a self-loop connection. However, it would be possible to develop a CAD tool that would minimize these delays based on the worst-case delays through elements and the particular interconnection of elements in the system's design. This approach would optimize the performance of the sequencing network and might be useful where very high speed is required.

The composition of control elements in the sequencing network of a large system could be, in principle, analyzed by examining the reachability graph of the entire system modeled as a Petri net. However, the resulting state-space explosion makes this approach computationally infeasible. It is conjectured that an arbitrarily large system composed of control elements, processing elements and FIFOs can be decomposed recursively into components capable of a Petri net reachability analysis without excessive computational complexity. Proof of this conjecture must await future work. A positive result would establish a theoretical basis for the unlimited scalability of the blended approach that combines clocked and clockless components.

Acknowledgment

The authors wish to recognize the debt they owe to Fred Rosenberger, Wes Clark, Severo Ornstein, Mish Stucki, Tom Chaney and Charlie Molnar. It was Rosenberger who pointed out to one of us the occurrence of hazards in the synthesis of the macromodular control elements. He also demonstrated that it is possible to visualize metastability in SPICE. Clark, Ornstein and Stucki showed us the expressive power of the set of macromodular control elements. Chaney and Molnar led us to a deeper understanding of metastability.

Appendix

Figure 15 of Section 4.3.1 shows three abstractions of two ***D*** element circuits depicted in Fig. 14. Even these simplified abstractions are difficult to analyze in their non-linear

region except by simulation. However, considerable insight can be gained by a linear analysis. Even though this analysis applies only in the vicinity of the metastable point, it describes well the behavior as seen in several SPICE simulations of the complete circuit.

We change voltage variables to have their origins at the metastable point, halfway between the high and the low logic levels. Call these time-varying voltage levels $v_i(t)$ where *i* runs from 0 to *n* -1 and *n* is the total number of inverter stages. Thus, $v_i = 0$ for all *i* at the metastable point, $v_i = \frac{1}{2}$ is the high logic level, $v_i = -\frac{1}{2}$ is the low logic level and $|v_i| < \frac{1}{2}$ defines the metastable region.

An approximation to the circuit dynamics can be made by including stray capacitance C_s in the equivalent circuit as shown in Fig. 20.

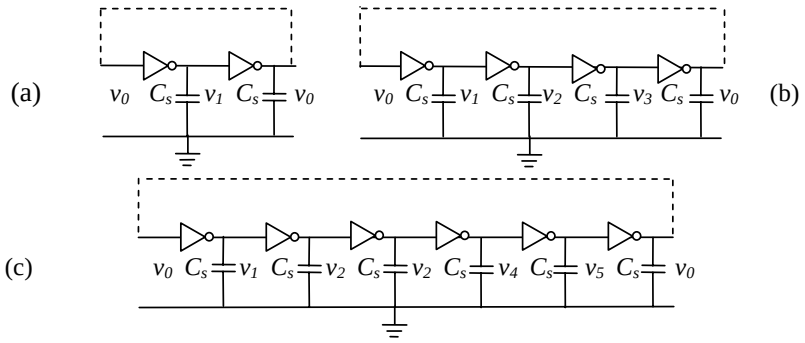


Figure 20. Inverter rings used in calculation of the ***D*** element transient response.

We assume that each inverter behaves as a linear transconductances $-g_s$ so that at each stage we have the differential equation $-g_t v_{i-1} = C_s \frac{dv_i}{dt}$ for $i = 1, 2, \cdots n-1$. Changing to a new independent variable $\tau = \left(g_t / C_s \right) t$ and taking the Laplace transform yields solutions of the form $v_j = \sum_{i=1}^{n-1} A_{ij} e^{s_i \tau}$ where the s_i are the *n* solutions of the equation $(s_i)^n = 1$. Given the initial value $v_i(0)$ of all the v_i, the constants A_{ij} can be determined and the voltage waveforms can be obtained. Of particular interest are the sum and difference of the voltages at the outputs *b* and *c*. These voltages can be calculated for *n* = 2, as in Fig. 20 (a), and are

$$\begin{aligned} v_1 + v_0 &= -e^{-\tau} \\ v_1 - v_0 &= \delta e^{\tau} \end{aligned} \qquad \text{Eq. 9}$$

where the initial values of the voltages are $\{v_0, v_1\} = -\frac{1}{2}\{(1+\delta), (1-\delta)\}$. Here δ is a constant that represents the small, but non-zero, difference between the initial values of

the two output voltages. If it were exactly zero, the time to resolve to a standard logic level would be unbounded, but the probability of this occurring in a real circuit is zero.

These calculations can be repeated for *n* = 4, as in Fig. 20 (b), and the results for small δ are

$$\begin{aligned} v_2 + v_0 &= \tfrac{1}{2}\delta e^{\tau} \\ v_2 - v_0 &= \sin\tau + \cos\tau \end{aligned} \qquad \text{Eq. 10}$$

where $\{v_0, v_1, v_2, v_3\} = \tfrac{1}{2}\{(1+\delta),(1-\delta),-1,-1\}$.

Similarly for n = 6, as in Fig. 20(c), the results for small δ are

$$\begin{aligned} v_3 + v_0 &= \tfrac{1}{3}(4e^{\tau/2}\sin\beta\tau - e^{-\tau}) \\ v_3 - v_0 &= \tfrac{\delta}{3}(e^{\tau} - e^{-\tau/2}\cos\beta\tau) \end{aligned} \qquad \text{Eq.11}$$

where $\{v_0, v_1, v_2, v_3, v_4, v_5\} = \tfrac{1}{2}\{1,-(1+\delta),-(1-\delta),1,-1,-1\}$ and $\beta = \tfrac{\sqrt{3}}{2}$.

For *n* = 2 or 6 the difference voltage grows exponentially from a small value δ until the standard output voltage levels are reached. A similar behavior occurs for the sum voltage in the case *n* = 4. Reaching standard voltage levels causes the linear model to fail and the oscillations observed for *n* = 4 or 6 will be damped out. These results have been observed in a complete SPICE simulation of the ***I*** module [12]. Also note that there is a fixed relationship between the period of the oscillations and the time constant of the exponential growth of the difference voltage for *n* = 2 or 6 and the sum voltage for *n* = 4. The oscillation period cannot be changed without changing the time constant.

References

[1] J. L. Peterson, *Petri Net Theory and the Modeling of Systems*, Prentice-Hall, 1981.

[2] T. Murata, "Petri nets: Properties, analysis, applications," Proc. IEEE, 77(4), pp. 541-580, 1989.

[3] T. H. Ming, *Synchronization Design for Digital Systems*, Kluwer Academic Publishers, 1991.

[4] C. J. Myers, *Asynchronous Circuit Design*, Wiley, 2001.

[5] W. A. Clark "Macromodular Computer Systems," *Spring Joint Computer Conf.*, AFIPS Proceedings, 30, Thompson Books, Washington D.C., pp. 335-336, 1967.

[6] S. M. Ornstein, M. J. Stucki, W. A. Clark, "A Functional Description of Macromodules," *Spring Joint Computer Conf.*, AFIPS Proceedings, 30, Thompson Books, Washington D.C., pp. 337-355, 1967.

[7] M. J. Stucki, S. M. Ornstein, W. A. Clark, "Logical Design of Macromodules," *Spring Joint Computer Conf.*, AFIPS Proceedings, 30, Thompson Books, Washington D.C., pp 357-364, 1967.

[8] W. A. Clark, C. E. Molnar, “Macromodular Computer Systems,” *Computers in Biomedical Research,* 4, R. Stacy and B. Waxman, Eds. Academic Press, New York, pp 45-85, 1974.

[9] C. L. Seitz, “System timing,” In: C. A. Mead, L. A. Conway, Eds, *Introduction to VLSI Systems*, Chapter 7,. Addison-Wesley, 1980.

[10] C. E. Molnar, T. P. Fang, F. U. Rosenberger, “Synthesis of delay-insensitive modules,” In: H. Fuchs, ed. *Proc. 1985 Chapel Hill Conference on Very Large Scale Integration,* Computer Science Press, pp 67-86, 1985.

[11] R. Rudell, A. S. Vincentelli, “Multiple valued minimization for PLA optimization,” IEEE Trans. on CAD, CAD, 6(5), pp. 727-750, 1987.

[12] U. G. Swamy, J. R. Cox, G. L. Engel, D. M. Zar, “Design of an interlock module for use in a globally asynchronous, locally synchronous design methodology,” Washington University Tech Report WUCSE-2005-52, 2005.

[13] T. J. Chaney, C. E. Molnar, “Anomalous behavior of synchronizer and arbiter circuits,” IEEE Trans. on Computers, C-22, pp.421-422, 1973.

[14] T. J. Chaney, Personal communication, 2004.

[15] Stephen Unger, *Asynchronous Sequential Switching Circuits*, Wiley-Interscience, pp. 177-183, 1969.

[16] N. Weste, D. Harris, *CMOS VLSI Design: A Circuit and Systems Persepctive*, Pearson Education, Inc., pp. 453-463, 2004.

[17] J. R. Cox, “Can a crystal clock be started and stopped?” *Appl. Math. Letters*, 1(1), pp. 37-40, 1988.