A new Approach to support Fault Simulation of Delay Insensitive Asynchronous Circuits with Synchronous Toolset

M. Mirzaaghatabar*, A. Rasooli†, B. Ghavami‡, Sh. Hessabi*
*Sharif University of Technology, Tehran, Iran {aghatabar, hessabi}@ce.sharif.edu
†Amirkabir University of Technology, Tehran, Iran {rasooli,b.ghavami}@aut.ac.ir

Abstract
Lack of global clock for synchronization in asynchronous circuits decreases the controllability of these circuits and thus makes asynchronous circuits hard to test. Delay Insensitive (DI) circuits exclusively use C-elements and inverters, provided that only single output gates are used. In this paper we present a new method to conceptually change this class of asynchronous circuits. The main idea is to change C-elements into other elements which can be modeled by synchronous tools. Then we use HOPE, a synchronous sequential circuits fault simulator, and apply it to DI class of asynchronous circuits. The stuck-at model is used for fault simulation. Our observations show that we can achieve considerable fault coverage, mainly 92.5%, in DI circuits by this method. To the best of our knowledge, this is the first effort in using synchronous tool to achieve fault simulation for asynchronous circuit class.

Keywords: Fault Simulation, Delay Insensitive circuits, C-elements.

1. Introduction
Synchronous circuits have been in use for a long time. These circuits have many problems, including power consumption, vulnerability to changes in the environment, clock skew, and so on. The introduction of asynchronous circuits aims at solving some of these problems.

Although the concept of asynchronous circuits dates back to 1950’s [1], this design method has not been able to acquire popularity due to hazard problems [1][2]. In recent years, a number of methods based on different timing models have been proposed to develop practical asynchronous circuits [3]. Asynchronous designs methods are mainly categorized into two groups, namely, bounded delay, and delay insensitive [4]. Considering the bounded delay models, the circuit delays are computed precisely, and accounted for in the design process. Regarding the delay insensitive models, synchronization between different sections is performed by generating and detecting request and acknowledgement signals. Consequently, every transition in the circuit must be acknowledged by the receiver of the transition. This property is called acknowledgment property [5].

Advantages of asynchronous circuits are as follows: eliminating the clock skew problem, modularity, lower power consumption, applying average delay instead of worst case delay, quick adaptation to newer technologies, and less vulnerability to changes in voltage and other environmental parameters such as temperature. These circuits also promise to tackle electro-magnetic interference problem [6] and are also timing fault tolerant [7].

In contrast to the mentioned advantages, there are some drawbacks regarding asynchronous circuits such as complex design procedure, larger number of transistors, and lack of asynchronous tools. As a result, having an automatic asynchronous design tool is extremely helpful in popularizing asynchronous design methods. For example, Persia design tool is developed in this regard [8]. Persia is an asynchronous design synthesis tool based on the QDI1 [9][4] timing model, and can support GALS2 [10] design as well.

Lack of asynchronous tools such as synthesis tools, ATPG (Automatic Test Pattern Generation), fault simulator, etc, results from hazard problem that make them unpopular. On the other hand, synchronous circuits have considerable power consumption which can be solved using asynchronous circuits, since their idle modules are off. For example, AMULET3 [11] is an asynchronous microprocessor that is used in industrial systems such as contact less smart cards [12] due to asynchronous circuits advantages. CAM [13] is another asynchronous microprocessor example.

---

1 Quasi Delay Insensitive
2 Globally Asynchronous Locally Synchronous
Therefore, if we modify the conceptual behavior of the asynchronous circuits such that they can be used by the available synchronous tools, we can take a step in the creation of the asynchronous tools. In this paper, we present a simple method to conceptually change the C-element in delay-insensitive circuits and design an asynchronous FIFO (First In First Out) element, and calculate the fault coverage using the HOPE software [14][15][16][17].

The rest of this paper is organized as follows. In Section 2, Delay Insensitive circuits are introduced. Section 3 describes the HOPE Fault Simulator program. Section 4 contains the discussion of our asynchronous FIFO element along with its test procedure using HOPE, and in Section 5 the simulation results is presented. Finally, Section 6 concludes the paper.

2. Delay Insensitive Circuits

Asynchronous design methods are mainly categorized into two groups, namely, Bounded delay, and Delay insensitive [4]. In bounded delay approach the timing of gates and wires are considered. They are used in the design of classical asynchronous state machines. The circuit delays can be computed precisely and used in the design process [18][19]. So, considering the bounded delay models, circuit delays are computed precisely, and accounted for in the design process.

Delay-Insensitive asynchronous circuits have no assumption on timing delays of gates and wires, hence they work correctly independent of the delays of both the gates and wires [4]. This class of asynchronous circuits was first introduced in 1960 by Monlar & Clark [4]. In delay insensitive models, synchronization between different sections is performed by generating and detecting request and acknowledgement signals. In this case, an initiator module generates a request signal for destination module and the destination module will acknowledge it [3], which is called acknowledgment property [5].

Each module in asynchronous circuits has input and output ports and different modules communicate with each other by asynchronous channels which are connected to their ports. Data communication is done by writing data to the ports and reading it from the corresponding port on the other side of the channel. For write we use the following Verilog write macro:

```
`WRITE (Port name, value)
```

If the sender decides to write another data on that port, it is suspended until the occurrence of the last data read from the port:

```
`READ (Port name, value)
```

The receiver module also remains suspended until a data is written on its counterpart port. Read and Write operations are implemented by handshake signaling. There are two so-called handshaking protocols: four-phase and two-phase handshake [4].

3. HOPE: Fault Simulator for Synchronous Sequential Circuits

HOPE [14][15][16][17] is a fault simulator for synchronous sequential circuits. It employs the parallel fault simulation technique and employs several heuristics to reduce the parallel fault simulation time. The stuck-at model is used for fault simulation in this software. HOPE was developed in the Bradley Department of Electrical and Computer Engineering, Virginia Polytechnic Institute & State University (VPI&SU). This software is Linux based, and easy to run. It has a standard input netlist for a circuit, which is described by a simple example. Consider the sequential circuit shown in Figure 1.

![Figure 1. A simple 2:1 Multiplexer](image)

This circuit is a simple 2:1 multiplexer, connected to a DFF (D Flip Flop). We wish to simulate its faults with HOPE software. First we should describe our circuit elements and then we should define some ports for inputs and outputs of the circuit and finally write the netlist for this circuit and connect the input and output ports to gate inputs and outputs and connect internodes to each other. The netlist of multiplexer is shown in Figure 2.

```
# Mux_2:1  # 2 inputs  # 1 output # 1 Nor #1 Or # 2 Ands # 4 gates # 1 DFF
INPUT(Input1)
INPUT(Select)
OUTPUT(Output)
Output = DFF(D)
V1 = AND( Input1 , Select)
V2 = NOR( Select, Select)
V3 = AND( Output , V2)
D = Or( V2, V3)
```

![Figure 2. Circuit Net List](image)
asynchronous circuit to this structure. HOPE simulates the circuits by given test vectors or random test vectors (determined by simulation mode parameter) and returns some parameters such as: Simulation Mode, Number of Test Patterns applied, Number of Collapsed Fault, Number of Detected Fault, Number of Undetected Fault, Fault Coverage, Memory Used (Kbytes) and CPU Time (Sec).


In this section, we present our method to change the DI asynchronous circuit conceptually, to make it applicable to HOPE fault simulator software. As mentioned before, a delay insensitive circuit has to be constructed exclusively from C-elements and inverters if single output gates are used. For example, Figure 3 is a delay insensitive circuit.

**Figure 3. A Delay Insensitive Circuit [20]**

C-element is a state holding operator which introduced by David-Muller [20] and has two inputs. When both inputs are ‘1’, then its output is ‘1’ and when both inputs are ‘0’, its output is ‘0’. In other cases, the element will hold its last output. Hence each data value is shown with these two bits. This representation of value is named Dual-Rail code [4]. This element is not supported by HOPE software, but other gates and elements that are used in delay insensitive circuits are supported, therefore, if we replace this element by a similar C-element, we can use it in HOPE.

**Figure 4. An Implementation of C-element**

There are two different structures which are conceptually like C-element, one of them is made of simple gates like AND and OR, and the other structure uses SR latch. These two structures are shown in Figures 5 and 6.

**Figure 5. First Structure of C-element**

As seen in Figure 5, we just use AND & OR gates that are available in HOPE. As we can see in Figure 5, there is a feedback in this combinational circuit which is not supported by HOPE. In the second structure, we use SR latch and HOPE supports DFF. Hence we use the second structure in our approach. We should just change the SR latch to DFF, which is simple. It should be mentioned that HOPE generates clock pulse for DFFs in circuit itself. It means that when you describe a DFF you should just determine its input and output netlist and the clock pulse is added automatically, however DI circuits are clock-less, and the frequency of this clock pulse is not important for us.

**Figure 6. Second Structure of C-element**

The main advantage of our idea is: each EDA (Electronic Design Automation) tools that supports SR latch or DFF can support this idea and lack of asynchronous tools are solved to some extent. The main drawback of this method is area overhead. Figure 7 shows an element of an asynchronous FIFO.
As mentioned before, we used dual rail code for data value representation. It means that there are three values in FIFO queue: true (10), false (01) and empty (00). Note that a dual rail code for a variable $x$ can be represented by two signal wires, named $x.t$ and $x.f$. Inputs to the FIFO queue must alternate between the empty value and a data value. To insert a true value into FIFO, $\text{pred}.t$ should be raised and if the $\text{pred}.f$ is raised, the false-value will be inserted to the FIFO queue. The signals $\text{succ}.t$ and $\text{succ}.f$ are the outputs of a succeeding FIFO element.

We can place any number of this FIFO element near each other, connect them together and make our own FIFO. In this paper, we simulate the FIFO elements faults.

First we should replace each C-element in Figure 7 with the structure shown in Figure 5. The new FIFO element is shown in Figure 8. Then we should write the netlist of the FIFO as described in Section 3, finally we can run the HOPE fault simulator on this netlist.

In the next section we will present the results of our fault simulation of the FIFO element. The main constraint in our approach is area overhead that is negligible for larger circuit. As our sample in this paper is a simple FIFO element the area overhead is considerable (about 50%) but in fact for other larger circuit this constraint is negligible as the larger circuits have other logics such as adder, logical gates like and, or, Xor and etc those are applicable by HOPE simulator.

### 5. Simulation Result

In this section, we will present the fault simulator result of the FIFO element using the HOPE. Our fault simulation results are shown in Table 1. The first column of the table contains the parameters returned by HOPE fault simulator software. This element was simulated four times in random mode with different number of test vectors. The second row shows the number of applied test vectors that varies from 5 test vectors to 10000000 test vectors. The fifth row shows the achieved fault coverage in each simulation.

It can be seen in fifth row that fault coverage of 10000000 test vectors are the same as fault coverage of 100 test vectors. This means that some faults are not detectable by random patterns and an ATPG must be used to detect them, provided that those faults are not redundant. Since there is no commercial asynchronous ATPG tool for DI circuits yet, we do not consider this case in this paper.

Our results show that we can achieve a considerable result, mainly 92.5% fault coverage by changing the concept of asynchronous circuits in delay insensitive class to be applicable to synchronous tools which is very important because of the lack of asynchronous tools. We also applied this technique to D-element, as shown in figure 9, and the simulation results are presented in Table 1. Figure 11 shows the fault coverage comparison between FIFO element and D-element.

<table>
<thead>
<tr>
<th>Simulation Mode (FIFO Element)</th>
<th>Random</th>
<th>Random</th>
<th>Random</th>
<th>Random</th>
</tr>
</thead>
<tbody>
<tr>
<td>Num. test patterns applied</td>
<td>5</td>
<td>10</td>
<td>100</td>
<td>10000000</td>
</tr>
<tr>
<td>Num. Collapsed faults</td>
<td>54</td>
<td>54</td>
<td>54</td>
<td>54</td>
</tr>
<tr>
<td>Num. Detected faults</td>
<td>23</td>
<td>40</td>
<td>50</td>
<td>50</td>
</tr>
<tr>
<td>Num. Undetected faults</td>
<td>31</td>
<td>14</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>Fault Coverage</td>
<td>42.539%</td>
<td>74.074%</td>
<td>92.593%</td>
<td>92.593%</td>
</tr>
<tr>
<td>Memory Used (KB)</td>
<td>139546</td>
<td>139153</td>
<td>147623</td>
<td>135335</td>
</tr>
<tr>
<td>CPU TIME (sec)</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>13.476</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Simulation Mode (D-Element)</th>
<th>Random</th>
<th>Random</th>
<th>Random</th>
<th>Random</th>
</tr>
</thead>
<tbody>
<tr>
<td>Num. test patterns applied</td>
<td>5</td>
<td>10</td>
<td>100</td>
<td>10000000</td>
</tr>
<tr>
<td>Num. Collapsed faults</td>
<td>25</td>
<td>25</td>
<td>25</td>
<td>25</td>
</tr>
<tr>
<td>Num. Detected faults</td>
<td>11</td>
<td>21</td>
<td>25</td>
<td>25</td>
</tr>
<tr>
<td>Num. Undetected faults</td>
<td>14</td>
<td>4</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Fault Coverage</td>
<td>44.00%</td>
<td>84.00%</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td>Memory Used (KB)</td>
<td>92313</td>
<td>92912</td>
<td>93589</td>
<td>94682</td>
</tr>
<tr>
<td>CPU TIME (sec)</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>9.476</td>
</tr>
</tbody>
</table>
The layout of new C-element is shown in figure 10. These layouts (Figure 4, 10) have designed with Magic toolset under TSMC18 roles. We also used HSpice to extract the power consumption overhead of our method. The power overhead was 58pW about 22% in a C-element and about 6% in a FIFO element and 14% in a D-element which are really negligible.

Figure 12. FIFO Element in new structure

Figure 10. Layout of C-element with new structure

6. Summary and Conclusions

In this paper, we discussed a new method for fault simulation of Delay Insensitive asynchronous circuits with a synchronous tool; HOPE. We defined the delay insensitive circuits and their handshaking protocols with their property and usage. We also mentioned the advantages of asynchronous circuits and the critical problem of synchronous circuits; i.e. power consumption, which is potentially solved in asynchronous circuits as idle modules are off. We also mentioned lack of asynchronous tools as a drawback for asynchronous circuits. This paper presented a method to solve this problem in a class of DI circuits.

By changing the C-elements of the DI circuits with a structure, which can be supported by synchronous EDA tools, we have achieved considerable fault coverage, i.e., 92.59%. Another advantage of this method is that, the lack of asynchronous tools is solved to some extent. Area overhead is the cost we paid for high fault coverage and other advantages.

Figure 11. Fault Coverage Comparison

7. Acknowledgment

This work is partially supported by a research grant from the Iran Telecommunication Research Center.

8. References


