DFT for Fast Testing of Self-timed Control Circuits

Sandeep Pagey
Cadence Design Systems (I) Pvt. Ltd.
NEPZ, Noida 201 305, India

Ajay Khoche and Erik Brunvand
Dept. of Computer Science, University of Utah
Salt Lake City, UT 84112, USA

Abstract

In this paper, we present a methodology to perform fast testing of the control path of self-timed circuits [9]. The speedup is achieved by testing all the execution paths in the control simultaneously.

The circuits considered in this paper are those designed using an OCCAM based circuit compiler [2]. This Compiler translates an OCCAM program description into an interconnection of pre-existing self-timed macro-modules [2, 10]. The method proposed involves modifying certain modules and structures in such a way that the circuits obtained by translation using these modified modules are testable in above mentioned way.

1 Introduction

Asynchronous circuits have been receiving a renewed interest by the circuit designers in the recent past. This is due to the advantages offered by these circuits such as freedom from clock related problems, possibility of low power consumption, simpler composition and average case performance as compared to worst case performance in synchronous circuits. There have been many recent efforts in the area of asynchronous circuits specification, synthesis and verification. Testing Asynchronous circuits however is a relatively new area.

In this paper we discuss a fast testing method for self-timed control circuits. In particular we focus our attention on self-timed circuits synthesized using an OCCAM compiler [2]. These circuits are composed of a predesigned set of library components called macro-modules [2, 10]. The OCCAM program is translated automatically into an interconnection of these modules, which implements the behavior described by the OCCAM program.

The method proposed in this paper uses the property of the self-timed circuits that the circuit halts in the presence of the faults because of the handshake required by the self-timed protocol. A faulty circuit will fail to complete the handshake and will thus halt. Testing the control path requires each path in the control flow graph of the circuit to be activated. The approaches reported in the literature test each of these paths separately (one by one). We propose to exploit the distributive nature of the control path of this type of circuits to speed up the testing by exciting multiple paths concurrently. Modifications to the modules have been proposed to achieve this effect.

2 Self-Timed Circuits

Self-timed circuits are a subset of a broad class of asynchronous circuits. These circuits generate completion signals to indicate that they are finished with their processing [9]. A signalling protocol used with the completion signalling allows self-timed systems to be composed of circuits which communicate using self-timed protocols. Self-timed protocols are often defined in terms of a pair of signals, one to request or initiate an action, and another to acknowledge that the requested action has been completed. One module, the sender, sends a request event (*Req*) to another module, the receiver. Once the receiver has completed the requested action, it sends an acknowledge event (*Ack*) back to the sender to complete the transaction. The circuits in our library use two-phase transition signaling for control. Two-phase transition signaling [9] uses transitions on signal wires to communicate the *Req* and *Ack* events described previously. Only the transitions are meaningful; a transition from low to high is the same as a transition from high to low and the particular state, high or low, of each wire is not important.

2.1 Self-Timed Module Library

As mentioned earlier, the circuits of interest in this paper consist of an interconnection of macro-modules. The specific set of modules used are those described in [2, 10]. These modules are described in brief in this section and are shown symbolically in Figure 1. All these modules follow two-phase transition signalling described above.

The control modules include circuits that act as OR gates for transitions (implemented using an XOR) and AND gates for transitions (implemented using a C-element). A transition-OR gate will produce a transition at its output whenever there is a transition at either input. An AND requires transitions at both inputs before producing a transition at the output. Also included are modules that steer transitions depending on a Boolean input signal (a Select module) or alternate transitions on the output for each transition on the input (a Toggle module). A Call module allows multiple sender circuits to have access to a common receiver circuit by implementing a hardware subroutine call. The Call module requires that the multiple requests be mutually exclusive so that it need not perform arbitration.

0-8186-7129-7/95 $4.00 © 1995 IEEE
The Fourth Asian Test Symposium

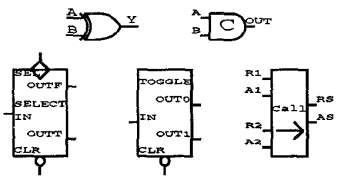

Figure 1: Control Modules for Self-Timed Designs

2.2 Synthesis Method

The Synthesis method of [2] involves description of the circuit in a subset of OCCAM. Once the circuit is described in OCCAM it is translated automatically into an interconnection of macro-modules described earlier. The translation is syntax directed in the way that each construct of the language is translated into a predetermined subcircuit. There are five constructs in the subset used: *SEQ, PAR, IF, WHILE* and *ALT*. *SEQ* and *PAR* constructs are used for sequential and parallel composition of the processes. A process is a subcircuit made of the library modules with a *Req* input and an *Ack* output. The subcircuits corresponding to the *IF*, *WHILE* and *ALT* constructs are shown in Figure 4, 5, and 6. In the *IF* construct the conditions are checked in sequence at the *sel* input of Select modules and if found true the corresponding process is executed on the true branchof the Select Module. In the *WHILE* construct, after initiation on the *Req* input the loop body process is executed until the loop condition on *Sel* input becomes false. The *ALT* construct is used to implement guarded processes. These processes require their associated guards to be true before the process is invoked. This construct differs from conditional construct in the sense that in an *IF* statement at most one process is invoked while in an *ALT* statement exactly one process is executed. Also the control remains within the statement until a guard becomes true.

3 Related Work

Testing asynchronous circuits is a relatively new area. Very few attempts have been made so far. Efforts directly related to our work are described in this section.

Hazewindus [3] has analyzed the delay insensitive circuits built using Martain's [5] synthesis methods. He found that a circuits halts for most of the faults. He outlined a procedure to generate tests to activate each path in control circuits. However he also found that there were faults for which the circuits does not halt. These are the faults on isochronic forks [3]. He had proposed an ad hoc scan approach to test these faults.

Roncken and Saejis [8] had proposed a method to test the circuits built using Tangram compiler developed at Philips labs. This synthesis method translates Tangram descriptions into circuits as an interconnection of predesigned modules like in the synthesis methods used by us. The test method proposed also used the property of these circuits that the circuit halts for most of the faults. However each path in the control circuit was tested separately. Also the faults on the isochronic forks inside the modules were not tested.

Kudva and Akella had proposed a design for testability method to test the circuits built using SHILPA(A High level synthesis system). This system also translates the circuits described in a language called HopCp into an interconnection of predesigned modules. In their method they proposed a modified design for a Select module which allows the control to be directed on a particular path in the control part. This was done by controlling the *Sel* line of the Select modules through a scan path. In their approach also each path in the control part was tested separately and scanning is required to select a particular path. Also the modules were considered atomic i.e. the faults inside the modules were not considered. The method proposed in this paper is different from the approaches described in the sense that all the paths in the circuits are tested simultaneously and no scanning is involved either. This reduces the test application time. In addition the faults inside the modules on isochronic fork were also considered with the exception of C element.

4 Test Methodology

4.1 Basic Requirements

As mentioned earlier, our method relies on testing all control paths simultaneously. This means that all the paths in the circuit are excited simultaneously. A new path is introduced in the circuit when a branching point or a fork is encountered in the circuit. While all the paths are excited automatically in the fork case, modifications are needed for the branch point where the branches are mutually exclusive. The requirements for our method to be applicable are described below.

1. At a branching point all the branches should be activated. What this means in circuit terms is that both the outputs of Select and Toggle elements should be activated upon a *Req* event at their inputs.

2. When two branches are merged through a Merge element a single event should be produced at the output of the Merge element after both the branches finish their processing. This is required to maintain the self-timed protocol even during the testing.

3. The third requirement is related to non-interference of the control paths. The control paths in general are non-interfering except when sharing of resources occurs. When sharing occurs in the circuits, we have to guarantee that progress on one control path is not stopped because of progress of some other control path. Also the shared module should be executed only once because the multiple execution do not give any additional information and increase the test time unnecessarily [8].

4. The control path should be decoupled from data path during testing so that the control path can be tested separately. In the type of circuits we deal with in this paper, the data interacts with control only on the *Sel* line of Select modules. The control path should be made independent of data

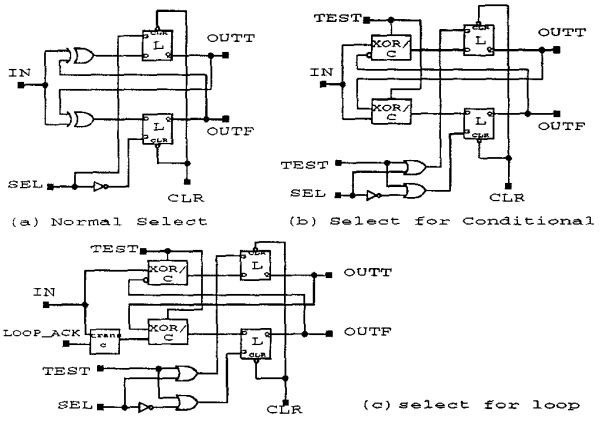

Figure 2: Basic and modified select modules

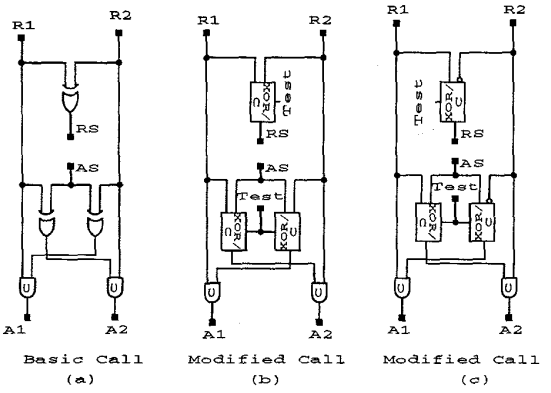

Figure 3: Basic and modified Call modules

by disabling the *Sel* line. This is achieved by modifying the Select element as explained next.

4.2 Modifications to the Modules

The modules in the library are modified to satisfy the requirements mentioned above. The modifications made to individual modules are described in this section.

XOR: The XORs are used at three places in the circuits. First as merge elements for merging control paths. These are also used inside Select and Call modules as shown in Figure 2, 3. When the XOR is used to merge branches of conditionals which are mutually exclusive, it is replaced an XOR/C element. An XOR/C elements acts as an XOR during normal mode but acts as a C element in test mode. This modification is made to satisfy condition 2 mentioned in the previous section during testing when all the branches are activated simultaneously. The behavior of this module as a C element during test prevents the output of this module being activated many times once for each merging branch. The XORs in the Select and Call modules are also replaced by XOR/C elements in the circuit. Those modifications are described below.

Select: The modified Select element is shown is Figure 2. Two different kinds of modifications are required in the Select modules depending upon the construct it is used for in the circuit. However, the modifications required to make the control path independent of data path are common to both designs. The *Sel* line is overridden by the *Test* signal during test mode through the OR gate, such that both the latches are enabled in test mode. Any transition on the input of the latches will be transmitted to their outputs in the test mode.

The design which is used for the *IF* construct is shown in Figure 2(b). In this design the XORs in the original design have been replaced by an XOR/C element as mentioned above. However the *B* input of the XOR/C Element in the upper branch is negated while the XOR/C in the lower branch has both the inputs non-negated. This is done to explicitly sequentialize the transition on OUTT and OUTF to avoid races. In this design, in test mode, an input event causes first an event on OUTT and then on OUTF.

The Select for the loop construct has been modified as shown in Figure 2(c). An additional module *trans-C* has been inserted between IN and the input to lower the XOR/C. This module acts like a C element in the test mode and is transparent for the IN input in normal mode. The *loop-ack* input of this module is connected to the *Ack* of the loop body. This modification causes loop body to be executed only once in the test mode, before generating its *Ack* on OUTF.

Call: The Call module also requires two different designs depending on the context in which it is used in. The case where a Call module is used to share a resource among two mutually exclusive branches in normal mode (e.g. in true and false branches of an *IF* statement), the requirement is that both the branches should progress and the shared resource should be executed once. The need for simultaneous progress arrives because in test mode both the branches get activated. The Call element modified to achieve this is shown in Figure 3(b). In test mode, A single request is produced at RS after both the branches arrive at this module and once the shared resource acknowledges on AS acknowledgements are produced on both the outputs thus both branches make progress.

The other context where a Call is used is to share a resource between processes which are always sequential . In this case the first design will not work because the the first process should be allowed to continue so that second process gets a chance to be activated. However the condition that the shared source should be called only once remains. This is achieved by negating the R2 input to the XOR/C generating RS during test mode so that when a request arrives at R1, a request is produced at RS. Note that the XOR/C element generating the input to the C element with output A1 also has it R2 input negated. This allows an acknowledgement to be produced on A1 upon receipt on AS. Thus in this design the first process is allowed to progress without waiting for the other as in the previous de-

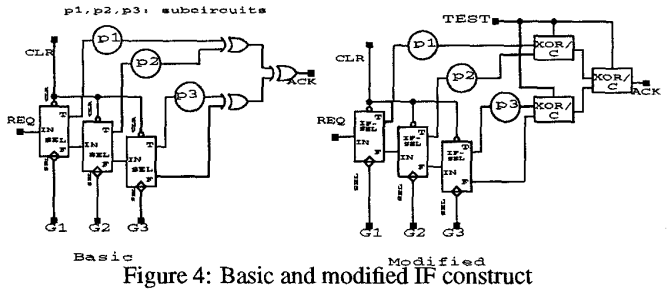

Figure 4: Basic and modified IF construct

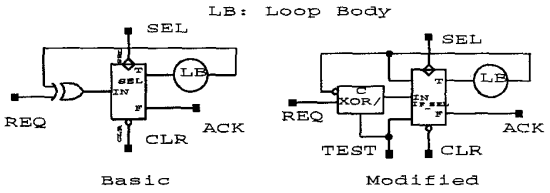

Figure 5: Basic and modified LOOP construct

sign. Now when the second process arrives at this module no event on RS is generated as R1 is still asserted. Whereas an acknowledgement is produced on A2 as the corresponding C element has a 1 at its other output. Thus the second process also progresses with the shared resource not being activated again.

One thing that should be noted at this point is that with the above modifications all the nets are activated once when their inputs are activated once in all modules. Moreover if there is a fault on any net then an acknowledgement for the event which activates that net with value opposite to that of fault, will not be generated. This means that to test a module it is enough to generate two events on its inputs, one rising and one falling. This will activate all the nets in the module with values 0 and 1.

5 Modifications to the Constructs

The modifications described above make the modules testable by generating two events at their inputs. In order to carry that argument at the language construct level the circuit structures corresponding to them have been modified. This allows testable circuits to synthesized which can be tested by generating two events at their inputs. The constructs which have been modified are described below:

IF: An example of the basic construct for a nested IF statement is shown in figure 4(a). Here G1, G2 and G3 are the conditions for the three nested IF statements. P1,P2 and P3 are the bodies of those IF statements. The conditions of this construct are checked in the order G1, G2 and then G3. If a condition is true at a certain point then the remaining conditions are not checked and an acknowledgment is produced for the construct on the *Ack* line through the tree of XORs which merge the acknowledgement from the bodies of IF statements. This construct has been modified by replacing the basic Select modules by the modified version described in the previous section so that both the branches of each Select can be activated. This modification results in all paths in this nested IF construct being activated upon activation of *Req*. The second modification in this construct pertains to XORs used for merging acknowledgements. These have been replaced by XOR/C elements. Thus in test mode a single acknowledgement is produced after all the paths have finished their processing. A fault on any path will result in *Ack* not being produced for either rising event or a falling event and thus will be detected.

LOOP: The LOOP construct has been modified in such a way that in test mode the loop is executed only once and then an acknowledgement is produced on *Ack*. Specifically the basic Select module has been replaced with the modified one for the LOOP as described in the previous section. Also the XOR before the Select has been replaced by an XOR/C gate with its feedback input being negated, which prevents further iterations of the loop. Thus an event on the *Req* input causes the loop body to be executed once and an acknowledgement to be produced on *Ack* line.

ALT: ALT Construct is not discussed in detail here due

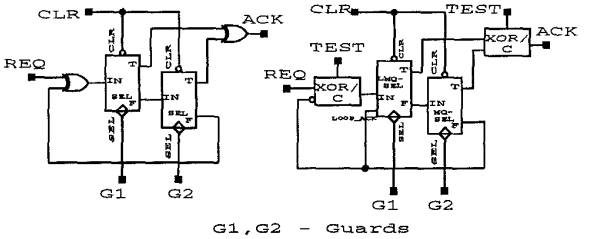

Figure 6: Basic and modified ALT construct

to space limitation, however it can be modified in a way very similar to the IF construct. The Q-Select [2] modules (Which are very similar to the Basic Select modules) are replaced by their testable versions. and the XOR tree for generating *Ack* is replaced by XOR/C tree.

The point to be observed here is that one activation at the input of a construct causes all the nets in that construct to be activated. Moreover if there is any fault on any of these nets then it inhibits the response on the output from being produced for either an invocation with low to high or high to low transitions. This means that to test any construct one simply needs to produce a low to high and high to low events on its inputs.

6 Example

The control part of a self-timed circuit to implement the GCD of two numbers A and B is shown in Fig. 7. This circuit has three possible paths through which the control can flow. In order to test the control path using the earlier methods [3, 8, 4], each path is executed one by one, thereby requiring six transitions on the REQ input. Executing each path one by one also requires setting up appro-

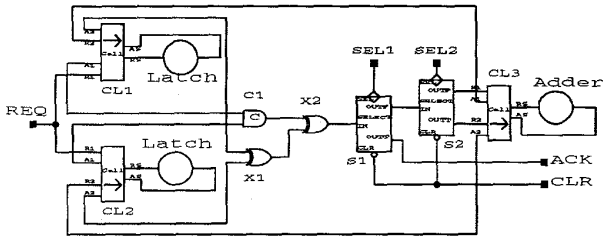

Figure 7: Control path of a GCD circuit

priate data values which allow a particular path to be executed. The method in [4] uses a scan path to control the choice of the path to be executed which requires additional time for scan. The method presented in this paper modifies the circuit such that it can be tested with only two input transitions on the REQ input. The XOR *X1* is replaced by an XOR/C element and XOR *X2* is replaced by an XOR/C element with a negated input as described for LOOP. The CALL modules *CL1* and *CL2* are replaced by a testable Call module of Fig. 3(c), whereas Call module *CL3* is replaced by the one shown in Figure 3(b). The SELECT module *S1* is replaced by the testable SELECT module of Fig. 2(c). The SELECT module *S2* is replaced by the testable SELECT module of Fig. 2(b).

Since the control path is completely isolated from the data path, setting up data values or scanning the SELECT modules is not required.

7 Advantages and Disadvantages

This approach offers following advantages:

Fast testing: The circuits can be tested very quickly as all the paths are tested simultaneously. Also the test application procedure is very simple as compared to scan based techniques. Shorter test time will result in saving expensive tester time.

No test generation: The testing of circuits involves only producing two events at the inputs, so no test generation is involved.

Automation: Such a scheme can be incorporated easily in existing compilers which are targeted towards such modules e.g. OCCAM Based [2] and SHILPA [1], where the testable version of the modules can be used in place of original modules.

General concept: This concept can also be extended to other methodologies for the design of self-timed or delay insensitive circuits, for instance Martain's method [5, 6].

Following are the disadvantages of this method:

Overhead: As with any design for testability method there are overheads associated with it. In this method the overhead comes from replacing XORs with XOR/C gate. Research in underway to design an efficient version of XOR/C gate. Initial design of an XOR/C gate showed around 15% degradation in performance. However one should note that this percentage degradation will not translate into the same percentage degradation for the entire circuit as many modules which form a path are not modified. The area overhead of the initial designs is not much and if we take into account the fact that control typically forms a small percentage of the circuit, the overhead over the entire circuit will be small. For example in the AMULET processor the control occupied only 15% of the entire chip area [7].

8 Conclusions

A fast testing method is proposed to test the control path of self-timed circuits generated by an OCCAM based syntax directed circuit compiler. The method involves modifying library components to allow circuits to be tested quickly. This method, unlike other methods proposed, tests multiple paths in the control unit of the circuit simultaneously. This method alleviates the need for test generation and is easily automatable. It can also be extended to other design methods. Results for various other example circuits are being generated presently. We are also working towards extending concepts presented here to the testing of the data path.

References

[1] V. Akella and G. Gopalakrishnan. SHILPA: a high level synthesis system for self-timed circuits. In *Proc. International Conf. Computer-Aided Design (ICCAD)*, 1992.

[2] E. Brunvand. *Translating Concurrent Communicating Programs into Asynchronous Circuits*. PhD thesis, CMU, 1991.

[3] P. Hazewindus. *Testing Delay-Insensitive Circuits*. PhD thesis, CalTech, 1991.

[4] P. Kudva and V. Akella. Testing Two Phase Transition Based Self-timed Circuits in a High Level Synthesis Environment. *High Level Synthesis workshop* May 1994.

[5] A. J. Martin. Compiling Communicating Processes into Delay-insensitive Circuits. *Distributed Computing*, 1(3), 1986.

[6] S. Pagey Fast Functional Testing of Delay Insensitive Circuits. *Technical Report*, Concordia Univ., Montreal, 1994.

[7] N. Paver. *The Design and Implementation of an Asynchronous Microprocesor*. PhD Thesis, Univ. of Manchaster, UK , 1994.

[8] M. Roncken and R. Saeijs. Linear Test Times for Delay-insensitive Circuits: A Compilation Strategy. In *Proceedings of IFIP Working Conference on Asynchronous Design Methodologies*, Manchester, UK, Elsevier Science, 1993.

[9] C. L. Seitz. System Timing. In *Mead and Conway, Introduction to VLSI Systems*, chapter 7. Addison-Wesley, 1980.

[10] I. Sutherland. Micropipelines. *CACM*, 32(6), 1989.