Testing in the Direct Mapping Domain*

Deepali Koppad, Alex Bystrov, Alex Yakovlev
University of Newcastle upon Tyne

Abstract—***The application of asynchronous circuits has been restricted because of the lack of technology to test them. In this paper we introduce a technique to test circuits obtained by the direct mapping technique from 1-safe Petri nets. Low-level physical faults in the cells implementing Petri net places are analysed and mapped into the high-level specification, a Petri net. This technique uses a "pseudo clock" in order to handle the hazards which may occur under certain types of physical faults. The clock also helps to activate faults which exhibit themselves only under some particular arrangement of signals and to deliver the precise information about the fault location to the test point.***

I. Introduction

Circuits without a clock signal are often referred to as asynchronous or self-timed. The clock replacement is a handshake, whose simplest version involve two signals, Request and Acknowledge. Although asynchronous circuits offer a number of promising features for low power, low EMC, high security applications, their design and testing are more difficult [3,11,12].

Synthesis of these circuits is an important task. Various techniques have evolved through the years. Direct mapping from Signal Transition Graphs (STG) is one such synthesis technique [1,7] which avoids algorithmic complexities and is based on output exposition and environment tracking. It helps reducing output latency.

Based on the technique of Direct mapping, a two level architecture [8] has been introduced comprising a tracker and a bouncer. The former introduces explicit context signals (ECS), one per output signal and works in parallel with the environment (precomputation technique), thus reducing latency. The tracker consists of David cells [2,3], which model places of a 1-safe Petri net derived from the specification. The bouncer uses minimum context signals from the tracker and produces the output. It consists of registers and has minimum impact on latency. Hence, it is necessary to study the tracker in detail in order to reduce the latency and this is the main aim of this paper.

Testing is verifying if the circuit is performing according to the specification by means of physical experiments involving application of test inputs and reading responses. It is checking for faults in the circuit. Because of potential hazards, which are difficult to propagate to the test point, testing of asynchronous circuits is challenging. Since there is no clock in these circuits, controlling and observing each step change in the circuit is not easy. Controllability is the circuit property reflecting the percentage of faults which can be activated (made to produce an error) by applying test inputs. Observability refers to determining the response of the circuit after applying the test patterns.

Faults are classified as stuck-at, delay, bridging, parametric, intermittent [4], which can be single or multiple. Our circuit consists of David cell (DC) structures which are sequential, self-timed, speed-independent [5] circuits. Since DC are simple structures we decided to test them using a special "pseudo clock" in order to filter out hazards and to control the order of signal application. In this paper, we look at the classical single stuck-at fault model with faults located at the inputs of logic gates.

*EPSRC supports this work in project STELLA (ER/S12036)

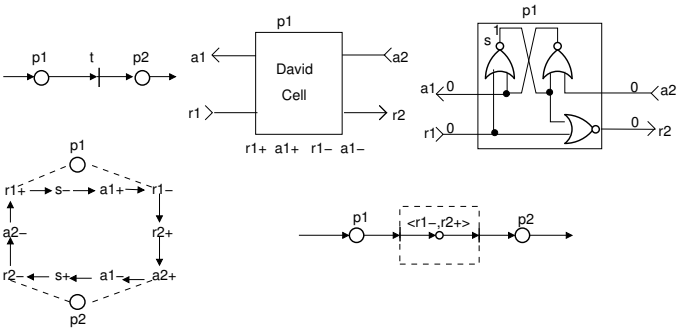


Fig. 1. David Cell Operation

In our approach, we look at the physical faults, i.e. stuck-at-0 (s-a-0) and stuck-at-1 (s-a-1), present in a DC structure which will be mapped at the interface of a DC. The interface faults will subsequently be mapped at the Petri net level. In future we plan to derive tests at the Petri net level.

Section 2 explains the DC structure, method of testing it, physical faults present in it, how these can be mapped to interface faults and subsequently to the Petri net level. In section 3, we introduce some typical David Cell structures and section 4 gives the conclusion.

II. David Cell

David cells were introduced by Rene David [2] and used by [3] in Direct Mapping and later by [1]. The idea introduced in [3] is to associate each place in a Petri net with a David Cell whose circuit diagram is shown in Fig. 1(c)

Place p1 in Fig. 1(a) is replaced by two wires (request, acknowledgement) and a four phase handshake interface (Fig. 1 b). Interface (r2, a2) of previous stage is connected to interface (r1, a1) of next stage. Fig. 1(c) uses logic 1 as active level. Fig. 1(d) shows the STG for the token to move from place p1 to place p2. The PN can be modelled as shown in Fig. 1(e). The dotted rectangle depicts the transition between p1 and p2, containing an internal place where the token 'disappears' for the time $\tau_{r1-\rightarrow r2+}$. This corresponds to one gate delay and so is considered as negligible.

A. Testing David Cell Structures

The structures composed of David cells are tested in our approach by mapping physical faults of the logic gates into the interface and further into the initial specification, a Petri net. The

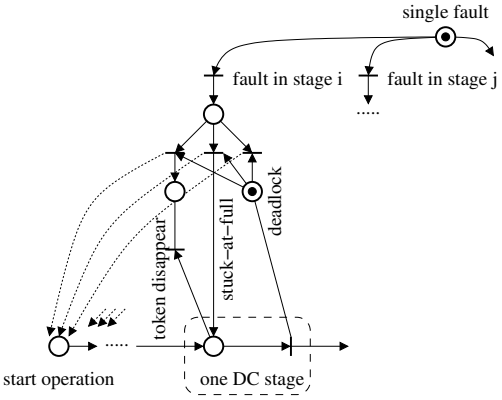


Fig. 2. Fault model representation at the Petri net level

fault model of a DC represented as a Petri net is shown in Fig. 2. It captures three main types of errors created by a physical fault. A token disappearing error takes place when the DC executes its input handshake, but do not start its output handshake. This causes a deadlock in the system. A stuck-at-full error happens if a DC has its output request wire in the stuck-at-active state, which starts the output handshake prematurely and never finishes it. This also results in a deadlock. Finally there can be an error when a DC waits for a token to arrive, then starts its output handshake and freezes it. The model also reflects that only one type of error may occur in a single DC, furthermore, the faults are introduced before the system started its operation.

As one can see in Fig. 2 all faults result in deadlocks and hence can be tested by running a single cycle of operation (assuming the system is live and deterministic).

Apart from these main types of errors there exist two faults causing glitches at DC outputs. These are analysed below. In order to test them certain timing assumptions should be introduced. In our approach we use what we call a "pseudo clock" in order to slow down the transitions and capture the signals after the glitches have settled down.

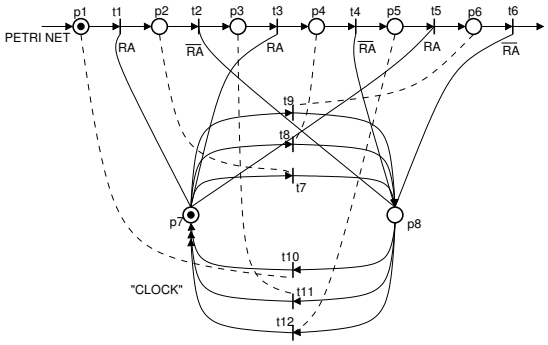
This idea is illustrated in Fig. 3(a). The solid lines are read-arcs, from the clock to the transitions of the PN. The dashed lines are feedback from the places of the PN to the transitions of the clock. When a token is present in place p7 of the clock, transitions t1, t3, t5, t7, t8 and t9 are enabled. Transitions t2, t4, t6, t10, t11 and t12 are enabled when a token arrives in place p8.

In the above example, transition t1 fires first, moving token from place p1 to p2, which enables transition t7 and hence token from p7 moves ahead to p8. So now transition t2 fires and token moves from p2 to p3. This continues till token finally arrives at t6. The introduction of clock makes the time discrete, and in this example, the token arrives at the output after 5 half-cycles of the clock. Thus, errors occurring in the main structure can be detected. Every place and transition of the PN is associated with a DC and an AND gate respectively (Fig. 3(b)).

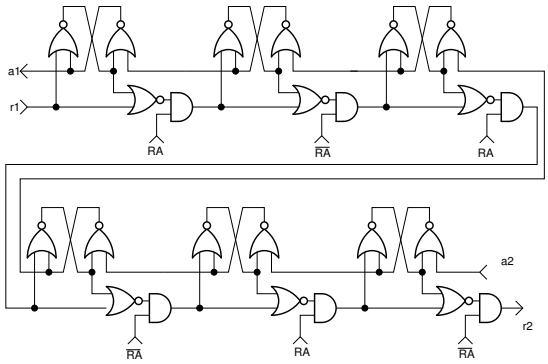
As DCs are speed-independent, gate delay faults do not result in errors.

B. Physical faults

As mentioned earlier, only single s-a-faults at the gate inputs are considered here. Suppose, we have a 2-input AND gate, with



(a)



(b)

Fig. 3. Structure of PN and "clock"

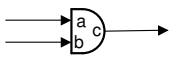


Fig. 4. AND Gate

TABLE I
TEST PATTERNS FOR AND GATE

a	b	c	Faults Tested
1	1	1	a=0|b=0|c=0
0	1	0	a=1|c=1
1	0	0	b=1|c=1

inputs a, b and output c (Fig. 4).

In order to test this gate for s-a-faults (s-a-0, s-a-1), we apply the test patterns as shown in Table I. Similarly, test patterns for an OR gate can be generated.

C. Testing a David Cell

A David Cell in Fig. 1(c) consists of NOR gates only. The AND gate in Fig. 5 is needed to implement the feedback from the bouncer as in [1] and also can be used to implement the pseudo clock technique as described above.

Table II shows the physical faults (column 2), the behaviour of these faults in a single David cell (column 3), errors caused by

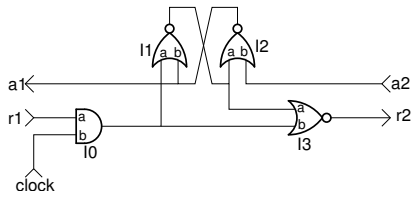


Fig. 5. Single DC with Clock

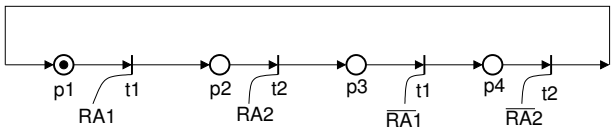


Fig. 6. Linear Structure

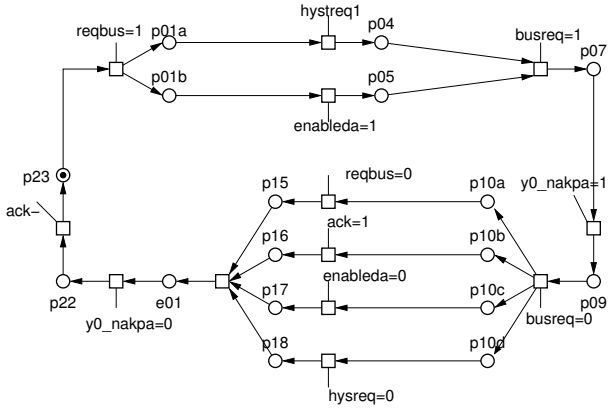


Fig. 8. Original Specification

them in the structure without clock (column 4) and in a clocked David cell chain (column 5) (Fig. 5) and how physical faults can be mapped at the interface.

Table II shows that 83% of faults are tested in a single cycle of circuit operation without implementation of any testability features. Adding clock helps to test one more fault (#8), giving 93% testability. The fault #16, however, requires delaying the reset phase of the input handshake, which can be implemented by an additional AND gate in the circuit of a1.

III. Typical David Cell Structures

In this section certain models which include linear, fork, join, choice and merge structures will be studied. These models will be tested for the faults mentioned in Table II.

A. Linear structure

Linear Structures are simple and easy to test. All faults mentioned above are easily detected.

Consider Fig. 6, with 4 places and 4 transitions.

Associating each place with a DC and the transition to an AND gate we have structure as shown in Fig. 7. The circuit is cut at the initial marking.

The circuit is reset and the output is observed at r2 after 4 half-clock cycles, without applying the clock inputs. Next apply all clock inputs and observe the output at r2.

The circuit is then initialised, i.e. a token is inserted at p1 (r1=1) and the output is observed at r2, both with and without applying the clock inputs.

To test the AND gates, insert a token at p1 (r1=1), and delay each clock input (read-arc RA) by one half-clock cycle and observe the output at r2.

Observation

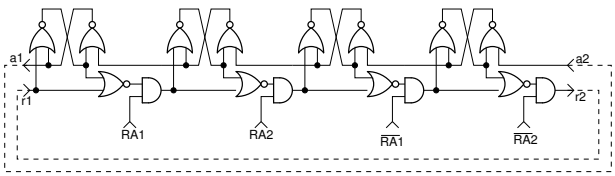


Fig. 7. Linear Structure with DC

Without inserting a token, we observe the output at r2 after 4 half-clock cycles, with and without applying clock inputs, we should not receive any token. Then we insert a token, and observe the output without and with the clock inputs. If the token arrives at output r2 before the expected time, i.e. 4 half cycles of the clock, then there is a s-a-1 fault in the tracker. If the token does not appear at all at r2 then there is a s-a-0 fault.

B. Fork and Join Structures

Let us consider an example which involves Fork and Join Structures, shown in Fig. 8. It is a benchmark called nakpa. Using the tool from [9] the Petri net is optimised and the redundant places are removed.

Cut the circuit at the initial marking and then associate each place with a DC (Fig. 9). The transitions are replaced by AND gates, where each AND gate has one input as clock.

1. Reset the circuit and observe the output at r2 after 9 half-clock cycles.
2. Apply all the clock inputs and observe the output at r2 after 9 half-clock cycles.
3. Then set the tracker, i.e. insert a token and observe output at r2 without applying the clock.
4. Insert a token, apply all clock inputs and observe the output. Token should arrive after 9 half-clock cycles at r2.
5. Next to test the AND gates, i.e. the inputs where the clock is connected, delay the token at each AND gate, one at a time. Eg. Insert a token at p23 (r1=1) (Fig. 9). Delay reqbus=1 by one half-clock cycle and observe the output at r2 after 9 half cycles.
6. Now to test the fork paths, use a technique similar to 5 i.e. delay the token in each path one at a time. Eg. insert a token at p23 (r1=1), delay enableda=1 by one clock cycle, apply all the other clocks correctly and observe the output at r2 after 9 half-clock cycles. If the token arrives at the expected time (9 half clock cycles), then the AND gate with input enableda=1 is s-a-1. If a token does not arrive at all then there is a s-a-0 fault. Under fault-free conditions token should arrive at output r2 after 10 half-clock cycles.
7. Similarly all the AND gates and fork paths can be tested, i.e. by delaying each input by one half-clock cycle, except four gates (a, b, c, d) (Fig. 9). For these four gates special techniques are needed which are explained later.

Observation

TABLE II
PHYSICAL FAULTS

#	Physical Faults	Single DC	DC chain errors, no clock	DC clocked chain errors
1	I0 a=1	-	-	Continuous firing: Every time clock input arrives the AND gate is activated and hence it will fire.
2	I0 a=0	-	-	r1 s-a-initialisation deadlock: AND gate is never activated hence the token cannot go further
3	I0 b=0	-	-	token jumps forward by 2 stages
4	I0 b=1	-	-	same as #2
5	I1 a=1	s=0, a1=1 on initial	deadlock: acknowledgement is received without making a request, hence resulting in deadlock	Deadlock in Predecessor: r1 can be set to 1 only when a1=0, since a1 is already 1 on initialisation token cannot go ahead
6	I1 a=0	r2 s-a-initial	deadlock: NOR gate cannot be activated resulting in deadlock	r2 s-a- initialisation (deadlock): NOR gate I1 is never activated and hence token will not go ahead.
7	I1 b=1	same as #5	same as #5	same as #5
8	I1 b=0	glitch at r2 after r1 reset	glitch at r2: this is a token disappearing error, where the input handshake is completed but the output handshake does not start. The clock is used to detect this fault.	r2 s-a-initialisation (deadlock): even though a glitch will occur but the time period is small and hence the AND gate of the next stage will not get activated, resulting in deadlock.
9	I2 a=1	a1 s-a-initial	deadlock: Explanation same as #6	a1 s-a-initialisation (deadlock): a1 will never be set to 1 i.e. NOR gate I2 is never activated, hence resulting in deadlock.
10	I2 a=0	a1=1 on initialisation	deadlock: Explanation same as #5	deadlock in predecessor: (Explanation same as 5)
11	I2 b=1	same as #9	same as #9	same as #9
12	I2 b=0	r2 stays active after receiving a2	deadlock: r2 is 1 all the time, indicating presence of token (stuck-at-full fault)	continuous firing: (Explanation same as 1)
13	I3 a=1	r2 s-a-initial	deadlock: Explanation same as #6	r2 s-a-passive (deadlock): as one input to I3 is always 1, output of I3 will never be set to 1
14	I3 a=0	r2 active on initialisation	deadlock: Explanation same as #12	continuous firing: r2 is always 1 so every time the clock input of the next stage is applied the AND gate is activated and it fires.
15	I3 b=1	same as #13	same as #13	same as #13
16	I3 b=0	r2+ $\parallel$ (a1+. r1-), hazard at a1 (early reset is possible if $\tau_{r2+,a2+} < \tau_{a1+,r1-}$)	gitch at a1 possible: need to delay resetting of a1 by including an AND gate	Need to delay setting of a1 . Not present in fast implementation of DC.

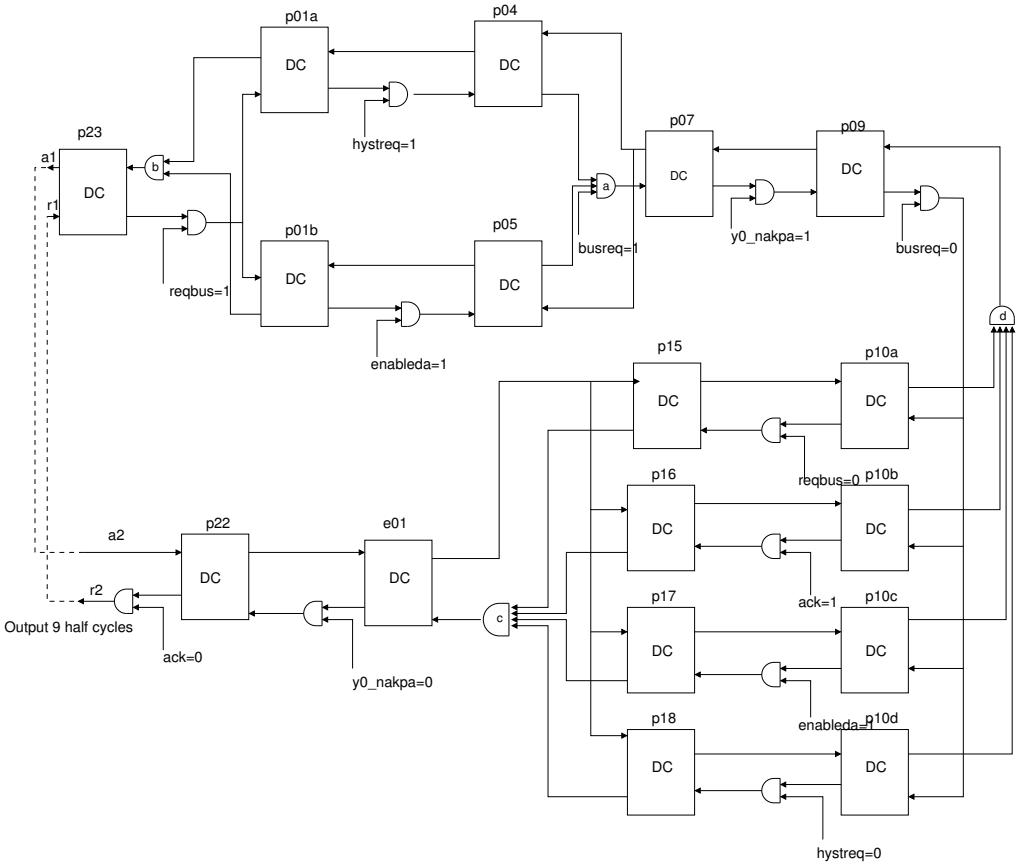


Fig. 9. Replacing places with DC

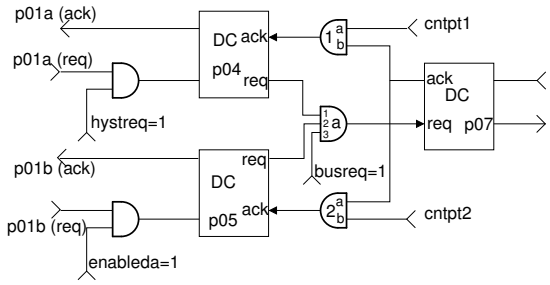


Fig. 10. Testing AND gate *a* using gates 1 & 2

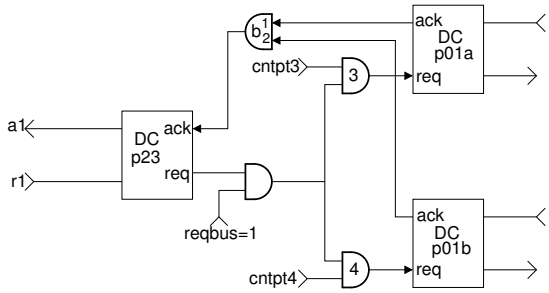


Fig. 11. Testing AND gate *b* using gates 3 & 4

No token should be received, with or without the clock, at output r2 after 9 half clock cycles when no token in inserted at input. When a token is inserted, we can observe that if the token does not appear at r2 then there is s-a-0 fault in the circuit. If it arrives early (i.e. before 9 half clock cycles) at r2 then there is a s-a-1 fault in the circuit.

Now we explain the special technique needed to test gates a, b, c, and d.

Gate *a* (Fig. 10)

In order to test gate *a,* gates 1 and 2 are used. The inputs to these gates are acknowledgement from p07 and *control points* (cntpt1 and cntpt 2). The control points are used to delay the acknowledgement from p07 to p04 and p05.

Initially, cntpt1=1, cntpt2=0. Gate *a* is activated when token arrives at p04, p05 and busreq=1. Request pin of p07 is set high. When p07 is ready to accept the token it sets its acknowledge pin to high i.e. pins 1b and 2a are set to 1. As cntpt1=1, gate 1 is enabled and gate 2 is disabled. So p04 receives an acknowledgement and then resets it request pin. So now gate *a* should get disabled and reset the request pin of p07. If this does not occur, i.e. request pin of p07 is still high then we can conclude that pin 1 of gate *a* has a s-a-1 fault.

Next we make cntpt1=0 and cntpt2=1, and repeat the above procedure for s-a-1 fault at pin 2 of gate *a*.

Similarly to test gate *c* we can include 4 gates and 4 control points.

Gate *b* (Fig. 11)

Input of gate *b* are acknowledgements from p01a and p01b.

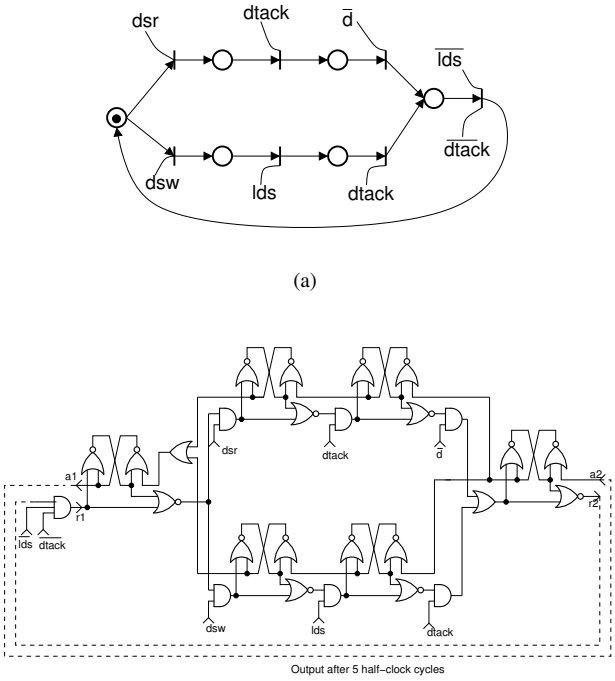


Fig. 12. VME bus

Gates 3, 4 and two *control points* (cntpt3 and cntpt 4) are used to test gate *b*.

Initially cntpt3=1, cntpt4=0. Token arrives at p23 and reqbus=1, so the first AND gate is activated. Since cntpt3=1, gate 3 is activated and a request is sent to p01a. When it is ready to accept the token it sends an acknowledgement on pin b1. But since cntpt4=0, no request is made to p01b and so pin b2 is not activated. Gate *b* should be disabled. But if pin b2 is s-a-1 then gate *b* is activated and p23 receives an acknowledgement.

Next, we make cntpt3=0 and cntpt4=1 and test pin b1 for s-a-1 fault.

Similarly, we can include 4 gates and 4 control points to test gate *d*.

During the normal operation of the circuit, cntpt1, cntpt2, cntpt3 and cntpt4 are all set to 1.

C. Choice and Merge Structures

Now we will see how the faults mentioned in Table II can be detected in choice and merge structures. For these structures we make use of the well known benchmark VME bus controller. Fig. 12(a) shows the petri net after removing the redundant places and optimising the structure.

Replacing each place by a DC and the transition by AND gate, we have a structure as shown in Fig. 12 (b)

In order to test choice and merge, we use a technique similar to fork and join.

1. First reset the circuit and observe the output at r2 after 5 half-clock cycles. (No clock inputs are applied)
2. Apply all clock inputs and observe the output at r2 after 5 half-clock cycles.
3. Then set the circuit, i.e. insert a token, without the clock inputs, observe the output at r2 after 5 half-clock cycles.
4. Insert a token, apply all clock inputs and observe the output at r2.
5. Next to test the input to the AND gates, delay each clock input, by one half-clock cycle and observe the output at r2 after 5 half-clock cycles.
6. Finally, to test the choice paths, stop the token from passing in one branch at a time. Observe the output at r2 after 5 half-clock cycles.

Observation

When no token in inserted at input, no token should be received at output, with or without clock. Again with and without clock, after a token is inserted if more than one token is received at the output at r2 then there is a s-a-1 fault. If no token arrives at output r2 then there is a s-a-0 fault.

IV. Conclusion

We can say that 83% of the faults occurring in a DC can be tested without the use of any additional features in a single cycle of circuit operation. Introduction of the "pseudo-clock" improves testability to 93%. In order to achieve full testability, introduce an AND gate in the reset phase to delay the resetting of acknowledgement *a1*. All the faults occurring in a DC could be mapped to the higher level specifications, i.e. PN level. There are a few issues that need further studies such as partitioning of DC structures into acyclic parts, implementation of test points and proper algorithmic techniques. Structures such as linear, fork, join, choice and merge were studied.

References

[1] A. Bystrov and A. Yakovlev, *Asynchronous Circuit Synthesis by Direct mapping: Interfacing to Environment.* Technical Report CS-TR-743 (accepted to ASYNC2002), University of Newcastle upon Tyne, UK 2001.

[2] Rene David, *Modular design of asynchronous circuits defined by graphs*, IEEE Transaction on Computers, 26(8):727-737, August 1977.

[3] Michel Kishinevsky, Alex Kondratyev, Alexander Taubin and Victor Varshavsky, *Concurrent Hardware: The theory and Practice of self-timed design*, Series in Parallel Computing. John Wiley & Sons, 1994.

[4] Gordon Russell, Ian Sayers, *Advanced Simulation and test methodologies for VLSI design*, London: Van Nostrand Reinhold (International), 1989.

[5] Al Davis, Steven M. Nowick*, An Introduction to Asynchronous Circuit Design*

[6] Petlin O, *Random Testing of Asynchronous circuits*, MSc Thesis, University of Manchester, 1994.

[7] Lee A. Hollaar, *Direct Implementation of asynchronous control units*, IEEE Transactions on Computers, C-31(12):1133-1141, December 1982.

[8] A.Bystrov, A. Yakovlev, *Synthesis of Asynchronous Circuits with predictable latency*, Technical Report CS-TR-754, University of Newcastle upon Tyne, 2001.

[9] D. Sokolov, A. Bystrov, A.Yakovlev, *STG Optimisation in Direct mapping of Asynchronous circuits*, University of Newcastle upon Tyne, DATE'03 March, 2003.

[10] Henrik Hulgaard, Steven M Burns and Gaetano Borriello, *Testing Asynchronous Circuits, A Survey,* Department of Computer Science and Engineering, University of Washington, Seattle, Technical report 94-03-06.

[11] Scott Hauck, *Asynchronous Design Methodologies: An Overview,* Department of Computer Science and Engineering, University of Washington.

[12] Alexandre V. Yakovlev and Albert Koelmans, *Petri Nets and Digital Hardware Design,* Department of Computer Science, University of Newcastle upon Tyne.

[13] A. V. Yakovlev, A. M. Koelmans, A.Semenov, D. J. Kinniment*, Modelling, analysis and synthesis of asynchronous control circuits using Petri nets*, Integration, the VLSI journal 21 (1996) 143-170.