DESIGN FOR TESTABILITY OF AN ASYNCHRONOUS ADDER

O. A. Petlin, C. Farnsworth, S. B. Furber

1. Introduction

Modern technological processes for producing VLSI circuits have created an opportunity to exploit the advantages of asynchronous circuits. Compared to their synchronous counterparts, asynchronous circuits have the potential for lower power consumption, offer greater design flexibility, exhibit average rather than worst-case performance and have no problem with clock skew [Lav93, Hauck95]. Asynchronous circuits can be divided into three major groups depending on the delay model assumption chosen: delay-insensitive, speed-independent and bounded-delay circuits [Lav93, Birt95, Brzo95]. In delay-insensitive circuits, gate and wire delays are unconstrained but finite. Speed-independent circuits also operate correctly regardless of their gate delays, but signal transmissions along their wires are assumed to be instantaneous. All delays within bounded-delay asynchronous circuits are constrained. Data in asynchronous circuits can be represented using either dual-rail or single-rail data encoding techniques. In the dual-rail data representation each bit of data is encoded using two wires. In four-phase dual-rail encoding, a high logic level on the ‘one’ or ‘zero’ wire and a low logic level on the corresponding ‘zero’ or ‘one’ wire indicates the transmission of a one or a zero respectively. If both data wires are set to zero data is not valid. The state ‘11’ is illegal. In the single-rail encoding a bit of data is represented by the logic level on one wire.

Most asynchronous circuits communicate using signalling protocols which use ‘request’ and ‘acknowledge’ signals. There are two basic signalling protocols which use two-phase or four-phase signalling. According to the two-phase protocol every transition on a control wire indicates an event. In the four-phase signalling protocol, both the request and acknowledge signals must return to zero before the next handshake procedure between the sender and the receiver starts. The data must be valid before a request is sent to the receiver. An asynchronous circuit with single-rail data encoding requires that the request signal is generated when the data is stable on the outputs of the sender and remains stable until the acknowledge signal is generated. This is called the bundled data constraint [Suth89, Birt95]. Using the bundled-data approach, the AMULET group in the Department of Computer Science at the University of Manchester has designed the AMULET1 microprocessor, an asynchronous implementation of the ARM6 RISC microprocessor [Furb94, Birt95]. The AMULET1 chip fabricated by GEC Plessey Semiconductors Limited demonstrates the practical feasibility of designing complex asynchronous VLSI circuits. However, the testability issues of asynchronous VLSI circuits must also be addressed before their commercial potential can be realised.

A fault model is used to describe the behaviour of a faulty digital circuit. The stuck-at fault model describes a faulty circuit at the gate level and is widely used to describe fabrication faults in a circuit [MClus86, Russ89]. According to this model, a faulty wire is stuck at one or stuck at zero if it is permanently connected to the power supply voltage (V_{dd}) or ground (V_{ss}) respectively. A stuck-at fault on a path in the circuit prevents any signal transitions along it. As a result, stuck-at faults in delay-insensitive circuits where all transitions are acknowledged cause the faulty circuit to halt [Dav90, Hulg94]. This situation is easy to identify by the absence of activity on the outputs of the circuit when it operates normally; this is called the self-diagnostic property. Speed-independent circuits exhibit the self-diagnostic property only for stuck-at output faults [Haz92].

Several design-for-testability techniques for asynchronous circuits have already been reported. A test strategy for stuck-at faults in handshake circuits has been described [Ron93], in which it was shown that handshake circuits can be tested in linear time. This work has been extended by adapting a partial scan test technique for test-

O. A. Petlin and S. B. Furber are with the Department of Computer Science, University of Manchester (UK)
C. Farnsworth is with Cogency Technology UK, Bruntwood Hall, Cheadle (UK)

Table 1: *Truth table for the full adder*

Inputs			Outputs	
A	B	Cin	Sum	Cout
0	0	0	0	0
0	1	0	1	0
1	0	0	1	0
1	1	0	0	1
0	0	1	1	0
0	1	1	0	1
1	0	1	0	1
1	1	1	1	1

Table 2: *Truth table for the full adder carry output*

Inputs		Output
A	B	Cout
0	0	0
0	1	Cin
1	0	Cin
1	1	1

ing an asynchronous digital compact cassette (DCC) error corrector decoder [Ron94]. Unfortunately, the reported results were obtained for testing dual-rail data paths which are relative easy to test. A scan design technique has been suggested by Wey et al. to design asynchronous finite state machines for testability [Wey93]. Several reports have addressed the testing of micropipelines [Khoc94, Pet95]. These techniques allow the test complexity of asynchronous sequential circuits to be reduced to just testing their combinational logic. However, the reported results did not address the testability issues of asynchronous circuits with data dependent control.

In this paper we investigate the testability properties of an asynchronous adder with data dependent control. The rest of the paper is organised as follows: Section 2 discusses the design of the AMULET1 asynchronous adder; the testability issues of a single-rail asynchronous adder are considered in Section 3; Section 4 and 5 address different aspects of the design and test of an asynchronous adder designed using dual-rail and hybrid data encoding; a case study of an asynchronous comparator is described in Section 6; and, finally, Section 7 concludes the paper.

2. The AMULET1 asynchronous adder

An asynchronous ALU is a major element in the AMULET1 microprocessor. It has been shown that about 80% of the operations performed by the ALU require different forms of addition [Gars93]. The correct performance of the adder as the ‘busiest’ part of the asynchronous ALU is therefore important for the correct functioning of the AMULET1 design as a whole.

Three input bits are used to implement a one-bit addition: two data bits and one *carry-in* bit which is effectively the *carry-out* signal from the previous stage of the multi-bit adder. The complete truth table of a 1-bit full adder is shown in Table 1. The performance of the multi-bit adder depends on the propagation speed of the *carry* signal through its stages. Table 2 illustrates the truth table for the carry output of the 1-bit full adder. According to

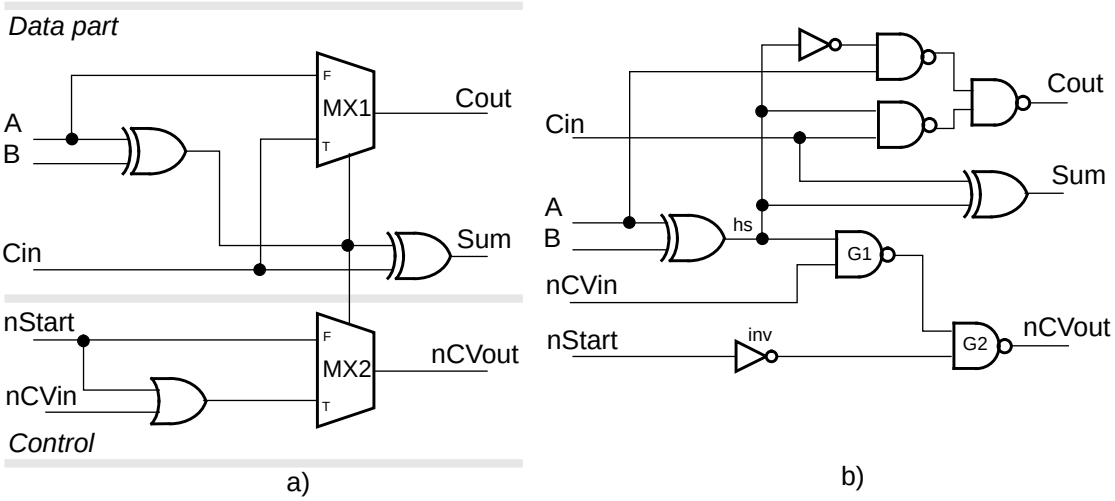

Figure 1 : *Single-rail implementation of an asynchronous 1-bit full adder: a) using multiplexers; b) using logic gates*

this table the *carry-out* signal can be predicted in half of the possible input combinations. This allows the correct *carry-out* signal to be generated without waiting until a *carry-in* signal is produced by the previous stage of the adder. This technique has been used in the implementation of the AMULET1 adder [Gars93]. In the AMULET1 asynchronous adder, addition results are ready when all the *carry-out* signals are ready. The carry chain of the adder is implemented using dual-rail data encoding where the readiness of the *carry-out* signal is identified by a transition on one of its two data wires. Since the *carry-out* signal of the AMULET1 adder is data dependent and data values which cause long carry propagation paths are relatively rare the adder itself exhibits average rather than worst case performance [Gars93].

3. A single-rail asynchronous adder

Figure 1a shows the implementation of a single-rail asynchronous 1-bit full adder using multiplexers. The adder design consists of distinct data and control parts. The data path of the adder produces an addition result on its *Sum* output and generates a *carry-out* signal on its *Cout* output. Note that the *carry-out* function is implemented according to Table 2. The control part of the adder is designed to indicate when a *carry* output is ready to be read by the environment. When the data is ready on inputs *A* and *B* a *start* signal is generated on the *nStart* input which is active low. If the values on the *A* and *B* inputs are equal the *start* signal is passed to the *carry-valid* output of the adder. If not, an active low *carry-valid-in* signal is transmitted from the *nCVin* input through the OR gate and multiplexer *MX2* to the *nCVout* output. The control part of the adder follows the four-phase signalling protocol. Figure 1b illustrates the gate level representation of this asynchronous 1-bit adder with single-rail data encoding. This adder performs in the same manner as described above.

The design of an asynchronous single-rail 8-bit adder is shown in Figure 2. In this design all the 1-bit full adders are connected together in a chain where the *carry* output and the *carry-valid* output of the previous 1-bit adder are connected to the *carry* input and the *carry-valid* input of the following 1-bit adder respectively. The *carry-valid* output of the adder (*Ack*) is produced on the inverted output of the 8-input symmetric C-element, the inputs of which are connected to the corresponding *nCVout* outputs of the 1-bit adders. The *carry-out* signal of the last 1-bit adder is used as the *carry* output of the 8-bit adder. The global *start* signal is connected to all of the 1-bit adders. The first adder (*Ads0*) does not have a *start* input since its *carry-valid* input is connected to the global *start* signal. The *start* signal from the *nCVin* input of adder *Ads0* is delayed for enough time for the *carry-out* signal to be stable before it is passed directly to the *nCVout* output.

A request for addition is sent by the environment on the *Req* input of the adder. When the data is ready on the *A* and *B* inputs two acknowledge signals are generated on inputs *AckA* and *AckB* of the two-input symmetric C-element. When the output of the C-element is set high an active low *start* signal is transmitted to the corresponding inputs of all the 1-bit adders. A rising event on the *Ack* output of the 8-input C-element acknowledges the completion of the addition. Once the results are read the request signal is returned to zero on the *Req* input. As a result, acknowledge signals on inputs *AckA* and *AckB* are set to zero. The two-input C-element is reset and the global *start* signal goes high. The handshake procedure is completed when the acknowledge signal on output *Ack* of the adder is reset.

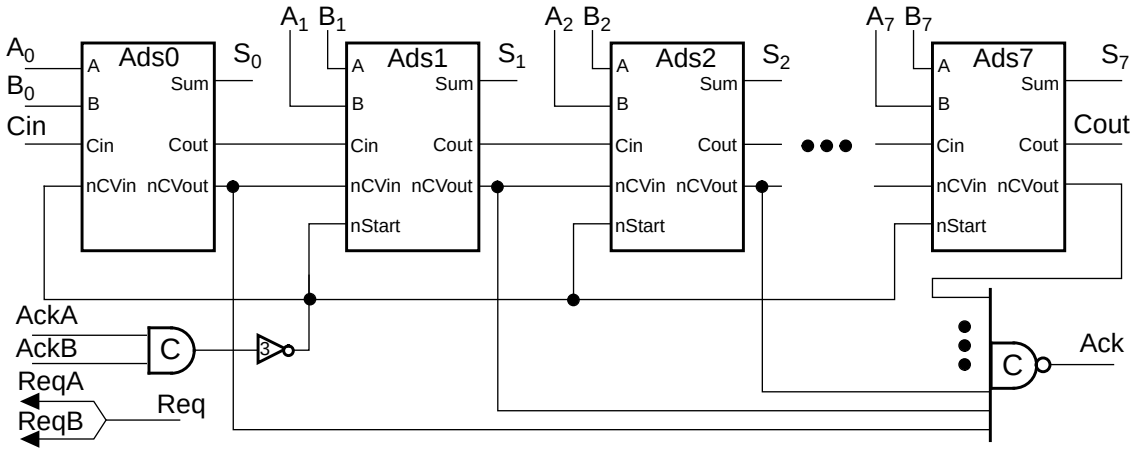

Figure 2 : *Asynchronous 8-bit adder with single-rail data encoding*

Note that a control signal which fires when the *carry-in* signal (*Cin*) is ready can be implemented separately (for instance, using extra signals *AckC* and *ReqC*), or the *Cin* signal can be transmitted together with one of the operands (*A* or *B*) as demonstrated in Figure 2. The choice between these techniques depends on the particular environment in which the adder operates. Hereafter, the *carry-in* signal for the adder is assumed to be transmitted together with one of the operands.

3.1 Testing of the single-rail asynchronous adder

In this section, the single stuck-at fault model including stuck-at input and stuck-at output faults is considered [Russ89]. In order to test the adder shown in Figure 2 a set of test patterns must be applied to its inputs. The test results are observed on the outputs of the adder. It is assumed that the inputs of the asynchronous adder are controllable and its outputs are observable by the environment. The detection of stuck-at faults in the data part of each 1-bit adder in the adder design shown in Figure 2 is trivial since its data inputs and outputs are controllable and observable during the test. Stuck-at faults in the control part of the adder can be divided into three distinct classes:

1. Stuck-at faults which are detectable by logic testing. For instance, stuck-at-0 or stuck-at-1 faults on the *nCVout* outputs are easy to detect since they violate the handshake communication protocol between the adder and its environment.

2. Stuck-at faults which can cause a premature firing on output *Ack*. A stuck-at-1 fault on the output of NAND gate $G1_i$ (i=1, 2, ... , 7) (in Figure 1b) does not change the logic function of the control part of the adder but causes a premature firing on the output of gate $G2_i$ when hs_i=1. This fault may or may not cause the environment to latch wrong data from the outputs of the adder depending on how fast or slow the environment performs.

3. Stuck-at faults which can cause delayed firings on the control output of the adder. These faults do not change the logic function of the control part of the adder but reduce its performance. For instance, a stuck-at-1 fault on input hs_i (i=1, 2, ... , 7) of $G1_i$ causes a delayed response from the adder.

Let us consider the Boolean function of output $nCVout_1$:

$$nCVout_1 = \overline{\overline{nStart \cdot hs_1} \cdot \overline{nStart}} = nStart \cdot hs_1 + nStart = nStart \qquad (1)$$

It is easy to show that $nCVout_i = nStart \cdot hs_i + nStart = nStart$, where i=2, 3, ... , 7.

Thus, the control part of the adder has logic redundancy. Redundant logic elements are necessary to ensure the proper timing function of the control part of the adder. This makes some of its stuck-at faults impossible to detect by logic testing. A fault analysis of the control part of the adder has been carried out with the help of automatic test generation tools designed at Virginia Polytechnic Institute [LeeTR93]. As a result, 27 redundant stuck-at faults have been identified. The fault coverage of the tests generated for detecting faults in the control part of the adder is 53%.

3.2 Design for testability of the single-rail asynchronous adder

In order to make the asynchronous adder shown in Figure 2 testable, the logic redundancy of its control part must be removed during the test. Figure 3 shows the design of a testable 1-bit adder. It operates in two modes: normal operation mode and test mode. The mode of the adder is changed by the Boolean signal *Tst* which is high in test mode and low in normal operation mode. Input *Tst* and the output of the XOR gate (*G3*) are connected to the inputs of the asymmetric C-element. The output of the asymmetric C-element controls the NAND gate (*G2*) which can operate either as an NAND gate or as an inverter depending on the value of its control signal. A CMOS implementation of gate *G2* is illustrated in Figure 4. If the operation mode input *Om* is low the gate acts as a two-input NAND gate. If *Om* is high, input *In2* of the gate is blocked and it operates as an inverter of input *In1*.

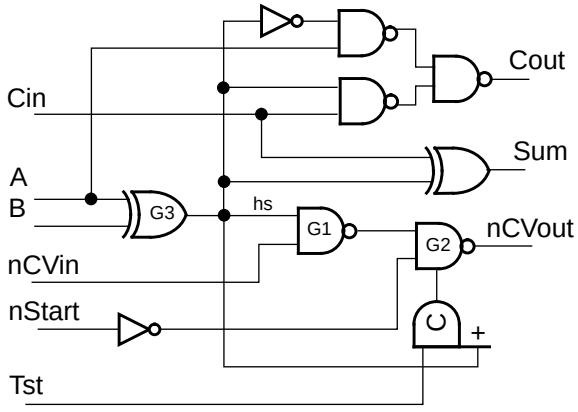

Figure 3 : *Testable asynchronous 1-bit full adder with single-rail data encoding*

In order to set the adder to test mode signal *Tst* and outputs hs_i of XOR gates $G3_i$ (i=1, 2,..., 7) are set to high. In test mode the control part of the adder is identical to an AND gate with output $nCVout_7$ as shown in Figure 5. Stuck-at faults in such a circuit can be detected easily by a standard set of (n+1) tests for an n-input AND gate: one 'all ones' test and n 'running zero' tests. For the circuit illustrated in Figure 5 n=7. Note that signal *nStart* is an active low signal which must be returned to one after the application of each test vector. Moreover, the application of 7 'running zero' tests detects whether or not all gates *G2* of the control part perform as inverters.

To detect stuck-at faults on the $nStart_i \rightarrow Ack$ path (i=1, 2,..., n) in the control part of the i-th 1-bit adder, the following test algorithm can be used:

1. i=1.
2. *Tst*=0; hs_i=0; hs_j=1 (for all $j \neq i$).
3. *Tst*=1. Gate $G2_i$ performs as a NAND gate whereas gates $G2_j$ ($j \neq i$) perform as inverters (see Figures 3 and 4).
4. Signal *nStart* is set to low and then to high.
5. If *Ack* has been changed twice, path $nStart_i \rightarrow Ack$ is fault free, then go to step 6 else go to step 9.
6. i=i+1.
7. If $i > n$ then go to step 8 else go to step 2.
8. The circuit is fault free. Go to step 10.
9. The circuit is faulty. Go to step 10.
10. End.

In summary, the logic testing of the asynchronous adder illustrated in Figure 2, which contains the testable 1-bit adders shown in Figure 3, is difficult due to the test complexity of its control part. For instance, 8 test vectors are required to test the data path of the adder whereas the number of tests required to test its data dependent control part is almost twice this number.

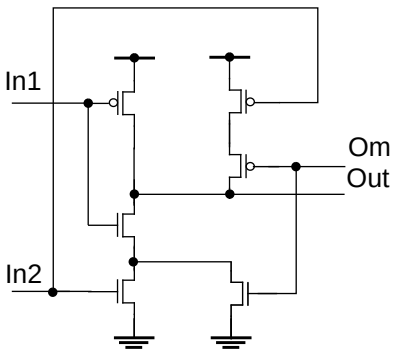

Figure 4 : *Transistor level implementation of the NAND/INV gate*

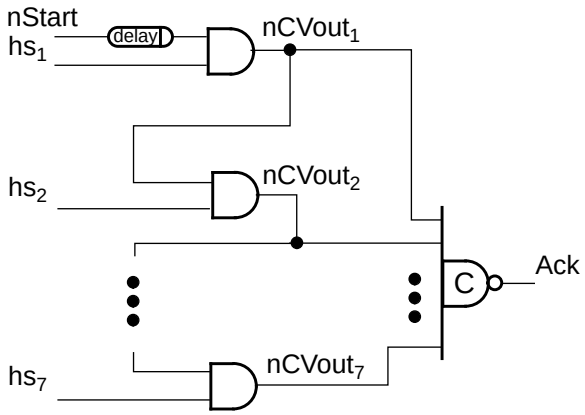

Figure 5 : *Control part of the single-rail 8-bit adder in test mode.*

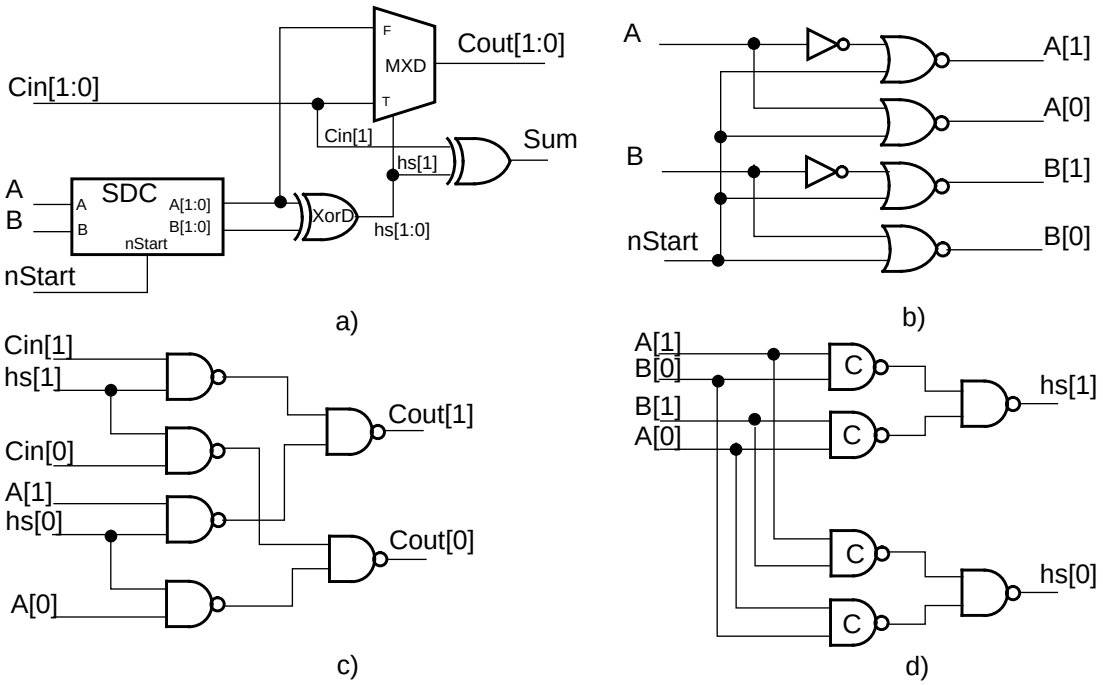

Figure 6 : *Implementations of a) a dual-rail asynchronous 1-bit full adder; b) a conversion element between single-rail and dual-rail data encoding; c) a dual-rail multiplexer; d) a dual-rail XOR gate.*

4. Dual-rail implementation of an asynchronous adder

A dual-rail implementation of an asynchronous 1-bit adder is shown in Figure 6a. It contains a single-rail to dual-rail data conversion block (*SDC*), dual-rail and single-rail XOR gates and a dual-rail multiplexer. The single-rail conversion block modifies the single-rail data from inputs *A* and *B* into the dual-rail data format. A gate level implementation of the conversion block is shown in Figure 6b. When signal *nStart* is high the outputs of the conversion block are kept low. If the data is ready to be transmitted to the adder signal *nStart* is set low and the single-rail data from inputs *A* and *B* is converted into the dual-rail format. Designs of the dual-rail multiplexer and XOR gate are illustrated in Figures 6c and 6d respectively. The use of symmetric C-elements in the design of the XOR gate ensures its delay-insensitivity which, in turn, simplifies its testing. It is easy to show that a stuck-at fault on the inputs of the symmetric C-element is equivalent to the corresponding stuck-at fault on its output. The single-rail result (*Sum*) of addition is produced by XORing signals *Cin[1]* and *hs[1]*.

An example of the dual-rail implementation of an asynchronous 8-bit adder is shown in Figure 7. The inputs and outputs of the adder are single-rail encoded. When a single-rail data is ready on inputs *A*, *B* and *Cin* acknowledge signals *AckA* and *AckB* are set high. As a result, signal *nStart* goes low and addition is started. The output data is ready if the dual-rail carry outputs (*Cout[1]* and *Cout[0]*) of all the 1-bit adders are different

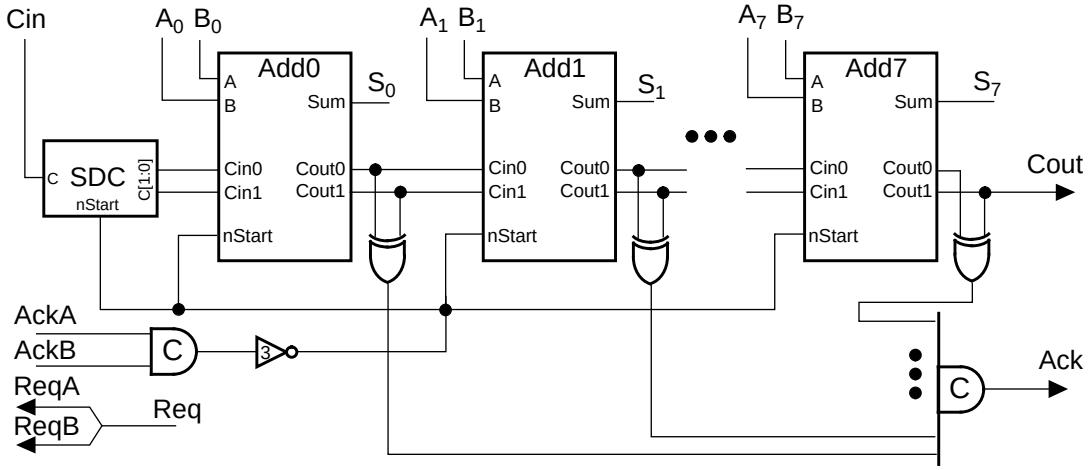

Figure 7 : *Dual-rail implementation of an asynchronous 8-bit adder*

which is indicated by a rising transition on output *Ack*. The actual *carry* output is taken from output $Cout_7[1]$ of the last 1-bit adder. After latching the outputs of the adder the environment returns request signal *Req* to zero. Acknowledge signals go low and signal *nStart* is set high. As a consequence, all the outputs of the adder are set to zero. A fault analysis of the dual-rail implementation of the 8-bit adder shown in Figure 7 was carried out using *SIMIC* design verification tools developed by Genashor Corporation [Sim94, Ashki94]. The results show that the dual-rail adder is fully testable for its stuck-at faults after the application of 29 test vectors during normal operation mode.

5. Hybrid implementation of an asynchronous adder

In this section a hybrid implementation of an asynchronous adder is discussed. The design is called 'hybrid' because, firstly, some of the blocks of the adder perform using dual-rail input data and, secondly, the hybrid adder has a control part similar to that of the single-rail adder. The implementation of a hybrid 1-bit full adder is illustrated in Figure 8. It converts the single-rail data from inputs *A* and *B* using the conversion block which is controlled by the active low *start* signal (*nStart*). Output *hs[1]* of the dual-rail XOR gate (*XorD*) controls the single-rail multiplexer (*MX*) which connects the *carry* input of the adder or output *A[1]* of the conversion block to its output. The control part of the adder uses both outputs of the dual-rail XOR gate to generate a *carry-valid* signal which is active high. When *hs[0]*=1, i.e., inputs *A* and *B* are equal, output *CVout* of the adder goes high indicating the completion of the addition. When input bits *A* and *B* are different *hs[1]*=1 and the symmetric C-element is primed (see Figure 8). The output of the C-element is set to high when input *CVin* goes high. As a result, a rising event is generated on the *CVout* output of the adder.

The design of the hybrid adder is similar to that of the single-rail adder shown in Figure 2. When the data is ready on the inputs of the adder signal *nStart* is set to zero and the input data is converted to the dual-rail format. The completion of the addition is indicated by a rising event on output *Ack* of the multi-input symmetric C-element. When the *nStart* signal is returned to one the data and control outputs of the adder are reset. In order to return the *carry-valid* output of the hybrid asynchronous adder to zero all the C-elements in the control paths of the 1-bit adders must be reset (see Figure 8). If all $hs_i[1]$ (i=0, 1, ... , 7) were set to high the C-elements in the control part of the adder are returned to zero sequentially starting from the first 1-bit adder. This is the worst case performance of the hybrid adder. A fault analysis of the hybrid 8-bit adder shows that the detection of its stuck-at faults requires the application of 33 test vectors during normal operation mode.

6. A case study of an asynchronous comparator

In this section the design of an asynchronous 8-bit comparator is considered. The comparator is used as a comparison block for a pair of 8-bit input vectors in an asynchronous block sorter [Berk93, Farn95]. Figure 9 illustrates the design of the asynchronous comparator. It contains an asynchronous 7-bit adder to perform subtraction of the data from inputs *A* and *B* (*A=A[7:1], B=B[7:1]*) as follows

$$\overline{Cout} \Leftarrow A + \bar{B} + Cin \quad . \tag{2}$$

The *carry* input of the adder is generated by ORing the least significant bits of 8-bit operands *A* and $\bar{B}$. If *Cout* is low then *A* is greater or equal to *B* otherwise *A* is less than *B*. Note that the 7-bit adder of the comparator does

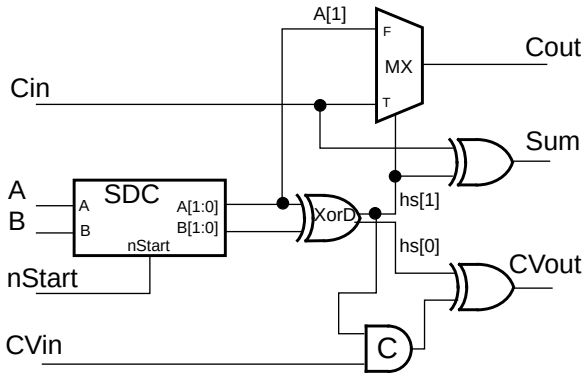

Figure 8 : *Hybrid implementation of an asynchronous 1-bit full adder*

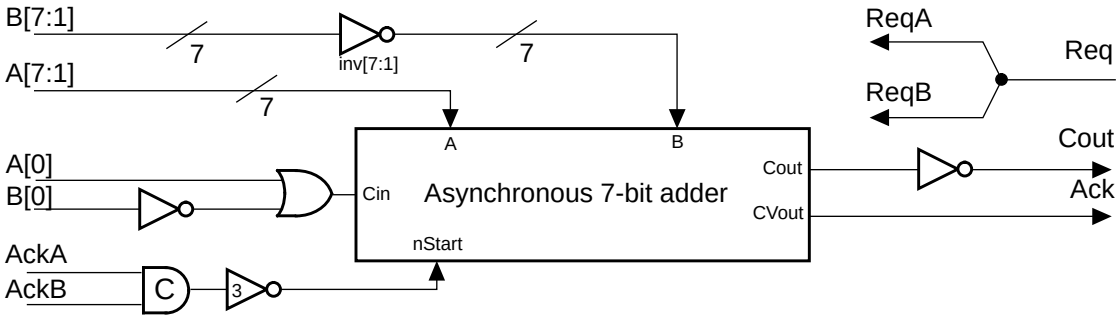

Figure 9 : *Asynchronous 8-bit comparator*

not produce the results of subtraction. The comparator shown in Figure 9 performs in a similar way as was described in previous sections for a multi-bit asynchronous adder.

The 8-bit comparator was designed and implemented using a 1μm double metal CMOS process with the help of *Cadence* CAD tools. Several versions of the comparator with different implementations of its 7-bit adder have been simulated using *SIMIC* verification tools. Simulation results are shown in Table 3. The single-rail adder without testability features is taken as a base for estimating the relative characteristics of the other adder designs since it requires the minimal silicon area and demonstrates the highest performance. The performance of each version of the comparator was calculated in normal operation mode by applying an identical set of 128 tests generated by a pseudo-random pattern generator.

According to the simulation results shown in Table 3 the comparator with the dual-rail adder demonstrates the largest area overhead (138%) compared to the comparator which uses the single-rail adder without testability features. The comparator with the hybrid adder shows the lowest performance which is close to that of the dual-rail comparator. The comparator with the testable single-rail adder demonstrates the minimal area overhead and performance degradation but requires a special test mode. The use of the hybrid adder in the comparator brings a compromise between area overhead, performance degradation and testability providing for the detection of all its stuck-at faults in normal operation mode. However, it is 30% slower and its implementation is almost twice as large as the comparator which uses the single-rail adder without testability features.

Table 3: *Simulation results of the comparator using different adder designs*

Adder design of the comparator		Area, $10^{-2} \times mm^2$	Performance, ns/test	AO[a], %	PD[b], %	No. extra pins
Single-rail adder	untestable	3.85	24.15	-	-	-
	testable	4.60	24.55	19	2	1
Dual-rail adder		9.17	30.48	138	26	0
Hybrid adder		7.40	31.50	92	30	0

a. AO is the area overhead
b. PD is the performance degradation

7. Conclusions

Different designs of an asynchronous adder and their testability properties have been investigated in this paper. The single-rail implementation of an asynchronous adder is least complex in terms of number of gates, and is fast, but it demonstrates low stuck-at fault testability due to the logic redundancy in its control part. The logic testing of a single-rail asynchronous adder requires a special test mode to be implemented in order to remove its logic redundancy. As a consequence, stuck-at faults which have not been detected in normal operation mode can be identified in test mode. The dual-rail and hybrid implementations of the asynchronous adder are fully testable for stuck-at faults in normal operation mode but they require more area and exhibit lower performance. The dual-rail implementation of an asynchronous adder is faster than the hybrid adder but requires more silicon area. The dual-rail and hybrid adders can be used in asynchronous VLSI designs where performance and area overhead are not critical but testability in normal operation mode is important. The testable single-rail version

of the adder can be used in asynchronous VLSI circuits which can be tested in both normal operation mode and test mode.

References

[Ashki94] A. Ashkinazy, D. Edwards, C. Farnsworth, G. Gendel, S. Sikand, “Tools for validating asynchronous digital circuits”, Proc. Int. Symp. on Advanced Research in Asynchronous Circuits and Systems (Async94), Nov. 1994, pp. 12-21.

[Berk93] Kees van Berkel, “Handshake circuits. An asynchronous architecture for VLSI programming,” Int. Series on Parallel Computation 5, Cambridge University Press, 1993.

[Birt95] G. Birtwistle, A. Davis (Eds), “Asynchronous digital circuit design”, Springer, 1995.

[Brzo95] J. A. Brzozowski, C-J. H. Seger, “Asynchronous circuits”, Springer-Verlag New York, Inc., 1995.

[Dav90] I. David, R. Ginosar, M. Yoeli, “Self-timed is self-diagnostic", TR-UT-84112, Department of Computer Science, University of Utah, Salt Lake City, UT, USA, 1990.

[Farn95] C. Farnsworth, D. A. Edwards, Jianwei Liu, S. S. Sikand, “A hybrid asynchronous system design environment”, Proc. 2nd Working Conf. on Asynchronous Design Methodologies, South Bank University, May 30-31, 1995, pp. 91-98.

[Furb94] S. B. Furber, P. Day, J. D. Garside, N. C. Paver, J. V. Woods, “AMULET1: A micropipelined ARM”, Proc. IEEE Computer Conf., March 1994.

[Gars93] J. D. Garside, “A CMOS VLSI implementation of an asynchronous ALU”, IFIP WG 10.5 Working Conference on Asynchronous Design Methodologies, Editors S. Furber, M. Edwards, Manchester, 1993.

[Hauck95] S. Hauck, “Asynchronous design methodologies: An overview”, Proc. IEEE, Vol. 83, No. 1, Jan. 1995, pp. 69-93.

[Haz92] P. Hazewindus, “Testing delay-insensitive circuits”, Ph.D. thesis, Caltech-CS-TR-92-14, California Institute of Technology, 1992.

[Hulg94] H. Hulgaard, S. M. Burns, G. Borriello, “Testing asynchronous circuits: A survey”, TR-FR-35, Department of Computer Science, University of Washington, Seattle, WA, USA, 1994.

[Khoc94] A. Khoche, E. Brunvand, “Testing micropipelines”, Proc. Int. Symposium on Advanced Research in Asynchronous Circuits and Systems (Async94), Utah, Nov. 1994, pp. 239-246.

[Lav93] L. Lavagno, A. Sangiovanni-Vincentelli, “Algorithms for synthesis and testing of asynchronous circuits”, Kluwer Academic Publishers, 1993.

[LeeTR93] H. K. Lee and D. S. Ha, “On the generation of test patterns for combinational circuits”, Technical Report No. 12_93, Dept. of Electrical Eng., Virginia Polytechnic Institute and State University, 1993.

[MClus86] E. J. McCluskey, “Logic design principles: with emphasis on testable semicustom circuits”, Prentice/Hall International Inc., 1986.

[Pet95] O. A. Petlin, S. B. Furber, “Scan testing of micropipelines”, Proc. 13th IEEE VLSI Test Symposium, Princeton, New Jersey, USA, May 1995, pp. 296-301.

[Ron93] M. Roncken, R. Saeijs, “Linear test times for delay-insensitive circuits: a compilation strategy”, IFIP WG 10.5 Working Conference on Asynchronous Design Methodologies, Editors S. Furber, M. Edwards, Manchester, 1993, pp. 13-27.

[Ron94] M. Roncken, “Partial scan test for asynchronous circuits illustrated on a DCC error corrector”, Proc. Int. Symp. on Advanced Research in Asynchronous Circuits and Systems (Async94), Nov. 1994, pp. 247-256.

[Russ89] G. Russell, I. L. Sayers, “Advanced simulation and test methodologies for VLSI design”, Van Nostrand Reinhold (International), 1989.

[Sim94] “SIMIC: design verification tool”, User's Guide, Genashor Corporation, N.J., 1994.

[Suth89] I. E. Sutherland, “Micropipelines”, Communications of the ACM, Vol. 32, no. 6, pp. 720-738, June 1989.

[Wey93] Chin-Long Wey, Ming-Der Shieh, D. Fisher, “ASCLScan: a scan design for asynchronous sequential logic circuits”, Proc. IEEE Int. Conf. on Computer-Aided Design, 1993, pp. 159-162.