Proceedings of the Eighteenth UK Asynchronous Forum

School of Electrical, Electronic and Computer Engineering
University of Newcastle
Newcastle upon Tyne, NE1 7RU, U.K.
4-5 September, 2006

18th UK Asynchronous Forum. Preliminary Programme

Day 1 Monday 4th September 2006

13.20 Registration

13.40 Tutorial 1: Demystifying Data-Driven and Pausible Clocking Schemes
Robert Mullins and Simon Moore, University of Cambridge
14.20 Modeling and Performance Analysis of GALS Architectures
S. Dasgupta, A. Yakovlev, University of Newcastle
14.40 Delay Insensitive Chip-to-Chip Interconnect Using
Incomplete 2-of-7 NRZ Data Encoding
Jian Wu and Steve Furber, University of Manchester
15.00 An asynchronous spiking neural network which can learn temporal sequences
Joy Bose, S.B Furber, M. Cumpstey, University of Manchester
15.20 Error Checking and Resetting Mechanisms for Asynchronous Interconnect
Yebin Shi and Steve Furber, University of Manchester

15.40 Coffee

16.00 On-chip Phase Regeneration Circuits
C. D'Alessandro, A. Bystrov, A. Yakovlev, University of Newcastle
16.20 Fault Tolerant Techniques to Minimise the Impact of Crosstalk on Phase
Encoding Communication Channels
Basel Halak, Alex Yakovlev, University of Newcastle
16.40 C-element Latch Scheme with Improved Fault Tolerance
K. T. Gardiner and A. Yakovlev, University of Newcastle
17.00 An Information Redundant Asynchronous Concurrent Error Detecting ALU
M.J. Marshall, G. Russell, University of Newcastle
17.20 Presentation by attendees
17.40 Business Meeting
19.30 Forum Dinner

Day 2 Tuesday 5th September 2006

09.00 Tutorial 2: Unfolding models of asynchronous systems:
applications to analysis and synthesis,
Victor Khomenko, University of Newcastle
09.40 CSC-Aware STG-Decomposition
Mark Schaefer, University of Augsburg, Germany
10.00 Validation of an Asynchronous Synthesis Back-End
Nitin Gupta, Doug Edwards, University of Manchester
10.20 Blame Passing for Analysis and Optimisation
Charlie Brej, University of Manchester
10.40 Completion Detection Optimisation based on Relative Timing
A. Mokhov, D. Sokolov, A. Yakovlev, University of Newcastle

11.00 Coffee

11.20 Comparative Analysis of Stuck-at Test Generation
for Asynchronous Speed Independent Circuits
D.P. Vasudevan, A. Efthymiou, University of Edinburgh
11.40 On-FPGA Communication: An Opportunity for GALS?
Terrence S.T. Mak, Peter Y.K. Cheung, Pete Sedcole,
Imperial College London
12.00 Metastability in FPGA Devices
N.Minas, D.J.Kinniment, G.Russell, A.Yakovlev, University of Newcastle
12.20 Asynchronous Timing in the Survivor Memory Unit of a Viterbi Decoder
Wei Shao and Linda Brackenbury, University of Manchester

13.00 Close

Demystifying Data-Driven and Pausible Clocking Schemes

Robert Mullins and Simon Moore
Computer Laboratory, University of Cambridge
Robert.Mullins@cl.cam.ac.uk

Abstract

VLSI systems are often constructed from a multitude of independently clocked synchronous IP blocks. Unfortunately, while a synchronous design style may produce efficient block level implementations it does little to support their composition. The addition of asynchronous interfaces to each synchronous block is one way to simplify and strengthen their integration. Asynchronous interfaces allow blocks to be composed without the need to consider synchronisation failure rates, permit data-driven operation and provide greater freedom when designing on-chip buses and networks. This paper surveys the significant body of published work in this area. We highlight similarities between schemes that are often concealed by differences in specification or circuit style. We also present a number of new local clock implementations and provide solutions to mitigate the impact of clock-tree insertion delays. The ultimate goal of this work is to permit multi-clock synchronous systems to be composed simply, robustly and efficiently.

1. Introduction

Current architectural trends are driven by the observation that simply creating larger and more complex monolithic IP blocks is rarely the best use of growing transistor budgets. A more flexible and scalable approach is to create a network of simpler IP blocks. Technology scaling is subsequently exploited by adding additional blocks rather than increasing the complexity of each individual block. This communication-centric methodology aims to exploit a block size that produces an efficient circuit-level implementation and isolates the designer from the need to consider multi-cycle interconnect delays. Furthermore, by restricting each blocks complexity we aim to avoid the pitfalls of employing ever more complex and power hungry techniques to obtain ever decreasing performance gains. The IP network also provides the flexibility necessary to operate in a fault-tolerant manner, manage power and thermal goals and produce the multi-use platforms dictated by rising design and NRE costs.

Much of the complexity in such a system is shifted from the design of individual IP blocks, concentrating on computation, to their interconnection, management and scheduling. In this environment the simplifying assumptions that a synchronous design style traditionally offers are less evident. In contrast to simply optimising combinational logic within a single clock domain, the process of integration requires us to consider a physically distributed system, span clock domains and handle multi-cycle interconnects. System timing is often further complicated by the application of voltage and frequency scaling and static power reduction techniques such as power gating. The challenges posed by the broad range of timing and communication requirements are perhaps more naturally tackled by adopting an event-driven control paradigm.

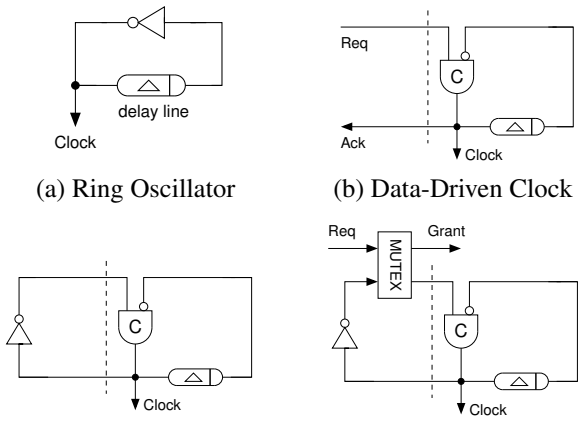
The techniques presented in this paper are designed to allow independently clocked IP blocks to be interconnected asynchronously, without the complexity of imposing additional clocks and synchronisers during the integration process. The schemes could also be used to construct a data-driven IP network in order to minimise synchronisation overheads, latency and superfluous switching activity. The use of asynchronous techniques also provides a robust framework for power reduction schemes, such as the voltage scaling of on-chip interconnects and IP blocks. The systems described here are often characterised as Globally Asynchronous Locally Synchronous (GALS).

2. Local Clock Generators

A ring oscillator constructed from a tunable delay line and an inverter (Figure 1(a)) may be used as the basis for a flexible on-chip clock generator. The frequency of such a clock generator may be periodically calibrated to an off-chip reference clock, as demonstrated in [15]. In a GALS system, each synchronous block is clocked from a local clock generator of this type.

When a free-running oscillator is employed, the datapath clearly plays no role in the generation of the clock. However, by making small modifications to this basic oscillator circuit we will demonstrate how interesting and useful interactions with the datapath may be developed.

The circuit illustrated in Figure 1(b) is the starting point for many of the published schemes and those presented here. In this circuit, the ring oscillator has been extended to require both an event on the req input and on the output of the delay-line before the next clock edge is generated. This is enforced by the use of a C-element that operates as an



(a) Ring Oscillator (b) Data-Driven Clock

(c) Ring Oscillator (II) (d) Pausible Clock

Figure 1. Pausible and Data-Driven Local Clocks

AND-gate for events [24]. By enforcing a strict four-phase handshake on the interface we are guaranteed a minimum clock period determined by the delay-line. In addition, we now have the opportunity to stretch the clock period by delaying the completion of the handshake. The circuit may also be viewed as a single stage of a micropipeline with the output handshake ports connected together [25]. We call this circuit a *data-driven clock*. As is, this clock simply generates a single clock cycle in response to each incoming input request.

If we complete the handshake by simply inserting an inverter between the ack and req ports, as illustrated in Figure 1(c), we produce a simple ring oscillator. A well documented approach to producing a pausible clock is to interrupt this cycle with a mutual-exclusion element (MUTEX) [24] (see also Appendix A.). This produces a clock that will normally oscillate unless we interrupt it by holding req high. This is in contrast to the data-driven clock where a complete handshake must take place during every clock cycle. This *pausible clock* circuit is illustrated in Figure 1(d).

The majority of the clock generator circuits described are based on either a data-driven or pausible clock template. The ability to stretch or delay the clock may be exploited in a number of ways. The original purpose of clock pausing was to create additional time for metastability to resolve, *e.g* to permit the safe transfer of data between different clock domains. An additional reason may be to create a data-driven clock that produces clock edges only when data is available for processing. This type of data-driven operation mimics a high-speed global clock without the associated synchronisation overheads and superfluous switching activity.

3. Input Ports

We distinguish between three different behaviours for handling locally-clocked IP blocks with multiple inputs. In each case we assume that once an input request is made it remains asserted until it is serviced.

- **Arbitrated Inputs:** At most one input request may be serviced per clock cycle. This requires the inputs to arbitrate for access to the IP block.
- **Sampled Inputs:** An event is used to trigger a sampling of all input ports. This sampling determines which inputs have data that is ready to be admitted on the next clock cycle. The sampling event is either a (delayed) clock-edge or the arrival of an input request. The precise choice of sampling event depends on the type of local clock generator.
- **Synchronised Inputs:** A request to admit data is only generated when valid data is present on all inputs.

Each of the behaviours described could be supported by a subset of a block's inputs. The synchronised input behaviour could also be trivially extended to wait for some subset of the blocks inputs to become ready.

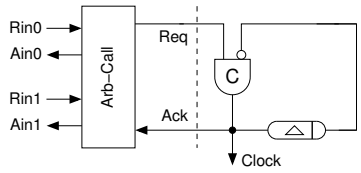
Scheduled communications could also be supported by the clock generators presented here, requiring data to be read from a specified input (or written to a specified output) port on a particular clock cycle. The design of deterministic GALS systems supported by a mechanism for communicating at regular intervals (or *recycle periods*) is described in [9].

4. Data-Driven Clocks

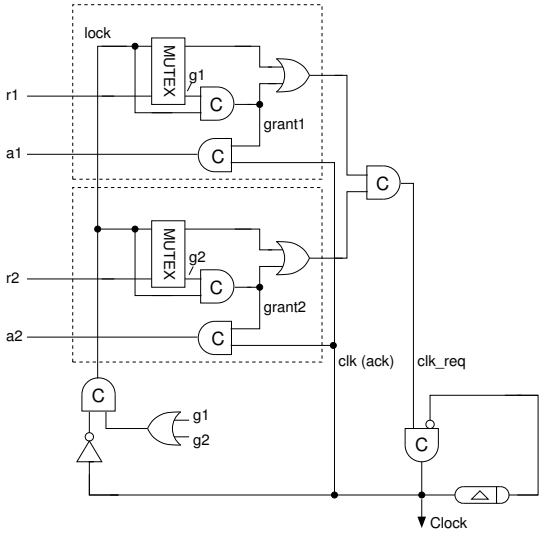
The data-driven clock circuit (Figure 1(b)) may be extended to support each of the input behaviours described in Section 3. Each of these circuits is illustrated in Figure 2. While arbitrated and synchronised inputs are trivial to implement, the sampled inputs scenario requires some explanation.

A data-driven clock with sampled inputs may be useful in an environment where an IP block may make forward progress regardless of the number of inputs that are ready. One example of such a block may be a locally-clocked router in an on-chip network. In this scenario, the detection of an input port request forces a decision to be made on whether to admit data from each input port on the next clock cycle. In the case of an on-chip router additional clock cycles would have to be generated to guarantee packets buffered within the router made forward progress when no new input data was forthcoming. The way in which these additional cycles could be generated is discussed in Section 4.1.

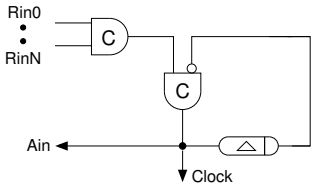
The circuit illustrated in Figure 2(b) supports a data-driven sampled-input behaviour using a circuit that takes inspiration from the static priority arbiter introduced in [3]. The circuit is quiescent until a request is made by one of the input ports. A $lock$ request is then asserted to force each MUTEX to grant either the lock or input port request. Only after it has been determined from which input ports data will be admitted in the next clock cycle, will a new rising clock edge be generated. To improve



(a) Data-Driven Clock with Arbitrated Inputs



(b) Data-Driven Clock with Sampled Inputs



(c) Data-Driven Clock with Synchronised Inputs

Figure 2. Data-Driven Clock Generators

performance the $lock$ signal is prevented from asserting before the falling clock edge. This prevents inputs from being 'locked out' early in the clock cycle.

4.1. Pipelines and Flushing

In contrast to a pausible clock generator where the clock is normally running, a data-driven clock only produces clock edges in response to input data. In some cases the architecture of the IP block may required additional clock cycles to be generated to complete an operation, *e.g.* in the case of a pipelined IP block or one that buffers data. These additional clock cycles may be generated in a variety of ways:

- **Eager Flushing** ensures a further clock cycle is generated without delay if the IP block has useful work to complete. In the case of a pipelined block a counter may be used to request these additional cycles. The counter is initialised to the pipeline depth after each successful input request. The counter subsequently requests clock cycles, decrementing its value on each cycle, until it reaches zero. If a sampled data-driven approach is used the counter would be responsible for asserting a $lock$ request in the case when no valid input data was present. In some cases it may be preferable to replace the counter with logic that examines datapath signals directly to determine if useful work is outstanding.
- **Time-Out Flush:** A slightly different approach is to wait for some predetermined time before initialising the counter. Only when the time-out occurs are the additional clock cycles generated. This approach potentially reduces the total number of clock cycles generated by providing an opportunity for new data to push previous values through the pipeline.
- **Uninterrupted Flush:** Depending on the requirements of the IP block it may be useful to implement an uninterrupted pipeline flush mechanism. In this case the pipeline is flushed before any new input data is allowed to enter the IP block.
- **Pull-Driven Flush:** It is suggested in [12] that it may be possible to switch from a data-driven (push) to pull-driven mode when no new input requests are forthcoming. Although no details of such an implementation are explored.

4.2. Related Work: Active Clock Handshake Interfaces

The handshake interface on the data-driven clock illustrated in Figure 1(b) is *passive*, *i.e.* it can only respond to an external request. By swapping the req and ack ports we can create a data-driven clock with an *active* handshake port. The clock now acts as a request that the environment must acknowledge. In order to be able to generate a clock the incoming req signal must be inverted (or held high to indicate that the environment is ready). Such an approach is explored in [11] to enable communication between a fast processor and slow memory.

An active clock handshake interface can still support the full range of input behaviours previously discussed. In addition, it is perhaps more natural in some cases to think of some communications as **conditional** [11], rather than **scheduled**. Arbitrated access to a single passive resource from multiple clock modules may now also need to be considered.

4.3. Related Work: Request-Driven Clocking

A data-driven clocking mechanism with a time-out based flushing mechanism is presented in [12, 13]. The term *request-driven* clock is used to describe their scheme.

4.4. Related Work: Clock Stretching

Clock stretching is a form of data-driven clocking where the handshake interface is replaced with a single *stretch* control signal that is asserted synchronously. The relationship with the basic data-driven clock circuit may

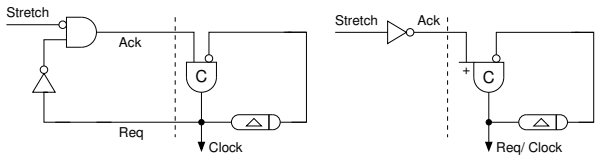
be highlighted by redrawing the clock stretching circuit as illustrated in Figure 3(a). The clock handshake port is now an active one as discussed in Section 4.2. Figure 3(b) simply removes the AND-gate by converting the C-element into an asymmetric gate. Both these circuits are equivalent to Bormann's stretchable clock generator as illustrated in [1, 2].

A clock stretching feature to permit asynchronous communication between synchronous systems is discussed by Seitz in [22] (Ch. 7, Sec. 8.4). Seitz indicates that this approach has been used in various proprietary designs since 1968.

Pěchoucěk observes that the ability to stretch the clock is the only solution which guarantees value-safe communication between independently clocked modules. He describes a system for extending the clock period of a system until metastability has resolved [20]. Pěchoucěk also outlines a data-driven clocking scheme where the generation of a fixed number of clock cycles is triggered by the arrival of input data. This type of clocking scheme was more recently employed to create an on-chip clock generator for a DSP [18] and a data-driven GALS clocking scheme for a low-power reconfigurable processor [28]. There are no synchronisation issues with such a scheme as the clock is always quiescent when the initial asynchronous data input arrives. This could be achieved in a robust manner by ensuring the ack signal from the data-driven clock is not deasserted until the processing of the data has completed.

Chapiro investigates the use of stretchable clock generators and introduces the term Globally-Asynchronous Locally-Synchronous (GALS) to describe synchronous system composed using such interfaces [4].

Lim describes the use of a stoppable clock generator in [14], again a single input to the clock generator is used to delay the generation of the next rising clock edge until data is available. Lim also describes the use of a MUTEX to provide an arbitrated input behaviour.



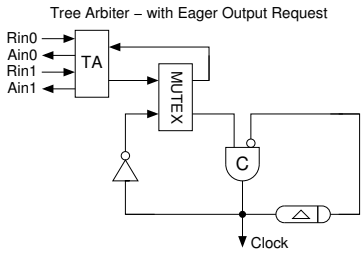
(a) (b)

Figure 3. Equivalent Stretchable Clock Circuits

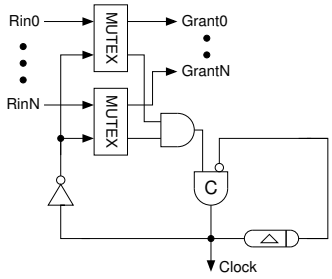
5. Pausible Clocks

Pausible clock circuits may be constructed using the simple template provided in Figure 1(d) as a starting point. Figure 4 illustrates how a pausible clock can support each of our input behaviours.

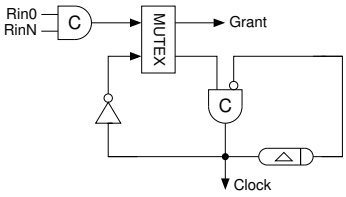
The tree arbiter shown in Figure 4 allows an input request to be initiated while it is determined which input port should be granted access [10]. If necessary, the eager request generation could be omitted. A tree-arbiter implementation is provided in Appendix A.



(a) Pausible Clock with Arbitrated Inputs



(b) Pausible Clock with Sampled Inputs



(c) Pausible Clock with Synchronised Inputs

Figure 4. Pausible Clock Generators

Previous work has illustrated how pausible clock generators may be used to facilitate point-to-point value-safe communication between independently clocked IP blocks [16]. Figure 5 illustrates the receiver side of such a communication (note, the handshake protocol used here is a two-phase one). Data is latched safely in the first input register while it is guaranteed no rising clock edge can take place. After this operation is complete and the input request is removed, a rising clock edge is generated that safely transfers the input data into the synchronous domain. It is our belief that all existing high-throughput pausible clock schemes latch the input data in this manner.

An alternative is to replace the MUTEX element with an arbitrated call (see Figure 6). A new rising clock edge may now be requested by either the inverted clock or new input data. If the input port is granted we can safely enable the input register and allow new data to enter the synchronous block. This approach reduces the chance that the clock period is extended by removing the need to block the generation of the next rising clock edge while the data is latched and the handshake is completed. An

implementation of an arbitrated-call element is provided in Appendix A.

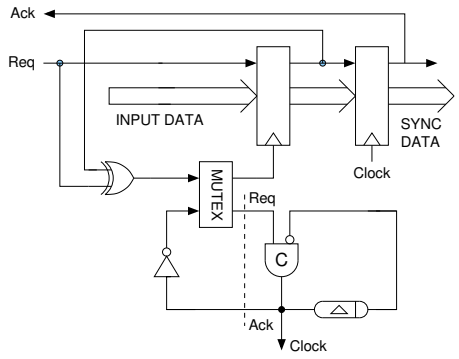


Figure 5. Consumer side interface driven by a pausible clock generator

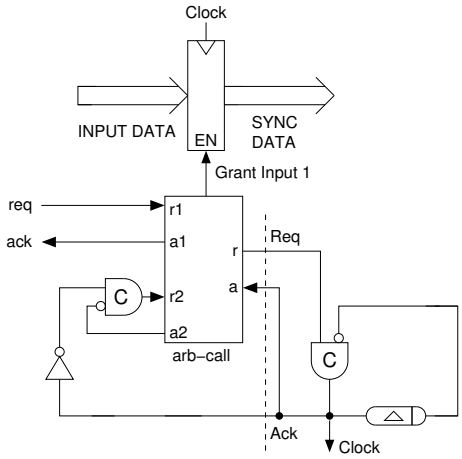


Figure 6. An alternative arbitrated-call based pausible clock generator

5.1. Related Work: Lim's Operation Module

Lim [14] describes an extension to a data-driven clocking scheme where the synchronous block is designed to make forward progress without requiring a constant stream of input data. The clock is now normally enabled to run by allowing the module itself to make requests for further clock edges. The scheme is illustrated in Figure 7. If the *check input port* signal is low, the *start clock* input to the clock generator will remain high allowing the clock to oscillate. Input data may only be admitted when the *check input port* signal is asserted. Lim suggests this may be done periodically or every cycle.

To enable input data to be admitted on every clock cycle the clock itself could be used as the *check input port* signal. This produces a circuit close to our pausible clock template. The idea of generating a new rising clock

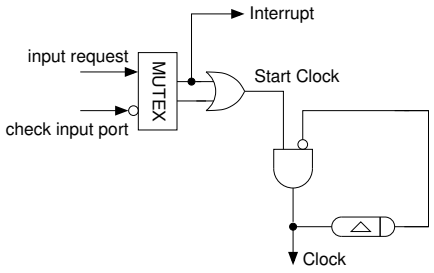


Figure 7. Lim's Operation Module

edge independently of which MUTEX input is granted is also exploited in our arbitrated-call based pausible clock generator. In general, the approach may be classified as a pausible clock generator with a **scheduled sampling** of input ports.

5.2. Related Work: Asynchronous Synchroniser Elements, Q-Elements and DFLOPs

The sampling or synchronising mechanism of the pausible clock may be applied at the level of a single register. The Amulet3 interrupt synchroniser is one example of this approach [7]. The synchroniser circuit is reproduced in Figure 8. This circuit is one component of the synchronous consumer circuit shown in Figure 5. It may be useful to think of this circuit as one that augments a register with an asynchronous (write) handshake interface. The handshake interface is required as the time required to synchronise an input is unknown (and unbounded).

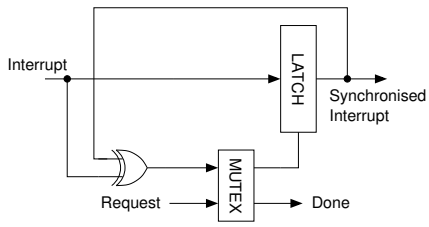


Figure 8. Amulet3 Interrupt Synchroniser (reproduced from [7])

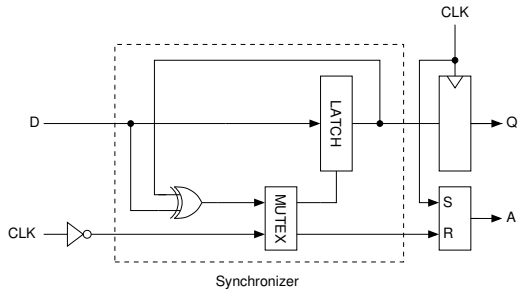
Rosenberger *et al* developed a technique to build delay-insensitive modules by exploiting input registers with asynchronous handshake interfaces [21]. These Q-modules operate in two distinct phases initiated by falling and rising clock edges. On a falling clock edge each input register (Q-element) samples its input and records this value – without updating its output. The subsequent rising clock edge is now delayed until each input register has acknowledged the completion of this operation. A rising clock edge is then generated prompting each Q-element to copy the synchronised input value to its output. Finally, when this 'update output' operation has

been acknowledged by all registers, time is scheduled for the computation itself to take place.

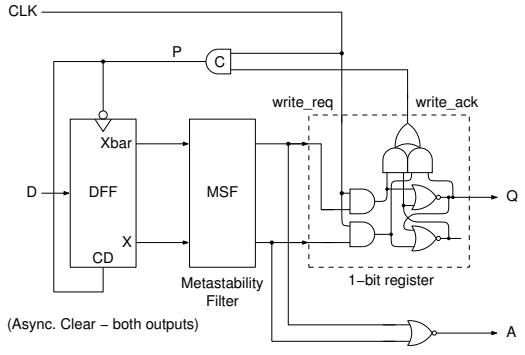
We may construct a Q-element from the synchroniser circuit shown in Figure 8. We simply need to add an additional output register that is updated on receipt of a rising clock edge. The acknowledge output (A) is implemented with a SR-latch. Note, the Q-module specification requires this acknowledge to make a transition from high to low to indicate the current input value has been synchronised or read. The acknowledge is reset when the output is updated by a rising clock edge. The circuit is illustrated in Figure 9(a).

The Q-element described by Rosenberger *et al* is a more direct implementation of the required behaviour. A static-logic reimplementation of this is provided in Figure 9(b). This circuit is in the style of the original Q-element, although the emphasis here is on the major components of the circuit and their interfaces. Other possible implementations and optimisations are explored in [26].

The derivation of a synchroniser circuit is also undertaken in [19]. Here the circuit is designed to also cope with the removal of an input before it has been sampled.



(a) A synchroniser based Q-element



(b) A static-logic reimplementation of a Q-element

Figure 9. Possible Q-element implementations

5.3. Related Work: Pausible Clocks

The pausible clock circuits described generate a clock even when no input data is present. For some applications it may be desirable for the clock generator to enter a sleep state with the clock stopped until new data arrives. A sleep mechanism of this type is explored in [16]. Here the sleep request is asserted synchronously by the clocked module.

The *pausible clock control* (PCC) circuit implemented by Yun and Dooply [27] closely resembles the pausible clock circuit shown in Figure 4(a) – a pausible clock with arbitrated inputs.

An overview of the many different GALS test chips produced at ETH Zurich using the pausible clock approach is described in [8]

6. Output Ports

In this section we briefly discuss a number of different output port behaviours.

- **Scheduled:** This type of port is used when an output operation must be completed on a particular clock cycle, *e.g.* when the output of the synchronous block is not registered. This port type will stall the generation of the next clock cycle until the data is successfully consumed.
- **Registered:** The addition of an output register permits the output operation and the next computation to take place concurrently. A registered output port only need stall the clock when the output becomes blocked for an extended period. At this point any further clock edges must be prevented to ensure data in the output port register is not overwritten.
- **Polled:** This type of port polls the output to determine when it is safe to send data. The clock is interrupted only in cases where additional time is required to resolve any metastability occurring due to the sampling of the asynchronous output port ready signal. The synchronous block is responsible for coping with blocked output ports.

The implementation of each of these output port behaviours requires no new techniques. Each may be based upon the existing input port and clock generator templates.

An example of how previously discussed approaches may be combined to produce specific input and output port behaviours is illustrated in Figure 10. This clock generator supports both a sampled input port (based upon a pausible clock) and a registered output port implemented using a stretchable clock. In general, each new port of a different style will require its own handshake port on the clock generator. A clock generator with N handshake ports is shown in Figure 2(c). It should be noted that combining different port types may alter the behaviour of individual ports or prevent some ports accepting any new data. Special care must be taken when combining both ports based on data-driven (stretchable) clocking templates and those constructed from the pausible clock template.

The circuit in Figure 10 is a simple example that could be improved in a number of ways. One extension would be to allow the output port to be clocked as soon as the computation is complete, even in the case when the input

port is pausing the clock. An in-depth discussion of such optimisations and each of the possible output port circuits is beyond the scope of this paper.

7. The Impact of Clock Tree Insertion Delays

In all the previous examples it has been assumed that the delay imposed by the clock tree is insignificant. In reality this *clock insertion delay* may vary from a few gate delays to many clock cycles. The precise delay will depend on the number of sequential elements in the synchronous block and its physical size. The design of a traditional synchronous system is mostly unaffected by this delay as there is no reason to distinguish between different clock edges produced by the clock source. If clock gating or the techniques described here are adopted, an association is made between particular clock edges and datapath operations. This forces us to consider the clock tree insertion delay in any analysis of the circuit.

This section outlines how the impact of clock tree insertion delays may be minimised. The analysis is presented for a simple data-driven clock, but applies equally to any local clock wrapper.

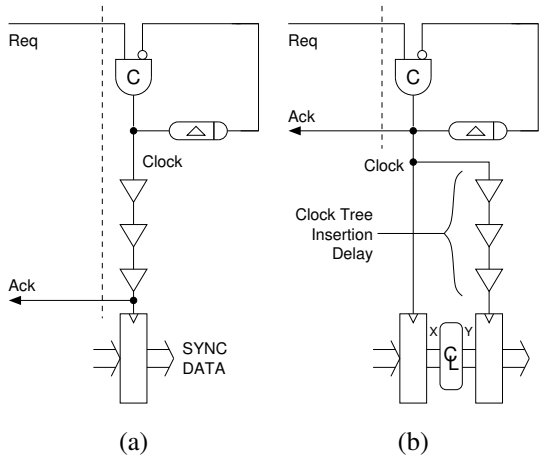


Figure 11. Accounting for the clock insertion delay when generating a data-driven clock

The circuit illustrated in Figure 11(a), shows a simple data-driven clock generator clocking a single input register. The clock tree insertion delay is shown as a chain of buffers. To guarantee that input data is correctly latched we must ensure an input request is only acknowledged after the input register has been clocked. This ack signal must therefore be delayed by at least the time taken to propagate a clock edge through the clock tree [6].

The concern now is that if the clock insertion delay is greater than half the clock period the clock will always be extended. In this case the clock period is increased from twice the delay of the delay-line to at least twice the clock tree insertion delay.

If we are certain the clock tree insertion delay is less than one clock cycle we can simply add an additional register to buffer the input data. This scheme is illustrated in 11(b). The additional latch holds the input data until it can be clocked into the synchronous module, allowing the handshake to complete quickly. If the clock insertion delay was to exceed one clock cycle this scheme would fail as new data would be latched in the first input register before the previous data item had been copied to the second. The constraint that there is at most one rising clock edge in the clock tree applies to many of the published aperiodic clocking schemes.

The time between the clocking of the first input register and the second is equal to the clock tree insertion delay. As this delay is fixed we may insert combinational logic between the registers in order to complete useful work during this period. Naturally, the delay of this combinational logic plus the register setup time must be less than the insertion delay ($t_{dXY} + t_s < t_i$).

It should be noted that an input register in most cases will require some buffering to distribute clock and enable signals, forcing the ack to be delayed to some degree.

7.1. Hiding Small Insertion Delays

In some cases the additional latency incurred by even a relatively small insertion delay will be unacceptable. In these cases, latency can be minimised by considering the clocking of the synchronous module's input registers separately from the clocking of its output and state registers. Two clock trees are now generated, one small low-latency tree to clock the module's input registers and a larger one to clock the modules state and output registers. The larger insertion delay is hidden by initiating a new clock edge early from a tap within the delay-line. The total delay to the tap plus the clock-tree insertion delay ensures the state/output registers are clocked one clock period after the input register. This "skewed tree" scheme is only applicable in cases where the insertion delay of the state/output register clock tree is less than half a clock cycle.

Depending on the output port style employed it may in some cases be necessary to stall the clocking of the output registers. In general, schemes could be developed that clocked input, state and output registers at different times from different clock trees. For example, it may be possible in some designs to clock output registers before state registers – thereby reducing latency while maintaining correct operation. The introduction of additional clocks and matched delays moves the design methodology towards a bundled-data asynchronous one.

7.2. Multi-Cycle Clock-Tree Insertion Delays

If a synchronous IP block is sufficiently large the clock insertion delay may exceed a single clock period. In this case additional input buffering is required to prevent the clock period from being extended significantly. The single input register added in Section 7 must now be extended to a

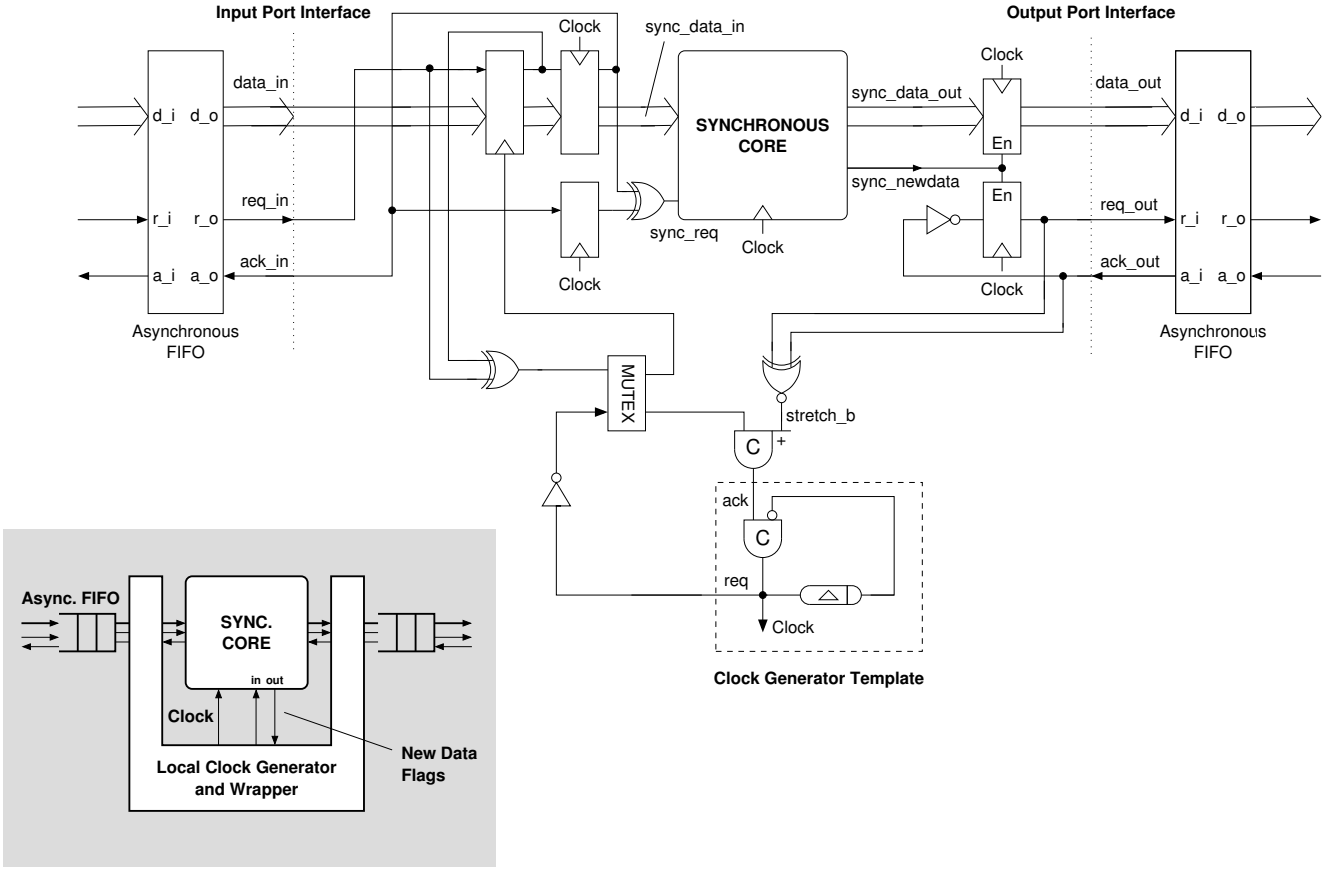


Figure 10. A locally-clocked synchronous block with a sampled/pausible-clock input port and registered/stretchable-clock output port. The example also shows asynchronous FIFOs used to buffer incoming and outgoing data. Note: the FIFOs employ a 2-phase handshaking protocol.

FIFO memory. The number of elements in the FIFO reflects the maximum number of rising clock edges which may be present in the clock tree at one time. Any newly arrived input data must wait at least this number of clock cycles before it is admitted into the synchronous block. We must also guarantee that data is always able to arrive at the head of the input FIFO before the clock edge used to admit it into the synchronous block.

In some schemes, *e.g.* pausible clocks, data is not necessarily admitted on every clock cycle. In these cases, we must carefully record on which clock cycle data has been scheduled to be admitted. The decision to admit data or not is readily available in many of the clock generators presented. This dual-rail value may be queued and subsequently used to admit data on the correct clock cycle at the associated input register. An outline of this scheme is shown in Figure 12.

Additional input buffering guarantees correct operation without extending the clock cycle time. The additional latency is unavoidable and can only be tackled by making input requests early with prior knowledge of the delays in the clock generator and clock tree.

The reader may consider the idea of 'promoting' data in the data FIFO so it may be read on an earlier clock cycle. Unfortunately, the clock cycle a data item will be admitted cannot be rescheduled in bounded time. Furthermore, the clock edges delineating these clock cycles will have already been dispatched. As they are already travelling through the clock tree their arrival time at the clock tree leaf cells cannot be influenced.

It should be noted that applying GALS techniques to systems composed from a small number of very large synchronous IP blocks is probably counterproductive even before clock-tree insertion issues are considered.

7.3. Related Work: Clock Tree Delays

Sjogren and Myers are first to highlight the issues associated with clock insertion delays and stoppable clocks in [23]. They focus on the need to handle substantial insertion delays but those of less than one clock cycle. Clock insertion delays are hidden in their handshaking protocol with the use of additional pipeline buffering.

It is useful to note that the 'state-holding gate' used in their stoppable clock circuit (and illustrated at the transistor level) is in fact an implementation of the asymmetric C-element as shown in Figure 3(b).

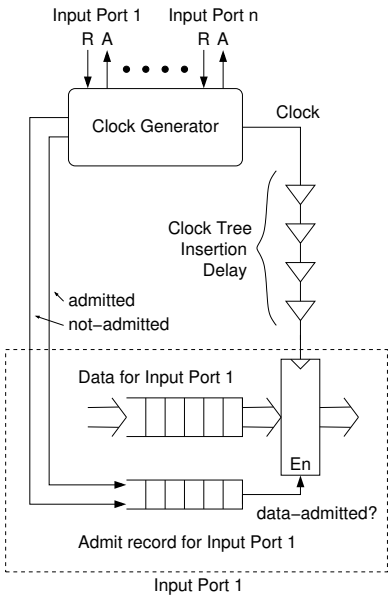


Figure 12. A record of arbitration decisions allows multi-cycle clock-tree insertion delays to be supported

8. Summary

A wide variety of ingenious aperiodic clocking schemes have been published to date. The aim of this paper has been to illustrate the similarities between many of these approaches. The paper has also presented a number of new mechanisms for supporting different kinds of input and output port and coping with clock insertion delays.

Current work is exploring the verification of local clock generators using the Veraci asynchronous circuit verifier [5]. Verified implementations of different port types together with formalised techniques for their composition, will form the major components of a GALS wrapper synthesis system. The use of data-driven clocks in the construction of on-chip networks with independently clocked routers is also being explored [17].

Acknowledgements

This work is supported by EPSRC (EP/D036895/1). The authors would also like to thank Alex Yakovlev and David Bormann for their comments on early drafts of this work.

References

[1] D. Bormann. GALS test chip on 130nm process. In *Proc. of the Second Workshop on Formal Methods for Globally-Asynchronous Locally-Synchronous Design*, 2005.

[2] D. Bormann and P. Cheung. Asynchronous wrapper for heterogeneous systems. In *Proc. Intl. Conf. on Computer Design (ICCD)*, 1997.

[3] A. V. Bystrov, D. J. Kinniment, and A. Yakovlev. Priority arbiters. In *Sixth Intl. Symp. on Advanced Research in Asynchronous Circuits and Systems (ASYNC)*, 2000.

[4] D. M. Chapiro. *Globally-Asynchronous Locally-Synchronous Systems*. PhD thesis, Stanford University, Oct. 1984.

[5] P. A. Cunningham. *Verification of Asynchronous Circuits*. PhD thesis, University of Cambridge, Jan. 2002.

[6] R. Dobkin, R. Ginosar, and C. P. Sotiriou. Data synchronization issues in GALS SoCs. In *Tenth Intl. Symp. on Advanced Research in Asynchronous Circuits and Systems (ASYNC)*, 2004.

[7] J. D. Garside. Processors. In J. Sparsø and S. Furber, editors, *Principles of Asynchronous Circuit Design: A Systems Perspective*, chapter 15. Kluwer Academic, 2001.

[8] F. K. Gürkaynak, S. Oetiker, H. Kaeslin, N. Felber, and W. Fichtner. GALS at ETH Zurich: success or failure? In *Twelfth Intl. Symp. on Advanced Research in Asynchronous Circuits and Systems (ASYNC)*, 2006.

[9] M. W. Heath, W. P. Burleson, and I. G. Harris. Synchro-tokens: A deterministic GALS methdology for chip-level debug and test. *IEEE Transactions on Computers*, C-54(12), Dec. 2005.

[10] M. B. Josephs and J. T. Yantchev. CMOS design of the tree arbiter element. *IEEE Trans. on VLSI Systems*, 4(4), Dec. 1996.

[11] J. Kessels, A. Peeters, P. Wielage, and S.-J. Kim. Clock synchronization through handshaking. In *Eighth Intl. Symp. on Advanced Research in Asynchronous Circuits and Systems (ASYNC)*, 2002.

[12] M. Krstic and E. Grass. New GALS Technique for Datapath Architectures. In *International Workshop on Power and Timing Modeling, Optimization and Simulation (PATMOS)*, 2003.

[13] M. Krstic, E. Grass, and C. Stahl. Request-Driven GALS Technique for Wireless Communication System. In *Eleventh Intl. Symp. on Advanced Research in Asynchronous Circuits and Systems (ASYNC)*, 2005.

[14] W. Lim. Design methodology for stoppable clock systems. *IEE Proceedings Computers and Digital Techniques*, 133(pt. E)(1), Jan. 1986.

[15] S. W. Moore, G. S. Taylor, P. Cunningham, R. D. Mullins, and P. Robinson. Self-calibrating clocks for globally asynchronous locally synchronous systems. In *Proc. Intl. Conf. on Computer Design (ICCD)*, 2000.

[16] S. W. Moore, G. S. Taylor, R. D. Mullins, and P. Robinson. Point to Point GALS Interconnect. In *Eighth Intl. Symp. on Advanced Research in Asynchronous Circuits and Systems (ASYNC)*, 2002.

[17] R. D. Mullins. Asynchronous versus synchronous design techniques for NoCs. Tutorial at the International Symposium on System-on-Chip, 2005.

[18] P. Nilsson and M. Torkelson. A monolithic digital clock-generator for on-chip clocking of custom DSPs. *IEEE Journal of Solid-State Circuits*, 31(5), May 1996.

[19] M. Nyström and A. J. Martin. Crossing the synchronous-asynchronous divide. In *Workshop on Complexity-Effective Design (WCED)*, May 2002.

[20] M. Pěchoucěk. Anomalous response times of input synchronizers. *IEEE Transactions on Computers*, C-25(2), Feb. 1976.

[21] F. U. Rosenberger, C. E. Molnar, T. J. Chaney, and T.-P. Fang. Q-Modules: Internally clocked delay-insensitive modules. *IEEE Transactions on Computers*, 37(9), Sept. 1988.

[22] C. L. Seitz. System timing. In C. A. Mead and L. A. Conway, editors, *Introduction to VLSI Systems*, chapter 7. Addison Wesley, 1980.

[23] A. E. Sjogren and C. J. Myers. Interfacing synchronous and asynchronous modules within a high-speed pipeline. In *Advanced Research in VLSI*, Sept. 1997.

[24] J. Sparsø and S. Furber. *Principles of Asynchronous Circuit Design - A Systems Perspective*. Kluwer Academic Publishers, 2001.

[25] I. Sutherland. Micropipelines: Turing award lecture. *Communications of the ACM*, 32(6):720–738, June 1989.

[26] W. S. VanScheik and R. F. Tinder. High speed externally asynchronous/ internally clocked systems. *IEEE Transactions on Computers*, 46(7), July 1997.

[27] K. Yun and A. Dooply. Pausible clocking based heterogeneous systems. *IEEE Transactions on VLSI Systems*, 7(4):482–487, Dec. 1999.

[28] Zhang et al. A 1-V heterogeneous reconfigurable DSP IC for wireless basebanddigital signal processing. *IEEE Journal of Solid-State Circuits*, 35(11), Nov. 2000.

Appendix A: Asynchronous Arbiters

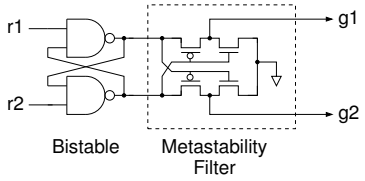


Figure 13. Mutual-Exclusion Element [22]

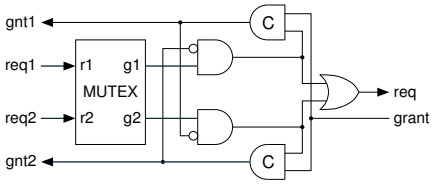


Figure 14. Arbitrated-Call [10]

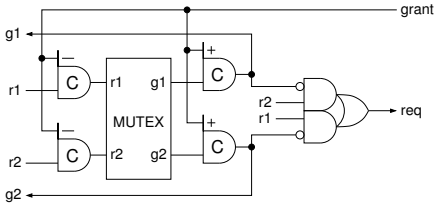


Figure 15. Tree-Arbiter Element with Eager Request (speed-independent implementation based on [10])

Modeling and Performance Analysis of GALS architectures

Sohini Dasgupta, Alex Yakovlev
School of EECE, University of Newcastle, UK
{Sohini.Dasgupta, Alex.Yakovlev}@ncl.ac.uk

Abstract— **In this paper we present a comparison of three clock control schemes and how it can be applied to an exisiting partitioned synchronous architecture to obtain a reliable, low latency and efficient Globally Asynchronous and Locally Synchronous architectures. The comparison highlights the advantages and disadvantages of one scheme over the other in terms of logical correctness, circuit implementation, performance and relative power consumption. We also present here circuit solutions for stretchable and data driven clocking schemes. These circuit solutions can be easily plugged into existing partitioned synchronous islands. To enable early evaluation of functional correctness, this paper proposes the use of Petri net modeling technique to model the asynchronous control blocks that constitute the interface between the synchronous islands.**

I. Introduction

Clocking circuits are becoming increasingly hard to design with larger chip sizes, higher clock rates and larger wire delays. The integration of various IP (intellectual property) cores on complex systems on chip requires a multitude of clock frequencies on a single die. Such integrations are enabled by modern deep sub-micron fabrication technologies in the form of chips with more than a billion transistors [12]. Globally asynchronous and locally synchronous (GALS) architectures aid such integration allowing synchronous independent blocks to interact through asynchronous communication channels.

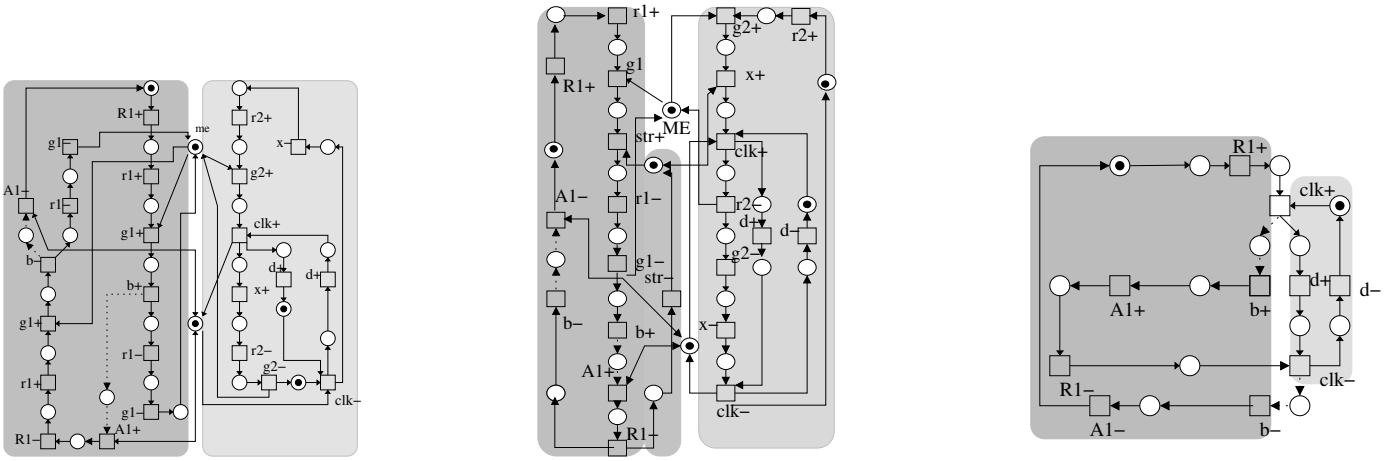
The GALS paradigm can be customised to meet the power and performance requirements to suite the target technology. There are a range of clocking strategies that can be applied to meet the above requirements. The modeling of the control circuit, enabled by tools like PEP [11] and its rendition into a gate level model using logic synthesis tools like Petrify [8], aids the exploration of the designs at a higher level of abstraction. This type of modeling abstractions are useful for analysis and validation of different design alternatives.

Contribution of the paper: The main goal of this paper is to present the comparison between three different GALS approaches. This comparison highlights the advantages and disadvantages of the three design solutions based on logical correctness, circuit implementations, power and performance analysis. The implementation of synchronous computational blocks are not cycle accurate, while the communication blocks are modeled in a cycle accurate manner. Petri net excels in its usefulness to model systems at higher levels of abstraction and tools like Petrify aid their translation into a gate level implementation. This type of modeling provides the designer with fast verification and implementation of the system. This paper presents the Petri net models of the three GALS architectures. These models are verified for correctness using in-house verification tools PUNF/CLP [9]. The verified models are fed to Petrify to produce logic equations for gate level implementation. We use two pre-synthesized blocks, namely, Mutual Exclusion Element [10] and FIFO [4] and these are plugged into the circuit implementation of each model obtained from Petrify. The gates are implemented on AMS $0.35\mu m$ technology library. This paper presents novel design solution for stretchable and data driven clocking scheme from prevalent conceptual models. GALS architecture with pausible clocking scheme is obtained from [3] and compared for efficiency and power consumption with the above mentioned approaches.

This research is supported by EPSRC grant (GR/S12036)

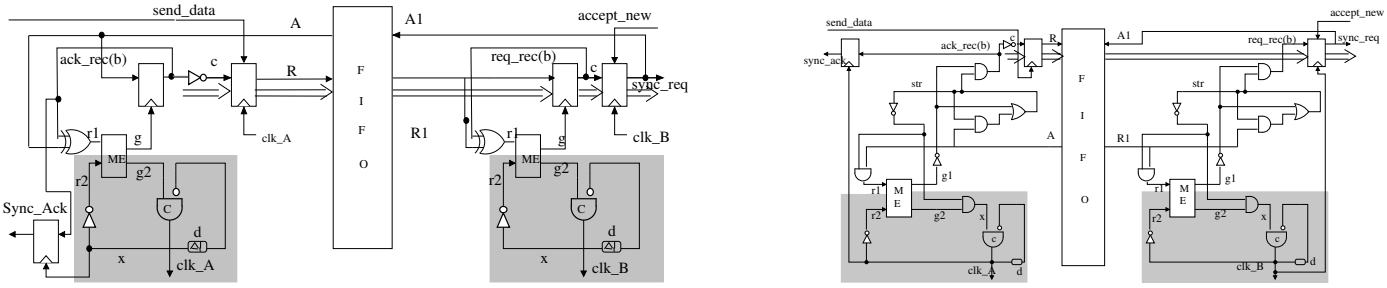
II. GALS systems and their models

To obtain a GALS implementation for a given multi processor system, three communication architectures can be employed. This section presents the conceptual Petri net models of the architectures and its implementation. This implementation is extended to a system with two clocked domains, one producer and the other receiver, to replicate communication between two synchronous islands. The two clocked domains communicate via a two stage asynchronous FIFO. For simplicity, the interaction of the communication interface with the synchronous module is not shown in the petri net models. This includes the signal *sync_ack* going to the synchronous module), which in turn releases an enable signal *send_data*, denoting the availability of data to send on the producer side. Similarly, on the Consumer side, the synchronous request(*sync_req*) is sent to the synchronous module after the reception of enable signal *accept_new* from it. Fig.1 depicts the Petri net models of the consumer block, i.e. the async-sync interface, of each of the three GALS clocking schemes. The dotted lines in three models denote that signal $b+ \rightarrow A1+$and $b- \rightarrow A1-$ take place in the presence of an enable signal *accept_new*, produced by the synchronous module, which is not depicted in Fig.1. These models were verified for functional correctness before feeding them into Petrify for a gate level implementation. The verification statistics can be found in [1]. Circuit level implementation of the async-sync interface models together with their sync-async counterpart, communicating via an asynchronous FIFO, thus



(a) Pausible clocking scheme (b) Stretchable clocking scheme (c) Data driven clocking scheme

Fig. 1. Petri net models of GALS architectures



(a) Pausible clocking scheme (b) Stretchable clocking scheme

Fig. 2. Clocking schemes: Pausible and Stretchable

obtained from Petrify is depicted in Fig.2 and 3.

Pausible clock: The pausible clocking scheme offers an elegant solution to metastability issue which comes into play when there is cross domain communication. Pausible clocks are characterised by a free running clock. A Mutual Exclusion element is inserted in the circuit to allow the clock to be interrupted when a data is ready to be transferred. The interruption of the clock enables safe transfer of asynchronous data. As shown in Fig.1, signals $g1$ and $g2$ are mutually exclusive and granting of $g1$ interrupts the clock. This leads to an asynchronous data transfer. This request is acknowledged on reception of the positive edge of the clock signal. The leftmost block and the FIFO block, depicted in Fig.2(a), constitute the interface between synchronous producer and asynchronous receiver. The interface arbitrates between granting in favour of the $r1$ signal, to transfer data to subsequent synchronous blocks or a clock request, to generate clock (clk_A) for its locally synchronous module. If the $r1$ is granted the data is latched in the first latch and the hold is released on the mutex. This allows clock request to win over the mutex. Therefore, data is stable before the clock arrives at the next stage of latch avoiding metastability at the second latch. The synchronous module always waits for a synchronous syn_ack. On reception of the synchronous $sync\_ack$, the module releases an enable signal for new data transfer. This type of design methodology is also explored in [2].

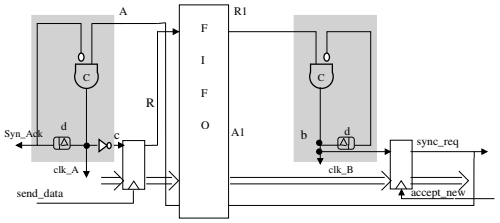
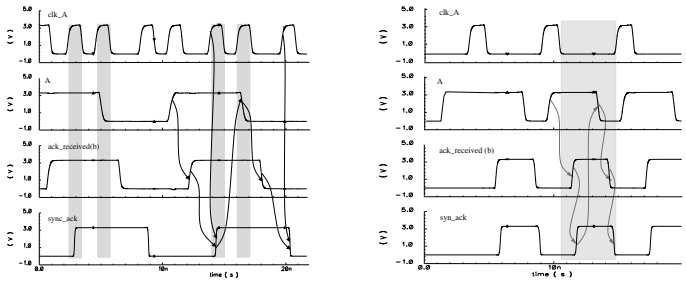


Fig. 3. Data driven clocking scheme

Stretchable clocking scheme: A stretchable clock can also be viewed as a free running clock like the pausible clock. The difference between two is that a stretchable clock knows in advance that the next clock cycle should wait for an asynchronous input. Therefore only in the absence of input request signals, the clock would be free running. This type of architecture leads to an increased throughput, since the request does not compete with the clock for an asynchronous data transfer. As shown in Fig.2(b), the assertion of the stretch signal prevents the clock from going high and remains asserted until signal $R1$ id deasserted. The synchronous module waits for a synchronous $sync\_ack$, in a manner similar to pausible clocking scheme. The signals



(a) pausible clock (b) stretchable clock

Fig. 4. Phase relationship at the producer block

ack_rec+(also denoted by b) and $clk+$ are mutually exclusive on the producer side due to signal str (this can also be seen on the consumer side, in the Petri net model shown in Fig.1(b), where req_rec+ is mutually exclusive to $clk+$). Therefore, positive edge of signal ack_rec cannot be synchronized on the positive edge of signal clk. If the signal ack_rec is synchronized to the negative edge of the clock cycle with a flip-flop, the system could run into a deadlock. This is due to the fact that if signal clk has already gone low before the triggering of signal $str+$, and then $str+$ occurs preventing signal $x+$(which causes $clk+$), from firing, signal $ack+$ would wait for the faling edge of signal clock, which would not be triggered till $str-$ occurs. Hence, ack_rec+ will never meet the set up and hold time of the falling edge of clock signal. Therefore the only solution is to use a latch, instead of a flip flop. The latch is made to sample the signal ack_rec when the clock is low. This synchronized ack_rec is then sent to the synchronous module which in turn sends an enable signal to indicate a data-ready-to-send status. This enable signal latches the $ack\_received'$ (also denoted by c) in the final set of latches to assert the request signal for sending new available data. A similar technique is presented in [6].

Phase relation between signals *clk_A, ack, ack_rec, Sync_ack* at the sync-async interface for stretchable and pausible clocking schemes, is depicted in Fig.4(a), (b). The shaded portion denotes the window when asynchronous data is received.

Data driven clock: In data driven clock scheme clock edges are produced in response to the presence of data at the input ports of the IP block. Therefore, the clock is not free running, unlike pausible and stretchable clocking schemes. The Petri net model of the async-sync interface of such an architecture is shown in Fig.1(c). Its gate level implementation is depicted in Fig.3. Since, power is an important issue in SoC applications, design methodologies which provide circuit solutions with reduced power consumption becomes highly attractive. This scheme significantly reduces power consumption as clock is only started when enough inputs have been received to carry out a particular computation. The circuit is switched off at other times. An extensive design solution for this approach can be found in [7].

A detailed description of the model and circuit level implementation of the three clocking schemes have been presented in [1].

III. Performance Analysis

To avoid the complexity of distributing a single global clock across the entire chip area and the varying power requirements for different blocks, the synchronous blocks can employ an independent internal clock. The frequency can be scaled depending on the performance requirements of the system. In this paper, it is assumed that the system is partitioned logically into synchronous islands communicating with other synchronous blocks through an asynchronous interface. The asynchronous interface interacts with the clock generator circuit of these synchronous blocks for cross domain data transfer. The Petri net models of the asynchronous interface and clock control circuit developed are fed to Petrify to give logic equations to build the gate level implementation of the architecture. These circuits are simulated on Cadence. We used mixed signal simulations to aid the monitoring of several signals using digital specification, while other parts of the circuit run analog simulations. This technique enables us to simulate a combination of both analog and digital signals. We avoided using complete analog simulation technique in order to write functional blocks in verilog to be incorporated in the system and small verilog codes to monitor and evaluate the metrics, discussed above, for analysing the system. Hence, the inputs and outputs could be monitored digitally. The core analog blocks of the circuit can be efficiently wrapped by digital blocks which preserves the precise estimation of delays within these analog blocks.

GALS system characterization parameters:

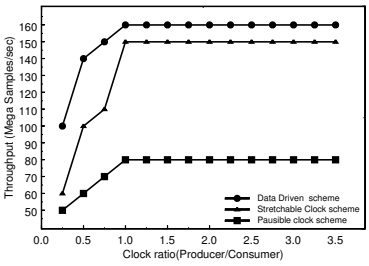
To characterize any design based on SoC applications, we need to define some metrics that are applicable to power and performance of a system. Similarly, for GALS systems we need to define such metrics. These are evaluated to analyze architectures for studying the effects of different system parameters on the performance of the system.

The metrics that are relevant for the analysis of pausible clock circuitry of GALS architecture are the number of times a clock is paused for a given simulation time and the average latency incurred due to such clock pauses. Another important system analysis metric for efficiency comparison is the throughput of the system, i.e., the average production/processing capacity of a system.

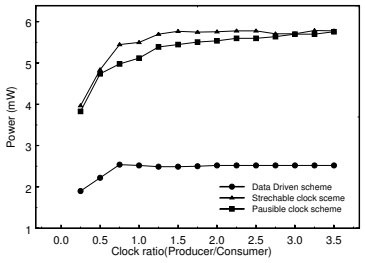
Due to increasing clock frequencies and smaller device sizes, it is becoming particularly important to consider the total power consumption metric in deciding on a particular design methodology. GALS based architectures reduce power consumption due to the ability to shift to an asynchronous mode when the local clock of the synchronous system is paused. Hence, a comparison of energy consumption in different GALS architecture would help choose between the different asynchronous communication circuitry. Therefore, an analysis of these metrics is useful for the designers to estimate the performance penalties in using one clocking scheme over the other.

IV. Circuit Level: Experimental Results

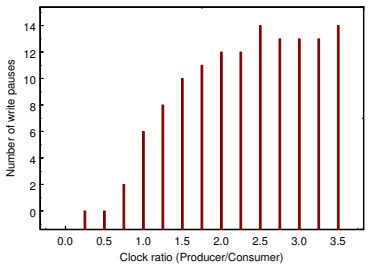
This section presents the results of power and performance analysis of GALS architecture with the three clock-



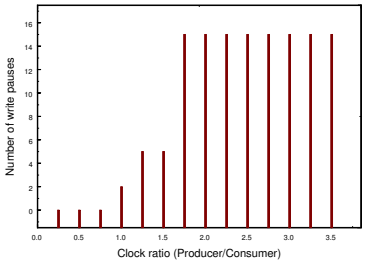
(a) Throughput analysis



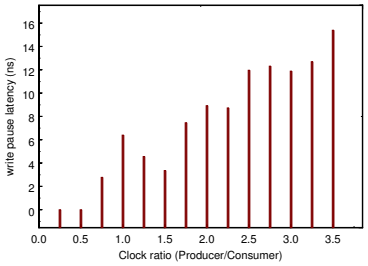
(b) Power consumption at interfaces



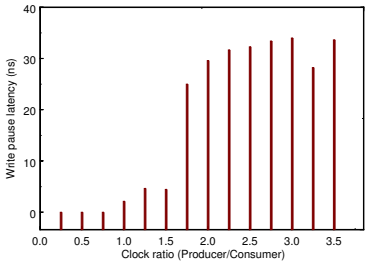
(c) No. of clock pauses in producer for pausible clock



(d) No. of clock pauses in producer for stretchable clock



(e) Clock pause time in producer for pausible clock



(f) Clock pause time in producer for stretchable clock

Fig. 6. Performance Analysis for GALS architectures

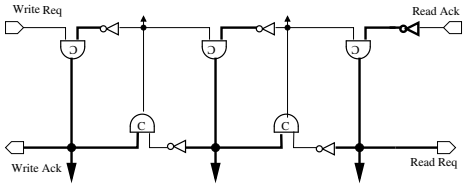


Fig. 5. FIFO design

ing schemes.

In our experimental setup, we use a 2 stage FIFO inter-module communication scheme. In the experiments we vary an input parameter, namely, the producer clock frequency. It is varied from 125 MHz to 1.75 GHz to observe the behaviour. The frequency of the consumer clock is maintained at 500 MHz. Higher frequencies are possible depending upon the complexity of the producer and consumer blocks. The ratio between the producer clock and consumer clock is called clock ratio. The clock ratio is varied from 0.25 to 3.5 in steps of .25. This allows to study the different phase relationship between the consumer and producer clocks.

Fig.6(a) shows the impact of changing clock ratio on the throughput of the communication channel. We observe that as the frequency of the consumer clock increases the throughput increases linearly up to clock ratio 1. This is because more data is being read by the consumer in the same period of time. After this time, the throughput reaches a saturation point. This is because the consumer clock operates at a lower clock frequency compared to the producer clock. Hence, there is no additional increase in throughput.

The throughput values obtained for stretchable and data driven clock are higher than pausible clock. This is due to the delay between two consecutive rising edges of the request signal $R+$. A detailed phase relation between signals that cause this delay is exemplified in [1]. It is observed that this delay is $12ns$ and $8ns$ for pausible and stretchable clocks, respectively. The throughput is maximum for data driven clock. It is higher than stretchable scheme since the signal A in the stretchable clocking scheme waits for synchronization for crossing over to synchronous domain to produce $Sync\_ack$. On the contrary, no such synchronization is needed for data driven clock as the clock starts when there is data to transfer and hence the signal A thus produced is already synchronized to the clock. This explains the trend of the curves in the graph that depicts the throughput of the different clocking schemes.

Fig.6(b) shows the power consumption, at an operating volatge of $3.3V$, with varying clock ratio. This plot refers to the effective power consumed over the time period needed to send a packet(same for all three protocols). We observe that as the clock ratio increases power consumption increases. This is because, as clock ratio increases, the throughput and operating frequencies of the synchronous islands, increases leading to an increased power consump-

tion. It is observed that the lowest power consumption is demonstrated by data driven clocking scheme as it doesn't have a free running clock and can be switched off when there is no data to send. Since, the implementation of the FIFO is same for all the protocols, complexity of port controller implementation of pausible and stretchable clocking schemes is also a factor that gives rise to such observations.

Fig. 6(c) and (d) shows the number of clock pauses in the producer for pausible and stretchable clocking schemes, as the clock ratio is increased. We see that as the frequency of the producer clock increases, the number of pauses increases. The asynchronous data transfer logic operates at a particular frequency. This frequency depends on the rate of production of R signal from the producer block and rate of reception of A signal from the consumer block. The transfer frequency becomes smaller than the frequency of the producer clock as the producer clock frequency increases and becomes higher than the consumer clock frequency. Hence, it takes longer to finish the cycle that de-asserts the grant on the arbiter. Due to this we observe more clock pauses as the period of the clock is too small to mask this delay. At lower frequencies, the time period is large enough to mask the pause during its lower half period.

The number of clock pauses in pausible and stretchable clocking scheme are comparable due to the scenario described above. But it can be observed from the graphs shown in Fig.6 (e) and (f), depicting total time incurred by these latencies that they are no longer comparable. The stretchable clocking scheme incurs longer latencies than than pausible clock. This is because the clock is only asserted when signal *str* is low. The arrival of signal A on the producer side or signal $R1$ on the consumer side, asserts signal *str*. When the producer frequency increases and becomes more than the consumer frequency, the FIFO gets filled up as more requests are produced than it can be consumed by the consumer module. Hence, the de-assertion of signal A is delayed. This phenomenon is exemplified in Fig.5. The FIFO is made up of a set of C-elements [5]. The shaded lines depict the signals that are asserted, while the non-shaded lines depict de-asserted signals. It can be observed that when the FIFO is full $Write\,Ack$ (A) remains asserted and is only de-asserted when an item of data is read from the FIFO, i.e. $Read\,Ack(A1)$ is asserted. The delay in the de-assertion of A, delays the de-assertion of signal *str*, which in turn delays the assertion of signal *clk*. This leads to a prolonged clock stretch. Such an occurrence is not observed in pausible clocking scheme. This is because, the reception of $b+$ immediately releases the grant on the arbiter and at this stage, the clock can arbitrarily win the grant to assert itself.

V. Conclusion

This paper presented the classification of different clocking schemes for Globally Asynchronous and Locally Synchronous architectures. These schemes have been modeled using Petri nets. A Petri net model of these interconnect architectures allows the designer to use existing logic synthesis tools, like Petrify to obtain gate level design solutions. Such solutions for GALS systems with stretchable and data driven clocking schemes have been presented in this paper. All the three clocking schemes exhibited reliable data transfer between the synchronous domains. A complex SoC can exploit any of the above given architectures depending on the requirements of the target system. These models can be plugged into existing partitioned synchronous blocks. These schemes can be extended to employ various power reduction methodologies in the wrapper without affecting the synchronous IP blocks.

In addition to the classification and design solutions for the three clocking schemes this paper also analyses the three systems on performance and power consumption criteria. Stretchable and data driven clocking schemes demonstrated higher throughput and lower power consumption charateristice, respectively, compared to the prevalent pausible clocking scheme. The stretchable and pausible clocking schemes are further compared on two other metrics, namely, the number of times the clock is paused or stretched and the total latency incurred by these pauses. Such an analysis aids the designer to make different design decisions based on power and performance.

Future work includes the development of a library of such Petri net models of each of the GALS clocking techniques, for different input coupling schemes (e.g. arbitrated, synchronized and sampled). We are also in the process of developing an automated GALS design tool which plugs these interconnects to already partitioned synchronous islands. This tool would ease the integration of the different interconnect models with the existing partitioned synchronous islands.

References

[1] S. Dasgupta, A. Yakovlev, Performance Analysis of Point-to-Point GALS interconnects. Technical Report NCL-EECE-MSD-TR-2006-114, Microelectronic System Design Group, School of EECE, University of Newcastle upon Tyne, UK, June, 2006.

[2] K. Yun, R. P. Donhue, Pausible clocking: A First Step Towards Heterogeneous Systems. In proceedings of International Conference on Computer Design, October 1996, Austin, TX.

[3] S. W. Moore, G. S. Taylor, R. D. Mullins, P. Robinson, Point-to-Point GALS Interconnect. In proceedings of Eighth International Symposium om Advanced Research in Asynchronous Circuits and Systems, 2002.

[4] I. Sutherland, Micropipelines: Turing Award Lecture. In Communications of the ACM, 32(6):720-738, June 1989.

[5] J. Sparso, S. Furber, Principles of Asynchronous Circuit Design - A System's Perspective. Kluwer Academic Publishers, 2001.

[6] J. Kessels, A. Peeters, P. Wielage, S. Kim, Clock Synchronization through Handshake Signalling. In International Symposium on Asynchronous Circuits and Systems, 2002.

[7] M. Krstic, E. Grass, C. Stahl, Request Driven GALS Technique for Wireless Communication Systems. In proceedings of 11th International Symposium om Advanced Research in Asynchronous Circuits and Systems, 2005.

[8] J. Cortadella, M. Kishnivsky, A. Kondratyev, L. Lavagno, A. Yakovlev, Synthesis of Asynchronous Controllers and Interfaces. Springer, Berlin, 2002.

[9] V. Khomenko, Model checking based on prefixes of petri net unfoldings, PhD thesis, University of Newcastle, (2003).

[10] C. Mead, L. Conway, Introduction to VLSI systems. Addison-Wesley Publication, October 1980.

[11] S. Melzer, S. Römer, and J. Esparza, Verification using PEP. In Proceedings of AMAST, 1996.

[12] S. Naffziger, The Implementation of a 2-core Multi-threaded Itanium Family Processor. In Proceedings of ISSCC, 2005.

Delay Insensitive Chip-to-Chip Interconnect Using Incomplete 2-of-7 NRZ Data Encoding

Jian Wu and Steve Furber
School of Computer Science, The University of Manchester
Oxford Road, Manchester M13 9PL, UK
Email: wuj@cs.man.ac.uk, steve.furber@manchester.ac.uk

***Abstract*— This paper proposes an incomplete 2-of-7 non-return-to-zero (NRZ) transmission method for high-performance and low power inter-chip communication. We use this method in the design of interfaces, including a link transmitter interface ('Tx i/f') and a link receiver interface ('Rx i/f'), which are employed to send and receive packets throughout a universal Spiking Neural Network chip multi-processor [1]. The focus of the design is on the protocol conversion between the delay-insensitive 1-of-4 return-to-zero (RTZ) on-chip CHAIN protocol [2] and the delay-insensitive incomplete 2-of-7 non-return-to-zero (NRZ) inter-chip protocol. In this design, the new protocol improves the performance and lowers the power consumption by reducing the inter-chip wire transition rate. Simulation shows that our communication interfaces are effective for low power and high inter-chip throughput.**

I. Introduction

Advances in integrated circuit technology allow more processors to be integrated onto a chip or a multi-chip system to achieve higher computing parallelism. A massively parallel multi-chip system incurs very high chip-to-chip capacitive loads. This causes the delay and power consumption of inter-chip communication to play an increasingly key role in the performance of a parallel system. However, conventional bus systems have difficulties in supporting such high connectivity because of their limited bandwidth. A solution to this problem is to use delay-insensitive point-to-point communication channels [3].

SpiNNaker is such a scalable multi-chip system designed specifically for the real-time simulation of large-scale spiking neural networks [4]. A major challenge in designing this large-scale neural network is to emulate the very high connectivity of the biological system. The high fan-in and fan-out of neurons suggests that an efficient asynchronous communication fabric is required. Delay-insensitive data encoding is therefore considered for use in the point-to-point asynchronous communication.

For the on-chip communication of the SpiNNaker multi-chip system, we use CHAIN technology which has an efficient delay-insensitive fabric for on-chip asynchronous communication [2]. However, a more power-efficient and time-efficient implementation is needed in inter-chip communications because an inter-chip transition has a more significant energy cost and inter-chip wire delays are longer.

In this paper, we propose the design of a link transmitter interface ('Tx i/f') and a link receiver interface ('Rx i/f') which extend the CHAIN communication system via inter-chip links. The function of the transmitter interface is to convert the on-chip CHAIN protocol into an inter-chip 2-of-7 NRZ protocol. The receiver interface performs the inverse conversion. The CHAIN protocol incurs four chip-to-chip transitions per 2-bit symbol, whilst the 2-of-7 NRZ protocol incurs only three chip-to-chip transitions per 4-bit symbol and therefore is more time-efficient and power-efficient.

II. Delay-Insensitive Communication

Delay-insensitive ('DI') communication is an attractive solution for system-level interconnection. In a delay-insensitive communication system, the receiver will return an acknowledge signal when it absorbs the data. The sender is allowed to issue the next data only after it has received the acknowledge signal. This feature makes the sending and receiving of the data able to operate at different speeds. Therefore delay-insensitive communication allows very flexible physical organization of chips [3].

Delay-insensitive codes are unordered, in which no code word is contained in another code word. Therefore the arrival of the delay-insensitive code can be recognized by the receiver, and then the receiver will respond to the sender after the detection. With this feature, the interpretation of the code word is not affected by delays. There are two main types of delay-insensitive codes: 1-of-n codes and m-of-n codes.

A 1-of-n code uses a group of n wires to transmit information. At each time, only one wire is allowed to be "1" to signal data. To detect the arrival of a 1-of-n data is easy because it only needs a simple n-input OR of the wires to perform the completion detection [5]. CHAIN uses such a 1-of-n data encoding protocol.

Another type of delay-insensitive code is the m-of-n code. The m-of-n code has a weight m out of length n [6]. That means each data word is encoded by m wires at level "1". An m-of-n code offers C_m^n possible symbols. There are many choices for m-of-n encoding, such as 2-of-4, 3-of-6 and 2-of-7 codes.

III. The CHAIN Architecture

CHAIN [2] is an architecture for SoC interconnect using delay-insensitive data encoding combined with a return-to-zero signalling protocol. Connections are built from narrow, high-speed, point-to-point links forming a network rather than a

bus. The data transferred through the CHAIN links is in a defined packet format in which an end-of-packet (EOP) signal is used to indicate the end of a data packet. Thus there is no need to ensure timing closure across the whole chip.

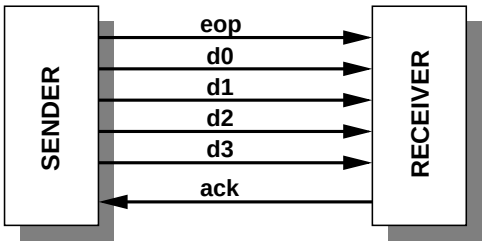


Fig. 1: CHAIN Link

A single CHAIN link is illustrated in Fig. 1. It has five forward-going wires plus one backward-going acknowledge wire. Four of the five forward-going wires are for normal data transmission using a delay-insensitive 1-of-4 code for the data encoding. When there is a transmission activity on one of the four wires, one of the two-bit codes 00, 01, 10, or 11 is represented. The fifth wire is used for carrying the end-of-packet (EOP) symbol, which is a packet control marker to set up the packer length. An acknowledge signal uses the 6th wire to realize self-timed control.

IV. Incomplete 2-of-7 NRZ Protocol

A. 2-of-7* code

Compared to a 1-of-n code, an m-of-n code can carry more bits every cycle but requires fewer wires and can have less or the same repeater stage logic size [5]. Because the throughput of DI communication is determined by the number of the bits transferred in a cycle, the m-of-n code is more efficient for a large message set. Table I shows a comparison of the cost and performance of some 1-of-n and m-of-n codes.

CODE	Possible Symbols	Useful Symbols	Throughput bits/cycle	Energy transitions/bit (RTZ)	Area wire/bit
Dual-Rail (1-of-2)	2	2	1	2	2
1-of-4	4	4	2	1	2
1-of-6	6	4	2	1	3
3-of-6	20	16	4	1.5	1.5
2-of-7	21	16	4	1	1.75
3-of-7	35	32	5	1.2	1.75

TABLE I: A Comparison of The Cost and Performance of Some DI Codes

As can be seen from Table I, the 2-of-7 code has a throughput of 4 bits per cycle. Therefore it is 2 times faster than the 1-of-4 code (2 bits per cycle) when having the same cycle time. In addition, the 2-of-7 code has the same transitions per bit (1 transition per bit) as the 1-of-4 code (CHAIN) but with 10% fewer wires (1.75 wires per bit) when representing a same group of binary values.

However, it is often not necessary to use all the symbols of an m-of-n code. For example, a 3-of-6 code has 20 possible symbols but only 16 of them are useful when used to present 4-bit binary codes. If only some of the symbols of an m-of-n code are used for data encoding, the code is called an incomplete m-of-n code (represented as m-of-n*).

In addition, the m-of-n* code also has some good features in code mapping and completion detection. In code mapping, the 2-of-7* code can be decomposed into a 1-of-3 code and a 1-of-4 code [5]. So the translation from two 1-of-4 codes into a 2-of-7* code is simple because there is no need to convert one of the two 1-of-4 codes. The arrival of a 2-of-7* code can be seen as the arrival of a 1-of-3 code and a 1-of-4 code or that of a 2-of-4 code. Hence the completion detection of the 2-of-7* code is also easier because the completion detection circuit for a 1-hot code is very simple. The implementation of the mapping and completion detection is described in section V. Furthermore, unlike in CHAIN, there is no need for an additional wire to carry an EOP signal in the 2-of-7* code because it has unused symbols. Therefore, we choose the 2-of-7* code for the chip-to-chip interconnect in our multi-chip system.

B. Transition Signalling Protocol

Both return-to-zero (RTZ) signalling and non-return-to-zero (NRZ) signalling can be used in delay-insensitive communication. RTZ signalling is a commonly-used encoding method. But NRZ signalling is more power-efficient and time-efficient. Fig. 2 shows the signal waveforms of the RTZ protocol and the NRZ protocol.

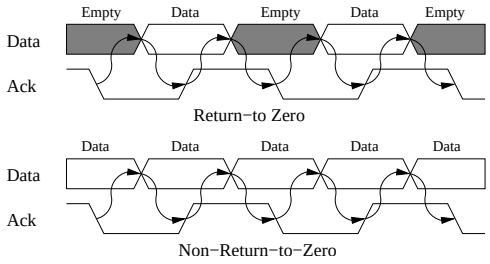


Fig. 2: Return-to-zero Protocol and Non-return-to-zero Protocol

In the RTZ protocol, the wires use Boolean levels to encode information. So each codeword or acknowledge signal has to use two transitions: The first one is from 0 to 1. The second one is from 1 to 0. In the NRZ protocol, the information is encoded as transitions. The transitions from 0 to 1 and from 1 to 0 both represent a logic 1 on the data wire or an acknowledge signal on the acknowledge wire. As a result, the NRZ protocol saves 50% of the transitions incurred by the RTZ protocol. Because the power required to generate a signal is nearly proportional to the signal's transition rate in CMOS logic [5], the NRZ protocol can reduce the power consumption by half.

In addition, the cycle time of data transmission can also be reduced because the NRZ protocol only uses one end-to-end cycle to send one symbol, whilst the RTZ protocol uses two end-to-end cycles.

The implementation of the NRZ protocol is often more complex and costs more chip area compared to that of the RTZ

protocol. But it is still a preferred solution in a system with high speed and low power requirements [8]. Therefore, the NRZ protocol is used as the inter-chip communication protocol in the SpiNNaker multi-chip system. The RTZ protocol is still used in the system for the on-chip communication because it is simple for implementation.

V. Implementation

Fig. 3 shows a block diagram of the structure of the Tx i/f and the Rx i/f. The multiplexer and demultiplexer perform as interconnection adapters between the CHAIN fabric and the NRZ 2-of-7 fabric. The encoder converts CHAIN 1-of-5 (including EOP) return-to-zero symbols to 2-of-7 non-return-to-zero symbols. The decoder performs the inverse conversion.

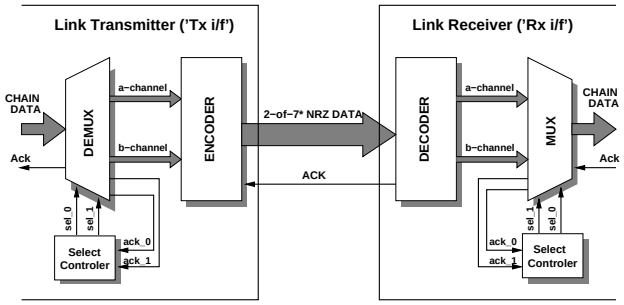


Fig. 3: Interface Architecture

Fig. 4 shows a diagram of the pipelined encoder and the pipelined decoder. The reason for adding the 2-of-7* pipeline latches is to increase the throughput of the interfaces by minimizing the cycle time. The loop between these latches defines the cycle time, which is determined by the lengths of the wires between the chips and the response time of the latches. The data rate of any delay-insensitive scheme is limited by the end-to-end cycle time of the system. A way to shorten the latches' response time is to add the latches in front of the phase converters. But because the code between the phase converters is NRZ code, completion detection is too complex for implementation. Therefore we add the pipeline latched in front of the code converters so that they can respond with the acknowledgement as early as they can without performing code convertion.

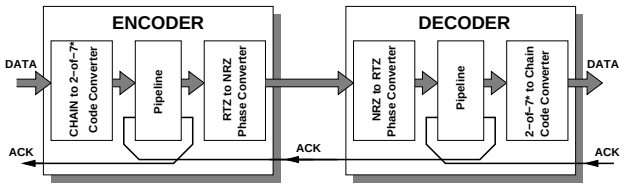


Fig. 4: Encoder and Decoder

A. The 2-of-7* Code Mapping

The 2-of-7* code can send 4 bits at one time, whereas the 1-of-4 code can only send 2 bits at one time. Given this, one 2-of-7* code can represent a pair of 1-of-4 codes.

The 2-of-7* encoding / decoding circuits are implemented using the approach of Delay-Insensitive Minterm Synthesis (DIMS) [9]. However, the implementation can be quite complex and inefficient if we select an unsuitable mapping of the binary values to code symbols. A method for choosing a suitable mapping is to decompose the 2-of-7* code into a 1-of-3 plus a 1-of-4 code, which represent 12 (3×4) symbols, and a 2-of-4 code plus the idle state (000), which represent 6 symbols [5]. To represent the EOP code, any other symbol of the rest of the 2-of-7 code can be used. In this case, we use 1100000. The mapping of the codes is shown in Table II.

EOP	1-of-4 code A	1-of-4 code B	2-of-7* code	
			Control	Body
EOP	A3 A2 A1 A0	B3 B2 B1 B0	C6 C5 C4	C3 C2 C1 C0
0	0 0 0 1	0 0 0 1	0 0 1	0 0 0 1
0	0 0 0 1	0 0 1 0	0 0 1	0 0 1 0
0	0 0 0 1	0 1 0 0	0 0 1	0 1 0 0
0	0 0 0 1	1 0 0 0	0 0 1	1 0 0 0
0	0 0 1 0	0 0 0 1	0 1 0	0 0 0 1
0	0 0 1 0	0 0 1 0	0 1 0	0 0 1 0
0	0 0 1 0	0 1 0 0	0 1 0	0 1 0 0
0	0 0 1 0	1 0 0 0	0 1 0	1 0 0 0
0	0 1 0 0	0 0 0 1	1 0 0	0 0 0 1
0	0 1 0 0	0 0 1 0	1 0 0	0 0 1 0
0	0 1 0 0	0 1 0 0	1 0 0	0 1 0 0
0	0 1 0 0	1 0 0 0	1 0 0	1 0 0 0
0	1 0 0 0	0 0 0 1	0 0 0	0 0 1 1
0	1 0 0 0	0 0 1 0	0 0 0	0 1 1 0
0	1 0 0 0	0 1 0 0	0 0 0	1 1 0 0
0	1 0 0 0	1 0 0 0	0 0 0	1 0 0 1
1	0 0 0 0	0 0 0 0	1 1 0	0 0 0 0

TABLE II: The 2-of-7* Code Mapping

This mapping method can significantly simplify the encoding and decoding circuits because it is not necessary to convert the 1-of-4 code of CHAIN. The encoding and decoding circuits are shown in Fig. 5.

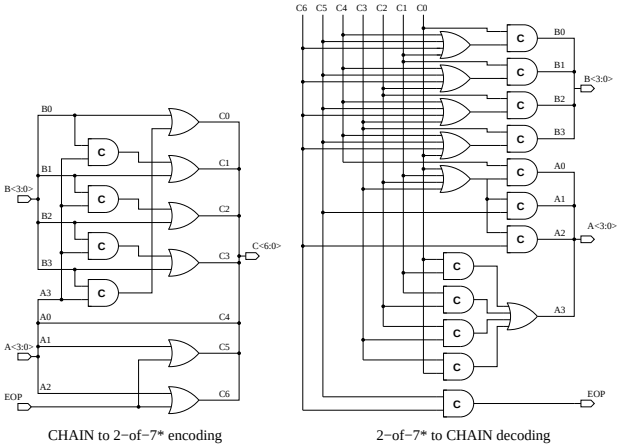


Fig. 5: Encoding and Decoding

The completion detection circuitry is also simplified thanks to this mapping method. The arrival of the 2-of-7* code is detected when the 1-of-3 code and the 1-of-4 code are detected. Furthermore, the 2-of-4 code can also be treated as two dual-rail codes in the completion detection circuitry. The

arrival of the EOP code is detected by the circuitry simply using a C-element. The implementation of the completion detection circuitry is shown in Fig. 6.

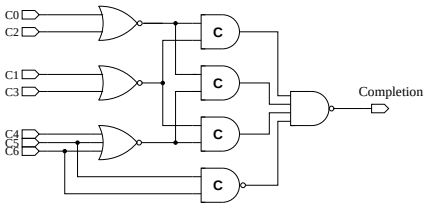


Fig. 6: Completion Detection Circuitry

B. Interconnection Adapters

In order to sustain the CHAIN link throughput, each conversion maps two 2-bit CHAIN symbols to a single 4-bit 2-of-7 symbol. Therefore, interconnection adapters are needed to perform the serial↔parallel conversion between the CHAIN fabric and the encoder/decoder.

To convert two serial CHAIN data streams into parallel transitions, we use a micropipeline demultiplexer (DEMUX) in the Tx i/f, which allocates the two data streams into the a-channel (A [3:0]) and the b-channel (B [3:0]). A micropipeline multiplexer (MUX) does the inverse operation in the Rx i/f by steering the data into CHAIN.

The operations of the DEMUX and the MUX are controlled by selection controllers. On receipt of the activation, the controller controls the sequential execution of two select actions (sel_0, sel_1) which determine into which channel the CHAIN data is sent by the DEMUX, or which channel the MUX data is received from. The selection controller is constructed from two S-elements, as shown in Fig. 7.

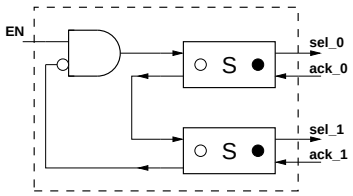


Fig. 7: Selection Controller

VI. Evaluation by Simulation

To evaluate the proposed 2-of-7* NRZ scheme, we designed two chip-to-chip interfaces for comparison. One uses the 2-of-7* NRZ protocol. The other uses the CHAIN protocol. Both of them were designed using the 0.13μm UMC CMOS gate library and simulated using Verilog with typical gate delays. The chip-to-chip wires were modelled with delays of 1.5ns in each direction and with a capacitance of 5pF.

The simulation results show that the 2-of-7* NRZ interface has a cycle time of 6.602ns and the CHAIN interface has a cycle time of 11.971ns. Because the chip-to-chip 2-of-7* encoding sends 4-bits in each direction per cycle, it gives a total throughput of over 605Mbits/s. Under the same conditions, the CHAIN interface, which sends 2-bits in each direction per cycle, gives a total throughput of only about 167Mbits/s. Thus the throughput of the 2-of-7* NRZ interface is about 3.6 times higher than the CHAIN interface. The energy consumption of the 2-of-7* NRZ interface is about 1/3 of that of the CHAIN interface. Both the higher speed and the lower power come at a price of larger area. Table III shows a comparison of the two interfaces.

	Throughput (Mbits/s)	Power (pJ/bit)	Area (Number of transistors)
2-of-7* NRZ	605	18.6	1971
CHAIN	167	54.1	164
Relative Performance	3.6	34.4%	12.0

TABLE III: Simulation Results

VII. Conclusion

In this paper we described a new chip-to-chip interface with a 2-of-7* NRZ protocol. Designing high throughput chip-to-chip communication interfaces requires careful design of both the architecture of circuits and code mapping. The simulation results show that this interface increase throughput by about 360% compared with the conventional CHAIN interface. At the same time, it decreases the power consumption by 1/3.

However, the implementation of the NRZ 2-of-7* encoder, decoder, and the completion detector increased circuit area and complexity. The extra area cost is acceptable because of the rapid shrink of transistor dimensions in modern integrated circuits.

References

[1] S. B. Furber, S. Temple and A. D. Brown, *On-chip and Inter-Chip Networks for Modelling Large-Scale Neural Systems*, Proc. ISCAS'06, Kos, May 2006.

[2] W. J. Bainbridge, S. B. Furber, *Chain: A Delay-Insensitive Chip Area Interconnect*, IEEE Micro, v.22 n.5, p.16-23, September 2002.

[3] S. B. Furber, A. Efthymiou and M. Singh, *A Power-Efficient Duplex Communication System*, Proc. of Int. Workshop on Asynchronous Interfaces, 2000

[4] S. B. Furber, S. Temple and A. D. Brown *High-Performance Computing for Systems of Spiking Neurons*, Systems of Spiking Neurons Proc. AISB'06 workshop on GC5: Architecture of Brain and Mind, Vol.2, pp 29-36,3-4 April 2006.

[5] W. J. Bainbridge, W. B. Toms, D. A. Edwards, and S. B. Furber, *Delay-Insensitive, Point-to-Point Inter-connect using m-of-n Codes*, Proc. of Int. Symp. On Asychronous Circuits and System, 2003.

[6] T. Verhoeff, *Delay-insensitive codes - an overview*, Distributed Computing, 3(1):1-8, 1988.

[7] R. Drost, *Architecture and Design of a Simultaneously Bidirectional Single-ended High Speed Chip-to-Chip Interface*, Ph.D. Thesis, Stanford University, Palo Alto, CA, Nov. 2001.

[8] S. B. Furber, J. Sparsø, *Principles of Asynchronous Circuit Design - A Systems Perspective*, Kluwer Academic Publishers, 2001

[9] J. Sparsø, J. Staunstrup. J., *Delay Insensitive Multi Ring Structures. Integration*, the VLSI Journal. Vol. 15. 1993.

An asynchronous spiking neural network which can learn temporal sequences

Joy Bose, S.B Furber, M. Cumpstey

School of Computer Science, University of Manchester, M13 9PL, UK
Email: {bosej@cs.manchester.ac.uk, sfurber@manchester.ac.uk, cumpstem@cs.manchester.ac.uk }

***Abstract*—We describe the design of an asynchronous spiking neural network that can learn and predict temporal sequences online. We concentrate on issues regarding the asynchronous functioning of the model such as timing relations between different autonomous components of the system.**

I. PROBLEM SPECIFICATION

Our aim is to implement a memory in spiking neurons that can learn any given number of sequences online (a sequence machine) in a single pass or single presentation of the sequence, and predict any learnt sequence correctly. A sequence is a series of symbols in temporal order, such as 'abc'.

The high-level description of the system (the functionality which is to be implemented in spiking neurons) is as follows: it takes as input a series of symbols constituting an input sequence, and for each input symbol it outputs a symbol which is the prediction for what the next symbol should be. If the input symbol is not part of a sequence previously learnt by the machine, the prediction will be incorrect but the machine will learn to predict correctly the next time the same sequence is presented.

Clearly, the prediction of the next symbol depends on the history of the sequence as well as the input symbol presented. In a system with infinite memory, the machine would be able to look as far back in the history as needed to produce an unambiguous prediction. However, we are using a finite neural memory which learns (writes to the memory) to associate the context or history of the sequence with the input, and so some noise is expected. The context or history itself is represented as a finite state machine, in which the new history is a function of the old history and the present input.

For example, if the sequence learnt is 'abcbd', the grammar learnt by the system can be represented as follows:

(Starting symbol) S → a

a → b	c → b	b → d
ab → c	bc → b	cb → d
abc → b	bcb → d	
abcb → d		

In this high-level description of the system, all the steps are assumed to take place in a perfectly synchronised way. However, in this paper we are interested in implementing this functionality using spiking neurons, which are essentially asynchronous, to get some insights on engineering and modelling issues in similar systems, as well as throw some light on the dynamics of interactions between biological neurons.

II. SPIKING NEURONS

A spiking neuron is a simplified model of a biological neuron that fires spikes or electrical impulses if an internal quantity of the neuron known as the activation exceeds a threshold. The activation is increased every time a spike fires at an input of the neuron, thus a neuron can be thought to accumulate input spikes. All spikes are of the same shape and information conveyed is only in the time of their firing. We assume that each spiking neuron fires only in response to its input spikes, i.e. there are no global control variables in the system that apply to all neurons. The neurons form layers, each layer performing a specific function and being connected to other layers, the spikes being transferred through the connection wires which may have different connection strengths, but we assume no wire delays.

III. SIMILARITY WITH BETWEEN A SPIKING NEURAL NETWORK AND ASYNC LOGIC CIRCUIT

In asynchronous logic design, communication takes place by transmission of electrical signals through wires, which can be considered similar to transmission of spikes in neural models. The electrical signals transmitted are in one of the two levels 0 and 1 (following binary logic), and the switching of levels could be considered as an event similar to firing a spike.

A standard asynchronous logic circuit has the handshake as its defining component. If we consider groups of spiking neurons interacting with each other, they show interactions similar to handshaking, in the sense that they can excite each other to generate corresponding bursts of spikes, which may be thought as the 'request' and 'acknowledge' signals. A latch, a standard component in asynchronous logic, can be implemented by a pair of spiking neurons which excite each other to fire a spike which keeps oscillating between them. The stored spike is released with the help of a third neuron that acts as a gate and resets the pair of neurons on receiving a 'request' control spike from another neuron. The released

spike can be considered as the 'acknowledge' signal in response to the 'request' signal.

IV. IMPLEMENTING THE SYSTEM USING NEURONS

In the high-level specification of the sequence machine, input symbols are associated with the context of the sequence and a prediction of the next symbol is generated. In the neural implementation, these symbols are encoded as bursts of spikes fired by layers of neurons. These spike bursts propagate like a wave through different layers of the system, each layer generating an output burst after receiving its input burst from the previous layer. The operation of the system is asynchronous, because there is no global mechanism such as a clock to synchronise the firing times of spikes and bursts of spikes across different layers.

In our system, we use a coding scheme known as rank ordered N-of-M code, in which we specify that N out of a total of M neurons in the layer can fire spikes in a burst, and the choice of the N firing neurons as well as the time order of their firing determines the code. The N-of-M code can be implemented by having a neuron that takes inputs from the M outputs of the layer and fires a resetting spike when N output spikes have fired in that layer that resets all the neurons. A neuron can be sensitised to a specific input firing order by multiplicatively decreasing the effect on activation increase for each successive input spike, and keeping the threshold of the neuron such that it fires when it has received the specific code. Finer details of the implementation of such a code using spiking neurons can be found elsewhere [2].

We use the wheel or spin model of the neuron in our implementation, in which the neuron can be visualised as a wheel spinning at a constant rate. The neuron has a quantity called activation or phase, which keeps on increasing at a constant speed, unless the neuron gets an input spike, which increases its activation (or phase) by an amount corresponding to the connection weight of the input neuron. The activation increases linearly till it reaches the threshold, which it will eventually, even if it gets no input spikes.

V. COMPONENTS OF THE SYSTEM

The system consists of the following neural layers as components: input, encoder, context, delay, address decoder, data store and output. Figure 1 shows the different component neural layers in the network and the connections between the components.

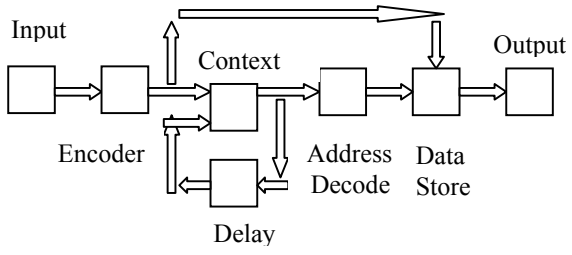


Fig. 1. Component neural layers of the sequence machine.

Most of these layers have fixed connection weights at their inputs and are essentially lookup tables, except for the data store (which is where the associations are written) and the context, which is like a finite state machine. We shall not mention here the detailed implementation of different components of the system, but the interested reader can find this elsewhere [1].

VI. TIMING DEPENDENCIES

For simplicity, we will consider only the input (ip), context (cxt), delay (del) and data store (store) layers, which are sufficient to achieve the basic implementation of the sequence machine. The spike bursts from these layers have to observe certain time constraints to enable the system to function.

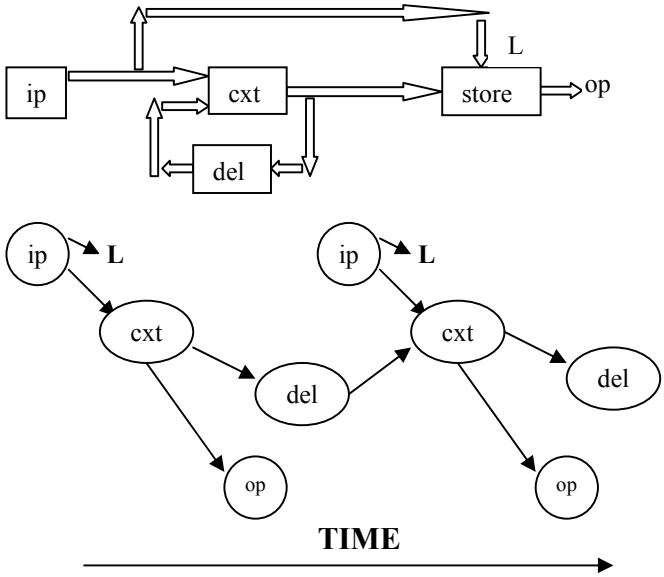


Fig. 2. The primary components of the system and their timing dependencies. Both the inputs to the cxt layer increase only the activation of the neurons, while the store layer has a normal input from the cxt layer to increase the activation, and a special learning input L which signals writing of the association of the ip and the cxt to the data store memory.

The timing dependencies between the bursts from different layers in the system can be summarised as follows:

1. The two input bursts to the context layer (fed back old context from the delay layer and the new input from the input layer) have to be approximately coincident, else the context neurons could start firing before receiving all the inputs, which would destroy the code transmitted by the burst, which is in the rank of the spikes. Therefore, the delay time (which is internal to the system) has to be matched to the gap between different inputs (which is external to the system).
2. The outputs of the store layer, which form the prediction of the next inputs to the system, must come before the next inputs to the system. So the gap between inputs should be bigger than the time taken for a spike burst to propagate through all layers of the system in the forward direction (excluding the

feedback through the delay layer).

3. The store neurons function in two modes: the normal or recall mode, in which they get input spikes from the context layer which increase the activations, and the learning mode, in which the association of the past context and the new input gets written to the memory. The learning mode gets triggered by the firing of the learning spikes from the input layer. We need to store the order of the context burst spikes until the next input burst comes and the association can be written to the memory. We do so by storing the order in the synapses of the neurons.
4. Also, in the learning mode, we have to make sure that the store neurons receive all the spikes from learning inputs (and complete writing the association) before receiving the spikes from the context (which are to be stored for the next association, when the new input burst comes). The latency of the context layer can ensure this.
5. Initially, the delay layer has no inputs (because there is no previous context) and consequently the context layer will fire slower (since its delay inputs are missing) than it would normally. We have to ensure that this does not destabilise the system, and the inter-burst interval stabilises after passing through a few layers.

VII. OTHER ISSUES

There are some other issues concerning implementation by asynchronous spiking neurons in general as well as some issues specific to the sequence machine system, which we have to deal with. They are briefly summarised below.

1. We need a signal to indicate the beginning and end of a burst, because all the neurons in a layer reset their phases when the burst begins. We consider the first input spike as the beginning of the burst to reset the phase of all neurons, and the output of the counter signifying that N neurons in that layer have fired, which is the maximum permissible according to the N-of-M code.
2. We need to store the rank ordering of the two input bursts to the context (from the delay and input layers) separately, since the increase of activation of the context on receiving any input depends on the position of that input in its respective burst. This is done by having two different feed-forward desensitisation neurons on the two kinds of inputs to the context, which keep track of the rank of the input spikes from both the layers.
3. The bursts of spikes have to be stable (not blow up or die out) and coherent (not interfere with each other, and clearly separated) as they pass through different neural layers in the system. This can be achieved by using a combination of feed-forward and feedback inhibition, as shown earlier [2].
4. We have to ensure that output spikes of any layer do not start firing before it has received all the input spikes from the layer before it, else this will spoil the code being transmitted. This can be arranged by having large thresholds and axonal or wire delays to ensure that this case does not happen.
5. We assume for now that the danger of the system being caught waiting forever will not happen, as long as the bursts are stable and coherent as described above.
6. The system is very sensitive to noise in the spike trains and so needs to be carefully engineered so that the times of firing are precise. However there is a degree of redundancy gained from using ordered N-of-M code, as the number of actual codes used in the alphabet is far less compared to the total possible number of codes, so some error is tolerable.
7. It is better to have the latencies (average time between the input and output bursts) of each layer comparable, in order to increase the stability of the system as a whole. To enable this, layers with more inputs should have higher thresholds and vice versa, because more inputs mean that the activations will rise quicker and the intra-burst separation will be small for that layer.
8. In our implementation of the system using spiking neurons, we have to ensure that the output spike burst from a layer in response to an input burst is equivalent (with respect to the code being considered, i.e. the rank and choice of neurons firing in this case) to what we would expect in the high-level model we are implementing (using ordered N-of-M coded symbols). The wheel model of spiking neurons that we have chosen meets this requirement.
9. The layers have control over their output spikes, but have no knowledge of their input spikes (unless each layer asks the layer before it). Therefore we have to ensure that the output spikes follow the correct code and there are no errors in generation of the output spikes, else the neurons in the next layer could keep waiting indefinitely for the expected number of input spikes. Having an N-of-M code solves this problem, as it is self error-correcting.
10. The robustness of the system depends on the stability of the bursts, so bursts emitted by different layers and the same layer during different waves should be well separated in time. Similarly, the inter-burst and intra-burst time separations should remain stable.

To ensure that the system works as planned, we have to use certain control mechanisms as discussed (while not compromising on the requirement that there should be no global variables in the system and neurons should fire based only on input spikes), but we have to minimise them as much as possible, in order to increase the flexibility of the system.

VIII. SIMULATION

We used a spiking neural simulator developed by M. Cumpstey [3] to simulate the complete system. The simulator is generic, event-driven, object-oriented and suitable for most common spiking neural models. We specify the network

configuration and the simulation file, and the simulator outputs a series of spikes from different layers along with their time of spiking. We had to make a few changes to the original simulator to incorporate some of the issues discussed.

Below is a diagram of the output of the simulator (with spike outputs of different neural layers against time) on being given a repeated input sequence 715171517151. The first time the sequence 7151 is given the output prediction is incorrect, but the system learns to predict correctly and the next time the prediction is correct. The sequence 7151 is used because it is the simplest sequence where we need to have knowledge of context to determine the successor of the symbol "1".

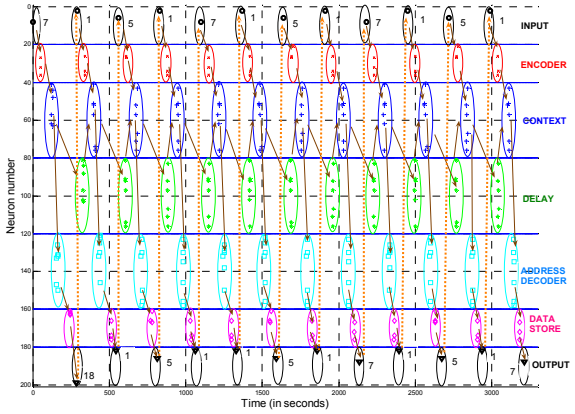


Fig. 3. Plot of spikes emitted by different layers in the sequence machine against time. The arrows denote causality, how a burst of spikes causes firing of another burst in the next layer after a delay. Spikes of the same shape and in parallel vertical bands belong to the same layer, and ellipses enclose bursts of spikes. The figure plots 12 different waves of spike bursts each triggered by an input spike, and forming the sequence 715171517151. After the first 7151 input when the system learns the sequence, the prediction of 15171517 on the output is correct.

In figure 3, spikes from different layers are plotted on the Y-axis and time on the X-axis. Spikes of the same colour belong to the same layer, and the arrows show how a burst of spikes from one layer causes the next layer to fire a burst after some time. We can see from the diagram that the spike bursts from different layers are coherent, stable, well behaved, and follow the timing dependencies mentioned. They also implement the high-level sequence machine by learning the given sequence 7151 in a single pass, and predicting it correctly in the second and third presentation of 7151.

IX. Conclusion and Future Work

We have shown that it is possible to build a system out of asynchronous spiking neurons that can perform the high-level functionality of the sequence machine.

We have mentioned how common components in traditional asynchronous design, such as the latch and the handshake protocol, might be implemented in spiking neurons. If we adapt traditional asynchronous design components such as latches to take into the additional constraints of spiking neural implementation, we could potentially reach a stage where it is possible to translate any spiking neural network into its equivalent asynchronous logic circuit, enabling us to make use of the synthesis tools available in asynchronous logic design to analyse a spiking neural network.

References

[1] An associative memory for the on-line recognition and prediction of temporal sequences, by J. Bose, S. B. Furber and J. L. Shapiro, in proceedings of International Joint Conference of Neural Networks (IJCNN 2005), Montreal, Canada, 31 July - 4 August 2005.

[2] A system for transmitting a coherent burst of activity through a network of spiking neurons, by J. Bose, S. B. Furber and J. L. Shapiro, in proceedings of 16th Italian workshop on Neural nets (WIRN 2005), Vietri sul Mare, Italy, 8-11 June 2005.

[3] SpikeNetwork: A spiking neural simulator, by M. Cumpstey, private communication.

Error Checking and Resetting Mechanisms for Asynchronous Interconnect

Yebin Shi and Steve Furber
School of Computer Science, University of Manchester
Oxford Road, Manchester M13 9PL, UK
Email: shiy@cs.man.ac.uk, steve.furber@cs.man.ac.uk

***Abstract* - Shrinking process technology and the increasing complexity of integration pose many challenges to the SoC industry. There has been a growing interest in asynchronous interconnect which can provide better support for Intellectual Property core reuse without the problems caused by synchronous interconnect. Meanwhile advanced integrated circuit fabrication is leading to the unreliability of circuit, which in turn leads to a demand for fault tolerant VLSI circuit. CHAIN is a packet switched on-chip network based on asynchronous interconnect using 1-of-4 return-to-zero (RTZ) code, and we use 2-of-7 non-return-to -zero (NRZ) code for inter-chip links. In this paper, we investigate the transient errors that could occur on both CHAIN and the inter-chip interconnect, and propose mechanisms for checking and resetting the links to correct or contain the errors.**

I. Introduction

A variety of Systems-on-Chips (SoCs) are widely used in many fields such as communication, computers and multimedia devices. As process technology shrinks, the complexity of SoCs has been increasing dramatically. Unfortunately, this poses some special problems. The most significant problem is clock synchronization since the clock skew will deteriorate continuously with increasing chip complexity [1]. It has become a difficult task to design a balanced clock tree to provide a global clock with reasonable clock skew, even impossible. Globally asynchronous interconnect is a promising alternative [2], which enables designers to integrate a number of synchronous modules into a Globally Asynchronous Locally Synchronous system, removing the problem of global clock synchronization.

Moreover the shrinking feature size of VLSI has important impact on on-chip communication architecture and the reliability of a circuitry. Shared buses, as used in a traditional on-chip communication infrastructure do not meet the demands of modern complex SoC design for high performance and low power consumption. Recently switched networks on chip have been proposed as a means to increase performance and to reduce the power used by on-chip communication [3]. Meanwhile the scaling down of feature size has made modern integrated circuits more susceptible to a number of factors, such as α particles, cosmic radiation, crosstalk, and power bounce. The reasons for this susceptibility are attributed to the smaller gate oxide thickness, lower supply voltage, and lower noise margin [4].

The APT group at the University of Manchester has proposed an asynchronous network-on-chip mechanism - CHAIN (CHip Area INterconnect) - as an on-chip interconnect [5], which was first implemented in a smartcard chip. Asynchronous arbiters, router control units and multiplexers are incorporated so as to complete the routing of packets, and pipeline latches are exploited along long wires to buffer the signals to increase link throughput. CHAIN uses a 1-of-4 code which is one-hot encoding, resulting in a simple implementation of the completion signal for the asynchronous handshake protocol. As described above, the possibility of the vulnerability of CHAIN to all kinds of transient errors will increase under the circumstance of very deep submicron process.

In this paper, the results of transient errors occurring on CHAIN are investigated, and corresponding error checking and resetting mechanisms are proposed. In section 2, the frameworks of two typical asynchronous communication systems based on CHAIN are introduced in detail. In section 3, we present the related error detection methods used for asynchronous interconnect. Then in section 4 a resetting approach to help the interconnect recover from erroneous scenario is discussed. Finally we give our conclusions.

II. The Asynchronous Communication Systems

A. *On-chip Interconnect*

The system framework we use in this work consists of one hardware communication channel and an external test bench which includes a transmitter (TX), a receiver (RX), an Error injector, a result checker and a monitor as shown in Fig 1. For simplicity, we only introduce one CHAIN channel into this asynchronous communication system, which means only one link within CHAIN interconnect is selected between TX/RX, routers or arbiters. Each module in the test bench is briefly explained as follows:

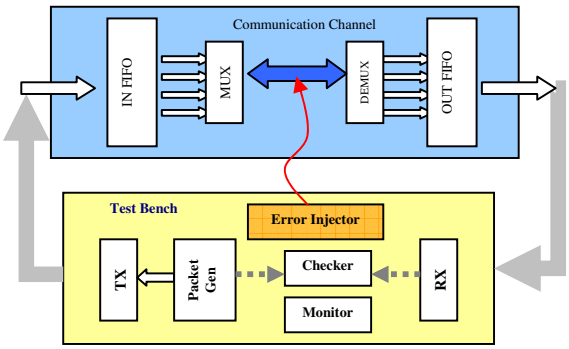


Fig 1 a simple asynchronous communication system based on CHAIN and its simulation environment

Packet Gen generates random packet data, including the address, ID, payload and parity/CRC fields with 'constraint randomize' provided by SystemVerilog. TX sends out each packet in bytes to the interface of IN FIFO located in the

design part, and to pad EOP (End of Packet) at the end of each packet. RX receives each byte data, re-forms the data into packets, and feeds them back to a Checker within the test-bench. The Checker module is able to compare the data from RX and TX, and calculate the parity or CRC of received packets. Monitor will collect the reports from the Checker and RX modules and take different actions according to the types of occurring errors. An error Injector module is used for injecting random errors onto CHAIN, and the frequency and duration for each spike can be randomized by applying different constraints.

Additionally the MUX (serializer) module serially forwards each of four 2-bit symbols from the parallel output of the FIFO with using a 1-of-4 code to CHAIN. Four select signals issued from 4 S-elements in order control the latch arrays, such that each of the four parallel data from the FIFO will pass though them as shown in Fig 2 and Fig 3. The final serial data is formed by an OR operation of each bit for four groups of data with the same bit position.

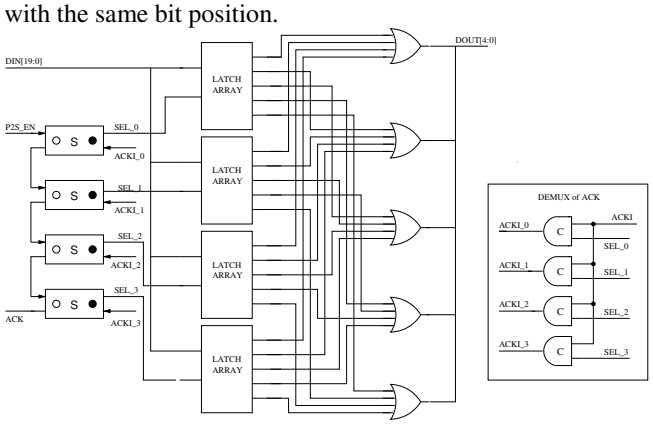


Fig 2 an asynchronous MUX

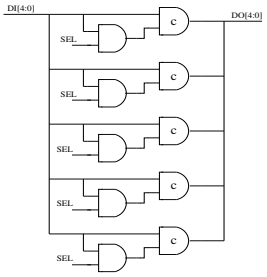


Fig 3 a latch array of C-elements

The DEMUX (deserializer) module buffers four symbols from CHAIN and then forwards all of them in parallel to the OUT FIFO on the link. In detail a control part is used to generate four select signals, ‘sel_0’, ‘sel_1’, ‘sel_2’, ‘sel_3’, which enable four paths of output data. The STG (Signal Transition Graph) of this controller is shown in Fig 4. In this graph, ‘di_cd’ denotes the completion signal for data input of DEMUX; ‘do0_cd’, ‘do1_cd’, ‘do2_cd’, ‘do3_cd’ respectively denote the completion signals of four groups of output data; ‘acki’ and ‘acko’ denote the acknowledge from a receiver and to CHAIN. Therefore we incorporate this simple controller synthesized by Petrify with C-elements to build a DEMUX.

B. Inter-chip Interconnect

A NRZ 2-of-7 code is implemented on inter-chip links to provide better performance with less wire cost and lower power consumption as shown in Fig 5, and specific transceivers need to be used for the conversion between 2-of-7 code and 1-of-4 code on CHAIN and the conversion between RTZ and NRZ.

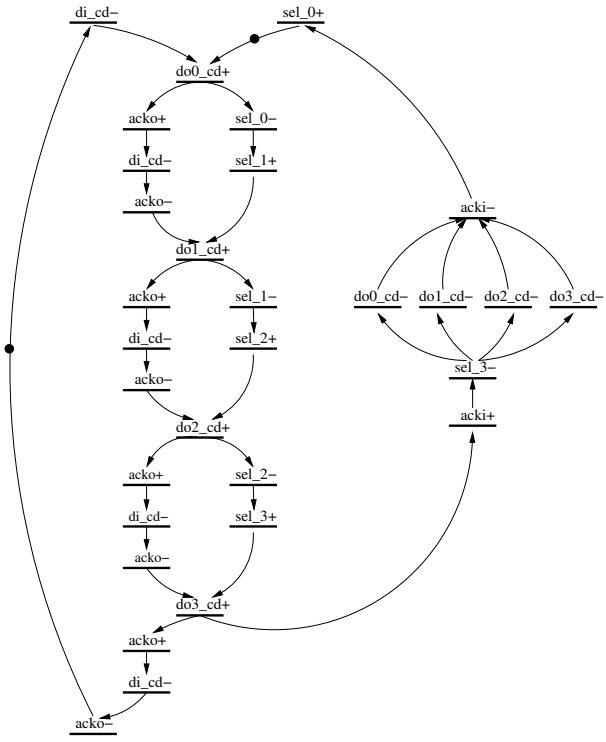


Fig 4 STG of the control part of DEMUX

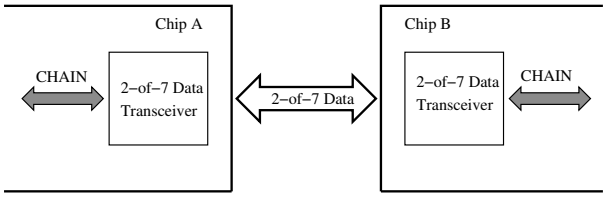


Fig 5 Diagram of chip interconnect

In this work, we also design a communication system as shown in Fig 5, consisting of an inter-chip link, CHAIN and the verification environment which is almost the same as the model used in the previous on-chip communication system. This inter-chip interconnect using NRZ 2-of-7 code is our target to inject spikes so as to simulate the error scenarios of a whole integrated system based on a printed circuit board suffering from some noise sources.

III. Error Detection on Asynchronous Interconnect

A. *Physical Layer Error Detection on CHAIN*

During the idle state between two valid data symbols propagated on CHAIN, if errors are induced, some unexpected bits of data could be introduced into the data sequence transferred on it. This insertion of extra data causes a sync error, which can be detected by a physical unit on the receiver. The receiver continually catches only three or fewer EOPs rather than the normal four EOPs in each packet. What's more, all the data in the following packet are shifted by one or more symbols. Consequently some EOP symbols appear on the beginning of the next packet.

In order to make the communication recover from the above errors, we can use two different methods. The straightforward way is to report the sync error to a relevant monitor when the receiver recognizes a sync error by detecting an unaligned EOP. This monitor will take some measure, for example resetting the link with the error occurrence to clear any remaining data in different parts of the link. However, at least two packets remaining on the link will be discarded in order for the link to recover from errors. To reduce this kind of packet loss, some other solutions may exist, since we don't need to drop packets apart from the current packet containing the erroneous data. Therefore the current packet with the error has to be discarded, while the receiver is able to continue to receive the next packet from CHAIN by shifting it by some bits. The second approach to shifting data could achieve better performance in terms of packet loss and timing, admittedly at the expense of circuit complexity compared with the first method.

The next scenario we need to consider is that errors could happen during the data transfer state of CHAIN. Soft errors have no significant impact on any data wire with the value '1' due to their duration of less than hundreds of picoseconds [6] and the nature of the delay insensitive handshake protocol forcing the pipeline latch to wait for the arrival of a valid bit. But it is different for the rest of wires with '0' values as more ones will probably be captured by the receiver during a data transfer on CHAIN if a positive glitch is coupled onto one of these wires with low values. We define this type of error as a protocol error. Because they do not cause sync problems or extra data, we just need to drop this packet. However the subsequent packets are not affected, so it is not necessary to re-align the data sequence or to report it to some monitor unit unlike the previous error scenarios.

B. *Data Link Layer Error Detection on CHAIN*

Error detection codes are used to provide data validation on the data link layer of most communication systems, which is based on the observation that not all the errors can be checked on the physical layer. For multiple errors on CHAIN during one packet transfer, 4 errors G0, G1, G2 and G3, as shown in Fig 8, occurring on idle states of CHAIN, lead to 4 extra symbols being inserted into a normal data sequence. Consequently the receiver is unable to identify it as either a sync error or a protocol error. In this case, the packet length will be increased by one byte so that the receiver can identify this error by checking the length field within the packet. We could not exclude some extreme cases in which the packet data are changed with the same packet length though the possibility of the change is quite low. Therefore parity/CRC codes applied on the data link layer can be employed with the previous methods on the physical layer to check the validity of a packet data.

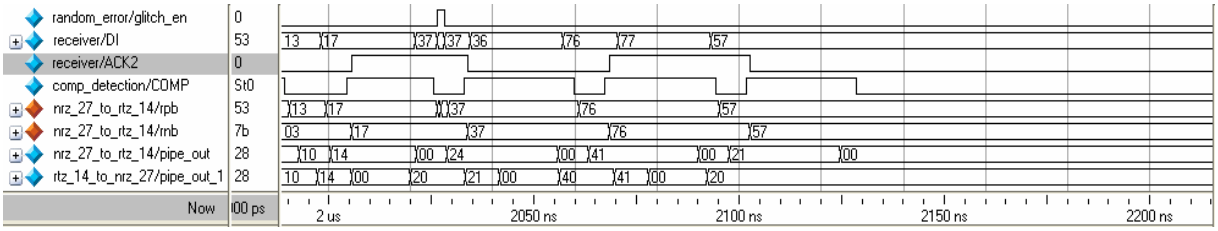


Fig 6 A Deadlock on an inter-chip link

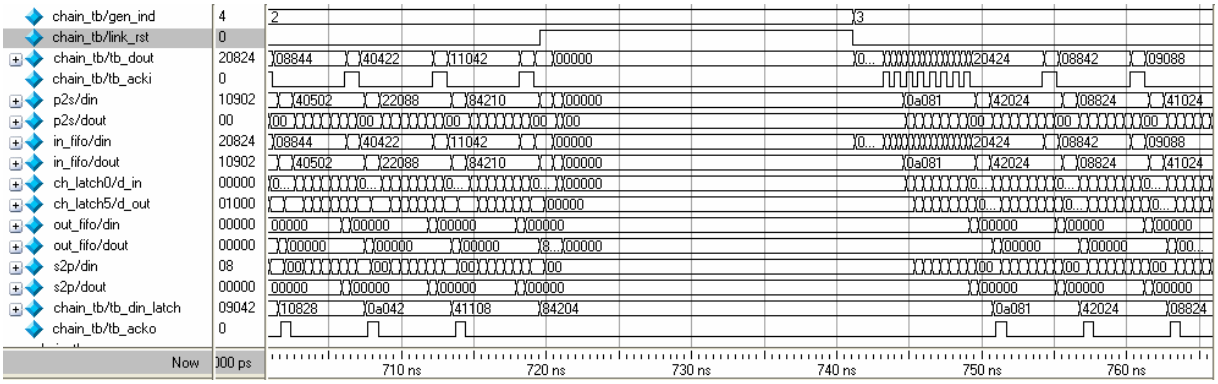


Fig 7 A Reset operation to recover the link from a deadlock

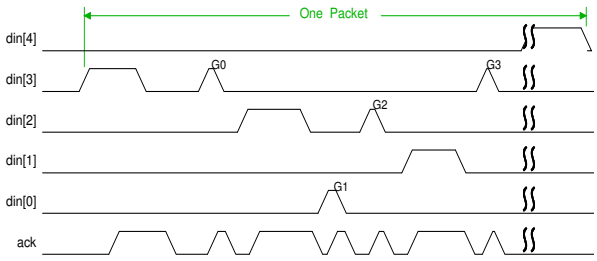


Fig 8 Four transient errors during idle states occurring in one packet

C. *Inter-chip Link Error Detection*

Random errors have a serious impact on the complex delay insensitive circuit since randomness of errors is able to break the basic handshake protocol. In fig 6, the glitch coupled with a wire of the output of the sender module changes DI from '37' to the other value, which causes the decoding circuit nrz_27_to_rtz_14 to work in a wrong way. The output signal pipe_out should be the delayed version of pipe_out_1, but the value of pipe_out is '24' rather than the correct value '20' at about 2030ns. Unfortunately the value '24' is viewed as correct data by the completion module within this decoder, causing three early issues of the completion signal (COMP). As a result, a deadlock on this inter-chip link arises from an early acknowledge back to the sender module but no further acknowledge is available to end the data propagating on the inter-chip link. As for a deadlock occurring on the inter-chip link, we can set a timer to monitor the status of this link. From a sender's point of view if this link has not received any acknowledge for the transmitting data and has halted for a long time beyond some reasonable threshold, a deadlock has probably happened and therefore a reset signal should be asserted to clear CHAINs on the two ends of the inter-chip link and any other relevant units throughout this link.

IV. A Reset Mechanism on CHAIN

Although an error cannot cause an on-chip link to deadlock even in the worst case, it will probably push the link into an erroneous state and it cannot transfer data correctly until we adopt some measures to restore it. As one simple option, we can construct a dedicated local reset signal, which is driven by the receiver and connected to any unit on the link including CHAIN, the two FIFOs, both of the MUX and DEMUX, and even the transceivers of a inter-chip link in order to clear all the remaining data. A deadlock happening during inter-chip communication or a sync error on CHAIN within a chip can be eliminated by applying a resetting mechanism.

As shown in Fig 9, the reset signal is introduced into almost all C-elements along the link as one input of the acknowledge signal, and when valid it will force every C-element to able only to receive new zero values. Therefore the receiver stops sending back acknowledge signals to the previous unit on the link and asserts the reset signal to the Link. Then when the transmitter takes an acknowledge signal with a '1' value, it drives all wires to '0' until the receiver finds the link idle for a while. Finally the receiver ends the resetting procedure by driving the reset signal low and prepares to sample the next packet. Fig 7 shows a resetting operation on CHAIN, and the reset signal is not cancelled until all data remaining on the link is removed.

Admittedly, a reset mechanism means that all the data after the corrupted data will be dropped. How much data are lost is mostly determined by the depth of the FIFOs and pipelines in CHAIN.

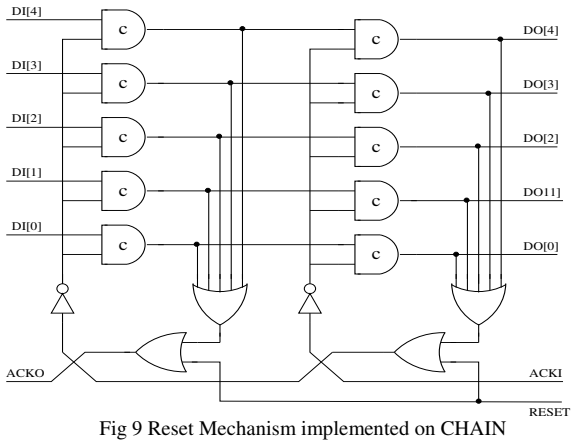


Fig 9 Reset Mechanism implemented on CHAIN

V. Conclusions

In this paper, we presented an environment to simulate the behavior of CHAIN when suffering from transient errors and analyzed all possible effects due to errors. Most of them are erroneous data with uncertain length inserted into packets according to the simulation result, and no deadlock arises. However the effect of an error injection is different for an inter-chip link using a 2-of-7 NRZ code and a random glitch may cause deadlock in some case. Therefore we discuss different detection methods for CHAIN and an inter-chip link in detail. Meanwhile in order to restore on-chip or inter-chip communication from an erroneous state, we propose a resetting circuit mechanism for asynchronous interconnect which can effectively clear all the data remaining on the links and guarantee the correctness of subsequent data transfers despite the fact that the loss of data is unavoidable.

References

[1] Jan M. Rabaey et.al, "Digital Integrated Circuits a design perspective", second edition, Pearson Education Inc., 2003, Page: 549-550

[2] DM Chapiro. "Globally Asynchronous Locally Synchronous Systems", PhD thesis, Stanford University, 1984.

[3] William J. Dally, Brian Towles, "Route Packets, Not wires: On-Chip Interconnection Networks", DAC 2001, June 18-22, 2001, Las Vegas, Nevada, USA.

[4] Allan Johnston, "Trends in radiation Susceptibility for Advanced Semiconductor Devices", 15 September, RD49 SEMINAR ON COTS, http://rd49.web.cern.ch/RD49/RD49News/ Allan_johnston.pdf.

[5] John Bainbridge, Steve Furber, "Chain: A Delay-Insensitive Chip Area Interconnect" IEEE Micro, vol. 22, no. 5, pp. 16-23, Sept/Oct, 2002

[6] Steven V. Walstra, Changhong Dai, "Circuit-level modeling of soft errors in integrated circuits", Volume 5, Issue 3, Sept. 2005 IEEE, Page(s):358 – 364.

On-Chip Phase Regeneration Circuits

Crescenzo D'Alessandro, Alex Bystrov, Alex Yakovlev
Microelectronics System Design Group
School of Electrical, Electronics and Computer Engineering
University of Newcastle upon Tyne, UK

Abstract— **Designs which require a phase relationship between two signals to be maintained along a link benefit from the use of repeaters which actively regenerate this relationship. This paper discusses some implementations of phase-regeneration circuits and attempts to introduce the reader to the issues encountered in the design of such circuitry. Simulation results are provided with discussion on the relative performance.**

I. Introduction

The design of reliable interconnects for on-hip block-to-block communication is becoming a crucial point for the development of new on-chip architectures, so much that designers now talk about communication-centric design.

In some cases a given time relationship between two signals is available and needs to be maintained. One such case is the phase encoding protocol initially proposed by D'Alessandro et al. in [2]. In the work the authors propose the use of two out-of-phase signals to carry informations; the binary phase relationship (+/-) encodes the bit of information being sent across the communication link. The work was then furthered in [3] for multiple-rail implementation, where the link is composed of several communication lines and the symbols are encoded in the order of occurrence of transitions on the lines. It is shown that the phase corruption introduced by the link, mainly through cross-talk between the lines, is significant and can greatly affect the correct recovery of symbols. The assumption that the phase of the signals is kept along a path without additional circuitry still holds for short paths, as the cross-talk induced by each line on the neighbouring one is limited, thanks to the limited coupling between the wires.

Devices which are able to recover the initial timing relationship between two incoming signals are therefore required. These will make sure that two edges travelling along a path will maintain the given phase throughout the path: given as input two identical signals delayed by a variable amount outputs the same signals with time delay always $\pm\delta$. In particular, in this work we focus on the type of repeaters that will only perform the delay correction if the input delay is less than the nominal delay δ.

Several possible implementations are possible for a repeater; some types of circuits can be distinguished:

- latch-based design, where the outputs are controlled by latches which are on the "data path"
- ME-based design where the MEs are on the data path
- ME-based design where the MEs are not on the data path

This work is supported by the EPSRC grant EP/C512812/1.

The designs can be implemented both at transistor-level and at gate-level. Some examples for the design styles will be analysed in the remainder of this section.

We introduce some measures to rank different designs: the *capture range* κ of the repeater is the set $\kappa = [\delta_{min}, \delta_{max}]$ of time separations at the input of the repeater that the device will be able to stretch back to the nominal value of δ. The *linearity* of the circuit corresponds to the linearity of the response of the design: given a range of time differences between two inputs, the linearity refers to the difference between the output delay between the signals and the input delay. The *latency* λ of the device is, intuitively, the time between the first input signal reaching the device and the corresponding output leaving the device. Finally, the *response time* ζ is the time between the first input reaching the device and the time limit before which the second event would not be delayed. For phase-locked loops (PLLs) it is customary to provide the capture range (or "lock-in" range) as the difference between the maximum and the minimum frequency the PLL will lock onto, as the set is centred around the natural frequency of the loop. Following the same custom, we similarly define $\kappa = \delta_{max} - \zeta$ (as $\delta_{min} = \zeta$).

Several implementations of the device are possible, each with different issues to be addressed. In particular, we can distinguish between "analogue" implementations of the repeater and "digital" implementations of the device; in particular, this distinction is based on the generation of the pulse which stops the late arriving signal from propagating to the output. The analogue implementations rely on analogue differentiators to generate the pulse; these differentiators are built using series capacitors. In the case of digital implementations the differentiators are implemented using gates. In this article we will focus in particular on some digital implementations of the repeaters.

In the remainder of the section, all the simulation results are normalised to FO4 delay. The experimental setup included input and output inverter buffers; the latency shown in the graphs includes these buffers. "Rise" and "fall" in the results refer to the inputs shown in the figures, e.g. after the input buffer, to simplify analysis.

II. Latch-based design

A conceptual idea of the device is given in Figure 1 a). The two input signals are fed into a pair of latches; the first input to propagate through one of the latches will cause a pulse at the "enable" input of the two latches, in turn causing the other signals to be prevented from propagating

through the latch for a given time. As can be seen from the simple example, two issues become of primary importance in the design of such devices: first, the latch of the signal arriving "late" should not enter metastability; second, which is correlated to the first, the time between an event triggering the pulse and the pulse reaching the "enable" input of the latches should be minimised in order to capture as many events as possible. In fact, if the latter condition does not hold, the late arriving signal will propagate through before the latch has been able to prevent this from happening.

The following description of the system is based on Figure 1 a), but can be adapted to other designs. Define as d_{setup} the setup delay of the latches and as d_q the delay $d \rightarrow q$ of the latch. From the figure, the letters a–d represent the path delays of the indicated paths; in particular, d represents the delay between the latch enable being released and the output being generated. Assume that two edges are travelling on the link and were generated with delay δ between each other and arrive at the input of the circuit with delay $\delta_{in} < \delta$.

The latency of the device is trivially $\lambda = d_q$. If

$$\delta_{in} < \lambda + a + b \tag{1}$$

the second edge will reach the output of the latch before the pulse is generated and therefore the circuit will have no effect. If

$$\delta_{in} \in [\lambda + a + b\,;\, \lambda + a + b + d_{setup}] \tag{2}$$

the latch may enter metastability, as the second edge will reach the latch during the time the pulse is being generated and will not respect the restriction imposed by the setup time of the latch. Finally, when

$$\delta_{in} > \lambda + a + b + d_{setup} \tag{3}$$

the latch will prevent the second edge to propagate until the pulse is completed.

The response time ζ will therefore be:

$$\zeta = \lambda + a + b + d_{setup} \tag{4}$$

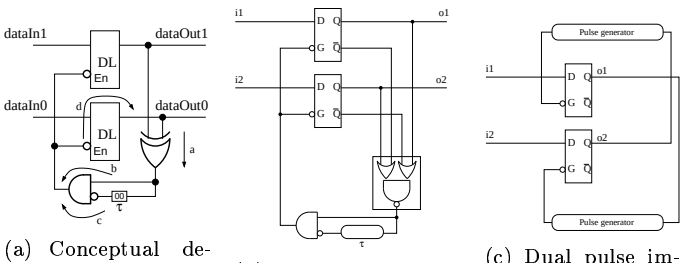
The nominal value of δ at the output will be equal to

$$\delta = a + \tau + c + d \tag{5}$$

The upper bound beyond which the device does not influence the path delays is going to be $\delta_{max} = \delta + \lambda$; in fact if $\delta_{in} > \delta + \lambda$ by the time the second event will occur the latch has been released. Note that if the input time separation is between δ and $\delta + \lambda$, the output will still be δ. The capture range κ will therefore be:

$$\kappa = \delta_{max} - \zeta = \tau + c + d - b - d_{setup} \tag{6}$$

These equations can be used to determine the value of τ given δ and a set of requirements. This first analysis shows that the value of δ must be chosen in such a way that, even when the phase is maximally corrupted, the condition



(a) Conceptual design

(b) Single pulse implementation

(c) Dual pulse implementation

Fig. 1. Latch-based implementations

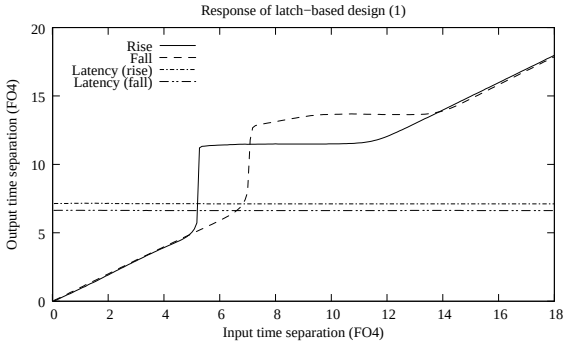


Fig. 2. Simulation results for the circuit in Figure 1 (b)

expressed in equation 3 is always respected. This can turn out to be a significant limitation to the speed of the circuit.

Based on Figure 1 (a) two possible implementations are obtainable, as in Figure 1 (b) and (c), according to the way the pulses which control the latch clock are generated. Figures 2 and 3 show the results obtained with these designs. The two designs have very similar responses; the expected output δ was around 12 FO4 delay, as this turned out to be the minimum time separation obtainable using a gate-level pulse generator and the available latches. The device responds correctly: when the input time separation drops below 12 FO4 the device pulls the edges apart so that the output time separation is preserved at the output. The design fails when the input time separation drops below a minimum value (equation 1). The latency of the device is around 6 FO4 delay for rising and falling edges. It is interesting that the rising and falling edges have different responses; this is due to the inimbalancebalance of the P- and N-type transistor in the gates available in the technol-

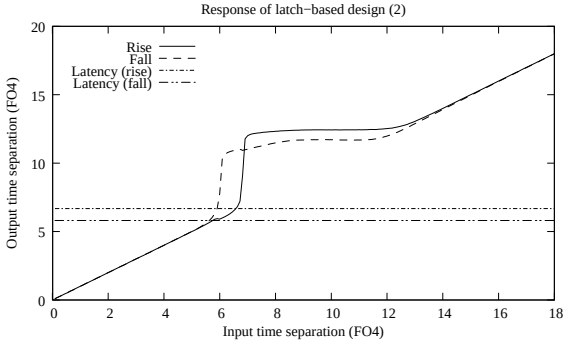


Fig. 3. Simulation results for the circuit in Figure 1 (c)

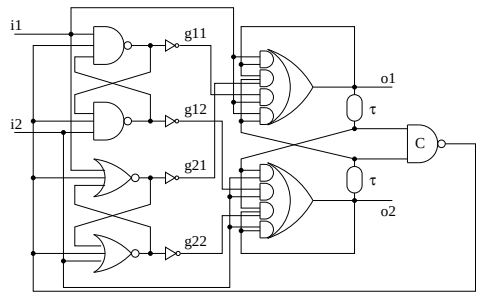


Fig. 4. Automatic synthesis design

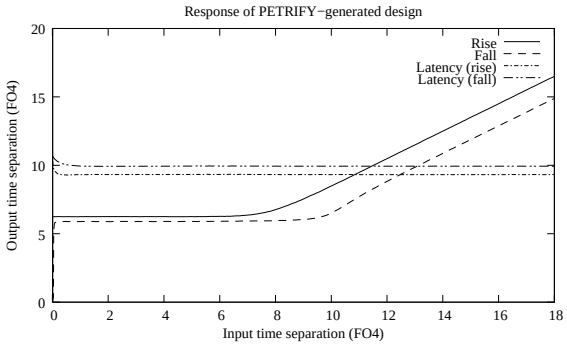


Fig. 5. ME-based design results for design in Figure 4

ogy employed to obtained the results.

For the purposes of "taxonomy" this type of design is "early-propagating", as an output is generated without waiting for the second edge to arrive.

The main advantages of this type of design is the relative simplicity of implementation. The gate count is relatively small and allows the repeater to be easily tuned to the required specifications. Also it would be relatively easy to scale up the design for multiple-rail implementations, simply by OR-ing all the XOR gates between pairs of wire and controlling a single pulse for all the latches on all wires. The latency of the design is also relatively low so that several repeaters can be placed along a line. Finally both design exhibit good linearity and the output curve are "flat" at the expected range. However, the capture range of the device is limited: both implementations stop working around 6 FO4 delay; around that point the risk of metastability in the latches increases until the devices fail. This type of design is therefore appropriate if the nominal δ of the link is in the order of tens of FO4. A faster design relies on transistor-level implementation; this will be described in Section IV.

III. ME-based designs

ME-based designs can be in turn divided into several types. An ME can be used to identify the first-occurring event and then, as both events have arrived, the signal is sent on with the correct time separation; otherwise the first output can be generated immediately after the first event arrives and the second after the second event, introducing some time separation if necessary; finally, the output of the ME can be fed to a wrapper logic to generate the outputs. The ME systems can be either "early propagating" if the first output is generated regardless of the second event at the input or "merging" if the outputs are generated only if both events have appeared at the input. In the first case the latency of the device will be reduced, while in the second case the latency of the device includes the input time separation. Conversely, in the first case if the input time separation is above the required δ, it is left untouched at the output; in the second case, the response is flat.

An example of "early propagating" design is 4. In this case two MEs are used for the rising (NAND) and falling (NOR) input transitions. The outputs of the MEs are fed to a control logic which takes care of the generation of the output given the ME outputs and the input signals. This control logic takes care of resetting the MEs after the output has been generated; therefore the next input must be generated only after the previous output has been produced. This introduces a limit for the bandwidth. Figure 5 shows the output response of the circuit. The latency of the design is of around 10 FO4, which increases slightly when the input time difference becomes small; this is due to the behaviour of the MEs (see Section **??**). The output δ was set to 6 FO4; the response is remarkably flat below 6 FO4. The design stops working below 0.04 FO4 when the NOR-based ME is used ("fall" curve in the graph); it works all the way down to 0.009 FO4 (simulation limit) when the NAND-based ME is used. One interesting feature of this design is that if the input δ is greater than the nominal value, the output time separation is less than the input. This is not always the case, as normally larger input delays are ignored by the circuitry. For the NOR-based ME branch this is particularly effective, as the response is flat between 10 and 0.04 FO4; for the other branch the limits are 8 and 0.009 FO4. Above the higher limit the design reduces the timing separation but not to the nominal value. The reduction, however, could allow the next stage to regenerate the nominal time separation.

An important point for this design is the fact that the control logic which takes the outputs of the MEs and the inputs and thus generates the outputs was obtained using automatic synthesis. The initial specification was described using an STG; this was then adjusted to take into account timing assumptions. The design was finally synthesised automatically using PETRIFY [1].

Finally, an example of "merging" design is shown in Figure 6. In this design two MEs are employed as in the previous case, but the output is generated when both inputs have arrived. This allows the circuit to introduce the expected delay regardless of the input time separation. Figure 7 shows the response of the device. It is remarkably "flat", in the sense that the time separation of the output signals is independent of the time separation of the input signals. Note that the capture range κ is infinite, as the upper bound $\overline{\delta}$ is infinite; however the response of the device is obviously limited by the metastability effect of the MEs. It can also be seen that the output time separation can be made very small: in this case it was only around 4 FO4, although it could have been arbitrarily smaller or larger. At very small delays ($\delta_{in} < 0.02$ FO4) the design

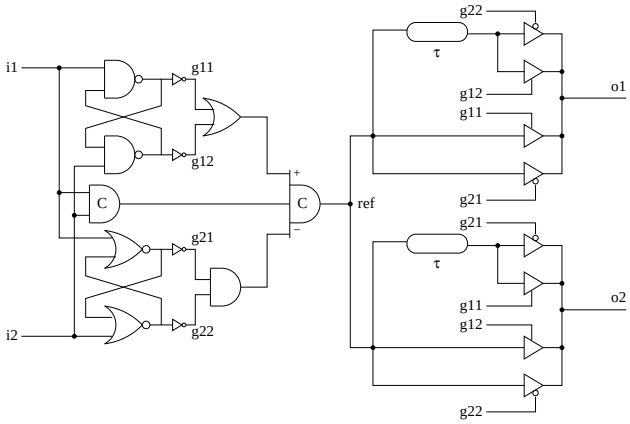


Fig. 6. ME-based implementation (merging)

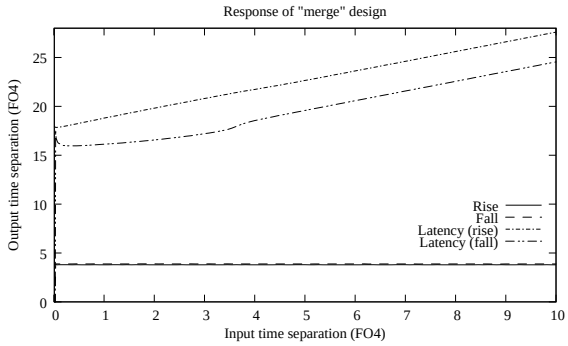


Fig. 7. ME-based design results for design in Figure 6

failed to regenerate the delay as the output value was different from the input. This is due to minute imbalances in the load of the input signal and of the output of the MEs, which lead to errors. This is the case only in the NOR-based ME: the NAND-based ME continues working throughout the required range, down to 0.009 FO4.

Another unwanted behaviour of this design is that the latency of the device increases with the input time difference. This is due to the "merging" characteristics of this design: as the production of the output is dependent on the presence of both input transitions the latency is equal to:

$$\lambda = \delta_{in} + d_C \tag{7}$$

where d_C is the propagation delay of the logic which generates the "reference" signal. An important note though refers to the change in behaviour of the latency for the falling transitions which occur around 3.5 FO4 input time separation. this is due to the slow response of the NOR-based ME; below 3.5 FO4 the ME delay dominates, so that the reduction in input delay is counteracted by an increase in ME resolution time, which causes the output delay to flatten out below this point.

IV. Transistor-level design

The latch-based design is improved if the response time of the circuit is reduced. This can be achieved using transistor-level implementations of the various parts of the repeater. One such solution is shown in Figure 8. The

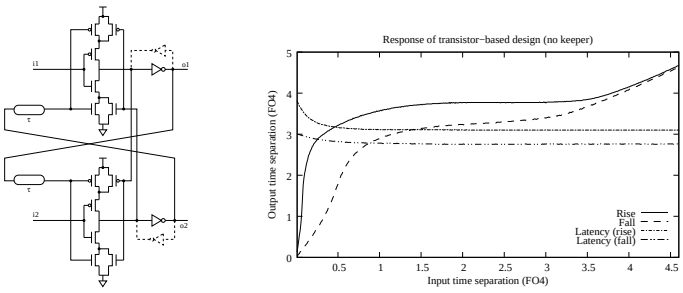


Fig. 8. Transistor-level design and simulation results

circuit shows smaller response time than the designs previously analysed, and also a smaller number of gates, resulting in less area consumption.

The response of the design is strongly dependent on the direction of the input transitions, due to the difference in speed between the P- and N-type transistors. Taking into account both transitions, the latency of the device is around 1 FO4 if the input and output buffers (not shown) are not taken into account; the response time ζ is also around 1 FO4. Intuitively, when N-type transistors dominate the response time is less than 0.5 FO4. The capture range κ is around 2.5 FO4; however τ was chosen to be only 2 FO4, but could have been a larger value, in turn increasing the capture range.

Note that the latency of the device increase exponentially when the input time separation drops, due to metastability of the device. However, in that region the device is already outputting a δ which is less than the nominal value. In this example, the operating area of the device would have been down to 1 FO4.

V. Conclusions

An introduction to the problem of designing phase-alignment circuitry within the context of phase encoding has been presented. Some designs and categories have been described and simulation results presented. The availability of different solutions with different dynamic behaviour allow the designer to identify the best design style according to the particular design requirements of the work at hand. Silicon results, preceded by a more thorough analysis of the layout information, are part of the future work, together with a method to explore the design space in an interactive manner.

References

[1] Jordi Cortadella, Michael Kishinevsky, Alex Kondratyev, Luciano Lavagno, and Alexandre Yakovlev. Petrify: a tool for manipulating concurrent specifications and synthesis of asynchronous controllers. In *XI Conference on Design of Integrated Circuits and Systems*, Barcelona, November 1996.

[2] C. D'Alessandro, D. Shang, A. Bystrov, and A. Yakovlev. PSK Signalling on SoC Buses. In *Integrated Circuit and System Design. Power and Timing Modeling, Optimization and Simulation. Proceedings of PATMOS 2005*, volume 3728 of *Lecture Notes in Computer Science*. Springer, 2005.

[3] Crescenzo D'Alessandro, Delong Shang, Alex Bystrov, Alex Yakovlev, and Oleg Maevsky. Multiple-Rail Phase-Encoding for NoC. In *Asynchronous Circuits and Systems, 2006. 12th IEEE International Symposium on.*

Fault Tolerant Techniques to Minimise the Impact of Crosstalk on Phase Encoding Communication Channels

Basel Halak, Alex Yakovlev, University of Newcastle upon Tyne, UK

Abstract

A communication scheme in which symbols are encoded by means of phase difference between transitions of signals on parallel wires is considered. A significant decrease in the reliability of such a channel is caused by capacitive crosstalk between adjacent wires. A more robust high-speed phase encoded channel can be designed by minimising its vulnerability to crosstalk noise. This paper investigates the impact of crosstalk on phase encoded transmission channels. A functional fault model is presented to formulate the problem. Three fault tolerant schemes are introduced which are based on information redundancy techniques and the partial order coding concept. These schemes are simulated with CADAECE using AMS CMOS 0.35µm process. Area overheads, performance and fault tolerant capability of those methods are compared. It is shown that a substantial improvement in the performance can be obtained for four wire channels when using the fault tolerant design approach, at the cost of 25% of information capacity per symbol.

Introduction

On-chip global buses are increasing in length with increasing die sizes, resulting in large propagation delays [1], [3]. The delays of those buses have two impacts; namely:

- They limit the system performance in many high-speed microprocessors [2], [3].
- They lead to an increase in the clock skew, which makes it difficult to accurately distribute a single global clock across the entire system [4].

This trend is anticipated to exacerbate in the future due to the increasing gap between gate delays and interconnect delay brought about by shrinking feature sizes [5]. Globally asynchronous locally synchronous (GALS) electronic system design is a methodology that addresses these problems. In GALS functional modules are designed using conventional design techniques. Each module is complemented by its own local clock generator and a self-timed wrapper that enables the modules to communicate using asynchronous handshake protocols [7]. It is expected that 40% of the electronic design in 2020 will be driven by handshake clocking [6]. Although the issue of self-timed communication protocols has been intensively investigated in terms of power efficiency, speed, area overheads and reliability in many papers, the effect of transient errors on self-timed channel reliability was first addressed in [8]. Transient errors caused by cross-talk, cross-coupling, ground bounce or environment interference, become more prominent as integration increases. Thus signal integrity is put at risk [9]. This motivates the fault tolerance approach to design. Multi-rail phase encoding is a novel fault tolerant, self-timed signalling protocol, it was proposed in [8]. The data is sent using the phase relationship between differentially delayed copies of a reference signal. Mutual Exclusion elements [10] are employed as phase detectors for data recovery. This technique outperforms many of the existing self-timed communication methods such as 1 of 4 RTZ (Return to Zero). It also exhibits high robustness against Single-Event Upsets (SEUs), hence it is more reliable.

However, capacitive crosstalk between adjacent wires may deteriorate their phases, which will strongly affect the integrity of the data being sent [8]. This problem can create a bottleneck to the system and may prevent the use of this technique to send data on long wires and/or at high frequencies. Crosstalk can be defined as a disturbance, caused by electromagnetic interference, along a circuit or a cable pair. A telecommunication signal disrupts a signal in an adjacent circuit or wire and can cause the signals to become confused and cross over each other.

In the case of adjacent wires the crosstalk noise effect can be explained as follows; when two wires are placed close together, the current flowing down one (which we will call “aggressor”) induces a current in the other wire “victim”. The electric field causes a current in the victim that flows both ways, backwards and forwards. For example if a single electron was at a point along the aggressor, it will tend to repel electrons in the victim in both directions away from that point. This type of coupling is often called “capacitive” coupling. The aggressor wire also generates a magnetic field, which in turn generates a current in the reverse or backward direction in the victim wire. This type of coupling is often called “inductive” coupling. So crosstalk is a direct result of the electromagnetic field radiated from the aggressor wire, therefore its coupling effects attenuate with distance. This means the crosstalk between non-neighbouring wires is less than that between neighbours. This fact is going to be exploited in the fault tolerant techniques introduced in this article.

There are two types of Crosstalk noise: Functional Noise and Delay Noise. A functional noise occurs when a transition on a wire (A) leads to a glitch on a neighbouring wire (B) as shown in figure 1-a. A delay noise occurs when two neighbouring wires switch simultaneously, which causes transition slowdown or speedup in the victim wire. This leads in both cases to a reduction in the original time distance between the two transitions as can be seen in figure 1-b.

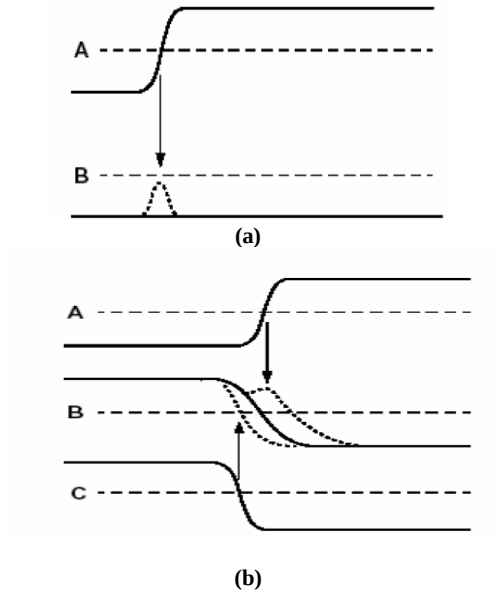


Fig. 1: Crosstalk Noise Types

In the case of phase encoded channels all wires switch close in time and in the same direction, they can be considered allies, i.e. they reinforce each other. This reinforcement is manifested as a reduction in the original phase between transitions. Figure 2 shows the transient response of the inputs (2-a) and the outputs (2-b) of a four wire phase encoded channel whose length is 2mm. As can be clearly seen the time distance 500ps between the first and the second transition (in1 & in2) was reduced at the outputs (out1 & out2) to 230ps. This also applies for (in3 & in4). This phase corruption does not generate errors as long as the mutex elements at the receiver side can decide which transition was the first one. However, our simulations showed that the crosstalk noise can in some cases lead

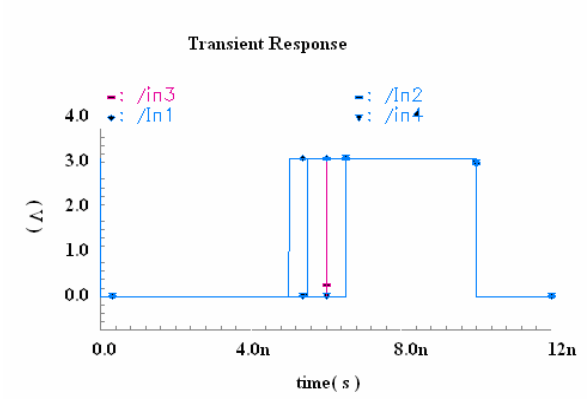
to phase conversion i.e. the transitions are received in an order different to their original one. Errors can also occur if glitches (see figure 1) are perceived as transitions by the mutex elements. These errors are filtered out if they happen outside the event window; otherwise they cause faults [8]. This problem is fatal to the multi-phase encoding technique.

The impact of crosstalk noise on communication channels has been addressed in many papers. Researchers have proposed several techniques aimed at reducing the crosstalk induced delays. The insertion of repeaters and shielding of bus wires are the most common methods. The selection of a proper global wire configuration has also been proved to significantly minimise the impact of crosstalk. Furthermore, research has recently shown that fault tolerant techniques can be employed to increase communication channels reliability in the presence of crosstalk. Other techniques rely on crosstalk avoidance codes .

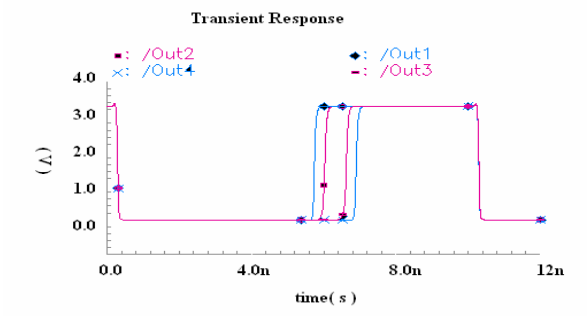
This article provides a comprehensive study on the use fault tolerant techniques to minimise the effect of crosstalk noise on multi-phase transmission protocol.

For deep submicron circuits the capacitive coupling is more prevalent, and the delay is dominated by the capacitance and the resistance [11]. Therefore our focus will be only on the impact of capacitive crosstalk.

The organisation of the paper is as follows, Section 2 introduces a functional fault model which formulates the impact of crosstalk noise on multi-phase transmission protocol. The background theory, implementation and simulation results of each technique are presented and analysed in section 3. Finally the conclusions are drawn in section.



(a)



(b)

Fig. 2: Crosstalk Effect on Four Wire Phase Encoded Channel

2. Transient Fault Model

As explained previously the information is encoded as a differential phase between signal transitions on wires. The crosstalk noise may corrupt this phase leading to systems failure. In order to tackle this problem, a fault model based on a functional abstraction of the crosstalk errors is introduced in this section. It consists of three types of faults, namely:

- Type 1 is a result of the conversion of phase between transitions on adjacent wires.
- Type 2 is a result of the phase conversion between transitions on non-adjacent wires
- Type 3 which includes both type 1 and 2, i.e. it is a result of the phase conversion between transitions on all wires.

Consider the case of a four wire phase encoded channel. The first wire is denoted *a*. The second is *b*, the third is *c* and the forth is *d*. Each combination of transitions (information symbol) will be represented by those four letters whose order indicates the order of signal transitions in time. For example *badc* means the second wire *b* transition first then the first wire *a* follows, and then the fourth wire *d*. Finally the last transition occurs on the third wire *c*. This notation will be used throughout the article.

Let us now assume that the combination *bacd* was sent. When the combination *bacd* is received, i.e. the transition order was preserved, no errors are said to have occurred. If the combination *abcd* is received, type 1 fault is said to have occurred, i.e. a conversion in phase between transitions on two adjacent wires (a, b). If the combination *bcad* is received, a type 2 fault is said to have occurred, i.e. a conversion in phase between transitions on two non-adjacent wires (a, c). Finally, if the combination *cbad* is received, a type 3 fault is said to have occurred, i.e. a conversion in phase between transitions on two adjacent wires (b, c) and on two non-adjacent wires (a, c).

3. Fault Tolerance Techniques

In this section three fault tolerance techniques for phase encoding transmission protocol are introduced. The first one is based on the traditional theory of concurrent error detection and correction [12]. The second one is based on the concept of partial order coding. The essence of the third technique is to encode each data bit as phase between signals on two particular non-adjacent wires. This allows the detection or detection and correction of type 2 faults.

3.1. Cluster-based Concurrent Error Detection and/or Correction Techniques (CCED)

The essence of this method is to map the normal output vector space of a system onto an extended code space such that only a subset of the code space represents valid information. This mapping can be obtained by adding extra bits, which are called check bits, to the data word to form a codeword which has useful error detection or detection and correction properties. Our simulations showed that the occurrence of type 1 faults is more frequent than type 2 and/or type 3 ones, therefore it was decided that the phase encoder should be designed in such a way that phase conversion between any two signals on adjacent wires should only cause one bit error. This reduces the effect of the crosstalk noise on the data integrity. For example, let us study the case of a 4 wire phase encoded channel. Assume that the data word (0000) was phase encoded as (abcd) and during its transmission was altered to (abdc). Here we have two possibilities. The first one is when the hamming distance between the two original codes is equal to one, e.g. the code of (abdc) represents (0001), (0010), (0100) or (1000). The second possibility is when the hamming distance between the two codes is more than one, e.g. (abdc) code is (0111). As can be clearly noticed type 1 faults causes one bit error in the first case and 3 bit error in the second case. The detection or/and correction of such errors requires adding extra testing circuitry to the design. We have designed two schemes. The first one has error detection properties. The second one has error correction properties. These circuits were simulated in CADENCE and simulated using AMS CMOS 0.35µm process. The simulation showed that both schemes function correctly. All type 1 fault has been detected and/or corrected.

3.2 Cluster-based Partial Coding Technique(CPC)

Although CCED method improves the reliability of the channel, it has large area overheads. CPC is another approach based on partial order coding concept; it has the same fault tolerance ability but requires less area overheads. The idea of this method is to encode the data by means of phase difference between transitions of signals

on non-adjacent wires so that the phase between any two signal on neighbouring wires does not carry any information. The theory behind this method is that the crosstalk between signals on non-adjacent wires is less than it is between signals on adjacent wires because of the fact that capacitive coupling effects attenuate with distance. This partial order coding masks type 1 fault and does not require any additional hardware. However less number of bits can be sent as not all combinations can be used. The circuitry of this method has also been designed and simulated and proved correct.

3.3. Direct Mapping Technique (DM)

This method consists of encoding each data bit as phase difference between signals on two particular non-adjacent wires as illustrated in table 1.

Data	Wires
I1	a & c
I2	a & d
I3	b & d

Table 1: Direct Mapping for 4 wire phase Encoded Channel

Direct mapping masks all type 1 faults as phase between adjacent wires does not carry any information. It also allows the detection and/or the correction of type 2 faults if extra testing circuitry is included in the design. This technique was implemented on a four wire phase encoded channel in order to detect single type 2 error. DM circuitry has been designed in CADENCE and simulated using AMS CMOS 0.35µm process. Its functionality has been verified by applying all possible combinations on the encoder and receiving them correctly at the decoder.

Table 2 shows a comparison between the three mentioned methods in terms of information capacity (column 3 and 4), extra hardware (column 5), fault tolerance capability (column 5), and its effect on performance (column 7) and on the design (column 8). The first column represents the number of wires in the phase encoded channel. The second column represents the method of fault tolerant design.

NO. of Wires	Method	Symbols	No of bits	Additional hardware for Testing circuitry	Fault Tolerance Capability	Effect On Performance	Effect on The Design
4	CCED	24	4	Yes	Type 1 Detection	None	None
	CPC	8	3	No	Type 1 Masking	Three Times Improvement is possible	None
	DM	8	3	Yes	Type 1 Masking Type 2 Detection	Three Times Improvement is possible	The receiver requires 90% less
5	CCED	120	6	Yes	Type 1 Detection	None	None
	CPC	42	5	No	Type 1 Masking	Improves Performance	None
	DM	16	4	Yes	Type 1 Masking Type 2 Detection	Improves Performance	The receiver requires 97 % less area
6	CCED	720	9	Yes	Type 1 Detection	None	None
	CPC	258	8	No	Type 1 Masking	Improves Performance	None
	DM	32	5	Yes	Type 1 Masking Type 2 Detection	Improves Performance	The receiver requires 99% less

Table 2: The Characteristics of CCED, CPC and DM Techniques

Conclusions

The phase encoded data integrity is put at risk by crosstalk-related delays. The phase between transitions deteriorates in long parallel wires due to coupling capacitance which depends on wire length and the operating frequency. This deterioration may lead to errors i.e. phase conversion, hence to system failure. The simulation showed that crosstalk related errors become prohibitively large for wire longer than 2 mm and at frequencies higher than 0.6 GHz. Therefore faults tolerant techniques are a necessity in order to allow the use of this communication protocol reliably. A fault model has been introduced to formalise the crosstalk problem. Three methods have also been presented. All of them are based on the partial order coding concept. The CCED technique allows the detection of type 1 faults. The CPC method masks type 1 faults. It also leads to a significant improvement in the channel performance. This compensates for the reduction in the number of symbols that can be sent. The DM technique allows the detection of type 2 faults. It also masks type 1 faults. Improvement in channel speed is also possible. The DM method significantly simplifies the design of the phase decoder and reduces its area, which facilitates the implementation of phase encoded transmission protocol on a higher number of wires.

References

[1] F. Caignet, S. Delmas-Bendhia, and E. Sicard, "The challenge of signal integrity in deep-sub micrometer CMOS technology," *Proc. IEEE*, vol. 89, pp. 490–504, 2001.

[2] D. Sylvester and C. Hu, "Analytical modelling and characterization of deep-sub micrometer interconnect," *Proc. IEEE*, vol. 89, pp. 634–664, May. 2001.

[3] D. Pamunuwa, L.R. Zheng, and H. Tenhunen, "Maximizing throughput over parallel wire structures in the deep sub micrometer regime," *IEEE Trans. VLSI Systems*, vol. 11, pp. 224–243, Apr. 2003.

[4] N. A Kurd, '' Multi-GHz Clocking Schemes for Intel Pentium 4 Microprocessors '' Proc. ISSCC 2001 Feb 2001 pp 404-405.

[5] Semiconductor Industry Association, International Technology Roadmap for Semiconductors (ITRS) 2003, http://public.itrs.net.

[6] Semiconductor Industry Association, International Technology Roadmap for Semiconductors (ITRS) 2005, http://public.itrs.net.

[7] D. Chapiro, Globally-Asynchronous Locally-Synchronous Systems, Ph.D. Thesis, Stanford University, 1984.

[8] C. D'Alessandro, D. Shang, A. Bystrov, A. Yakovlev and Oleg Maevsky, "Multiple-Rail Phase-Encoding for NoC", In Proceedings.12th International Symposium on Asynchronous Circuits and Systems, IEEE, pages 107–116, March 2006.

[9] M. Nicolaidis Eric Dupont and Peter Rohr," Embedded robustness IPs", In Proceedings of the 2002 Design, Automation and Test in Europe Conference and Exhibition (DATE'02), 2002.

[10] C. Molnar and I. Jones," Simple circuits that work for complicated reasons", In Proceedings Sixth International Symposium on Asynchronous Circuits and Systems, volume 1, pages 138–149, IEEE CS, April 2000.

[11] D. Pamunuwa, L.R. Zheng and H. Tenhunen, " Maximising Troughput Over Parallel Wire structures in Deep Submicron Regime", IEEE Trans. VLSI Systems, vol 11, no 2, Apr. 2003, pp. 224-243.

[12] G. Russell and Ian L. Sayers, Advanced Simulation and Test Methodologies for VLSI Design, London, 1989.

C-element Latch Scheme with Improved Fault Tolerance

K. T. Gardiner and A. Yakovlev
University of Newcastle,UK
{k.t.gardiner,a.yakovlev}@ncl.ac.uk

***Abstract*— This paper considers the propagation of transient faults in dual-rail combinational logic through C-element latches. The paper considers which faults may propagate in a standard latching structure. The method of duplication is described. A new latching scheme is then introduced and compared with previous schemes.**

I. Introduction

Transient faults in electronic circuits were previously only of concern in space and high altitude applications. As fabrication technology sizes have become smaller these faults have become a concern in terrestrial level applications [2]. The International Technology Roadmap for Semiconductor (ITRS) [1] also highlights transient faults as an increasing problem. This has caused an increase in research in this area in the last few years with work being carried out to analyse a circuit's susceptibility to a fault and to produce fault tolerance techniques. Only a small fraction of this work has focused on asynchronous circuits, such as: [3].

This paper addresses the issue of transient faults which affect dual-rail combinational logic of an asynchronous circuit and their propagation through a C-element latch. The scheme introduced exploits the build-in redundancy of dual-rail circuits to detect faults and stop them from being present at the output of the latches.

The rest of the paper is structured as follows: Section 2 introduces asynchronous circuits, the assumptions made in this work and muller pipelines; in Section 3 the method of duplication is described, Section 4 introduces the new latching scheme, Section 5 shows the results of several benchmarks, Section 6 concludes the paper and describes future directions of work.

II. Background

A. Asynchronous Circuits

Asynchronous circuits have no global clock to provide synchronisation, instead methods such as matched delay and completion detection can be used to mark the completion of an operation. Data can be passed between modules by means of handshaking. The ITRS [1] sees asynchronous circuit techniques at global or local level as a possible solution in future technologies for the problems of communication, robustness and power consumption.

B. Dual-Rail

One way of representing data in asynchronous circuits is to use dual-rail encoding. Here two lines are used to represent each bit as shown Table I. Alternating spacer and code-word states (logic 0 and logic 1) are passed through the circuit's logic. The current state is detected by completion detection.

dt	df	State
0	0	Spacer
0	1	code-word - Logic 0
1	0	code-word - Logic 1
1	1	Not used

TABLE I: Dual-Rail encoding

C. C-Element

The C-Element shown in Figure 1 is a common component in asynchronous circuits. A C-Element with inputs A and B and output C has the following behaviour: C = AB + C(A+B). Therefore the output changes only when the inputs are the same state, at which point the output changes to that state.

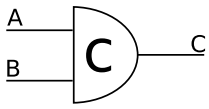


Fig. 1: C-Element

D. Muller Pipelines

Muller Pipelines [4] form the basis of the control circuits in many asynchronous circuits. Figure 2 shows a single stage Dual-Rail Muller Pipeline. The pipeline works as follows, the output (dto,dfo) holds a spacer causing acko to be logic 1. This leads to the previous stage latching data and the next stage to setting acki to logic 1. When a code-word appears at the input (dti,dfi) it is latched. This causes acko to change to logic 0 leading to the previous stage latching spacer. The spacer is presented to the input of the latch and is latched when acki changes to logic 0. The spacer moves to the latches outputs causing acko to change to logic 1.

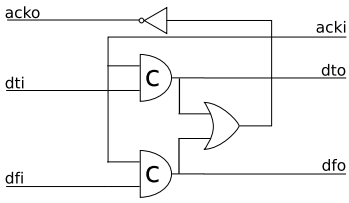


Fig. 2: Dual-Rail Muller Pipeline Stage

E. Analysis of Faults Propagation

To asses how prone to transient fault propagation the latch is each input combination must be considered to see if a fault will propagate. Consider the single stage from Figure 2. If dti,dfi are 1,0 and acki transitions to 1 the value 1,0 will appear at the stage's output. If before acki goes to 0 dfi goes to 1. The output will change to 1,1. The erroneous 1 value will remain till the following spacer state arrives at the latch due to the function of the C-element.

F. Assumptions

The following assumptions are made with regard to this work:

1) Transient faults are considered as a digital pulse
2) One fault occurs and the circuit returns to normal operation before the next fault occurs
3) A fault can not cause a change in value of a data state i.e. 01 to 10
4) When a dual-rail channel changes from spacer to data to 11 error. The data state is long enough to be recognised and trigger latching.

III. Duplication Method

The simplest method of hardening is the duplication of the circuit. Here the combinational logic is duplicated and the C-elements modified to have inputs from both the combinational logic blocks. Figure 3 shows an architectural level diagram of this scheme, Figure 4 shows the latches used in this scheme. Only when both the data inputs match is a code-word / spacer latched. This scheme has the advantage of simplicity with no extra control logic needed. But has a significant overhead in terms of area. Duplication is tolerant to single faults and has only a small increase in latency caused by extra inputs to the C-elements.

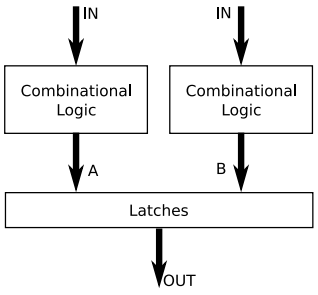


Fig. 3: Duplication Architecture

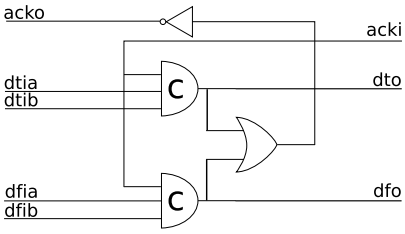


Fig. 4: Duplication Latch

State	Signal								
	dti	dfi	en1	dt1	df1	en2	dto	dfo	done
1	0	0	1	0	0	0	0	0	1
2	1	0	1	1	0	0	0	0	1
3	1	0	0	1	0	0	0	0	1
4	1	0	0	1	0	1	1	0	0
5	0	0	0	0	0	1	1	0	0
6	0	0	0	0	0	0	0	0	1
7	0	0	1	0	0	0	0	0	1

/home/a4706838/

TABLE II: Signal States

IV. Double Latch Scheme

From the transient fault analysis view point the Muller Pipeline has two features which give poor fault tolerance. Firstly, the behaviour of the C-element means that a transient fault is latched any time the enable signal is in a state which allows it. Secondly, any erroneous change in the output of the latch is presented straight away to the follow stage. A solution is to latch the data and then check it validity. A second latch is needed to stop the data being presented to the next stage until it is considered correct.

The proposed latch is shown in Figure 5 the Signal Transition Graph (STG) for the control circuit is shown in Figure 6. The first latch, L1, is composed of two C-elements L1a and L1b with a reset input that forces the output of the C-element to logic 0. The second latch, L2, is made up of two standard C-Elements L2a and L2b. Between the two latches are a NOR gate to detect spacer and a XOR gate to detect data. The XOR gate is used as the OR gate usually used to detect data in dual-rail circuits will detect the 11-state as a valid code-word. After the second latch is a NOR gate to produce a done signal which also doubles up as the acko signal. Table II shows the state of each signal for the cycle of latching data and spacer. The table shows the inputs, outputs, internal data signals and done signal. The data and spacer signals are not shown and can be easily derived form the internal data signals. The table shows the state of the latch as it operates in a fault free condition. The enable signals must pass through these states to ensure a fault does not propagate through the latching element.

A. Operation

In fault free conditions the latch operates as follows: Initially the latch holds a spacer and EN1 is high ready to receive data. When data is detected EN1 falls low before EN2 two goes high to latch the data in the second latch, followed by done going low. The change in done causes a spacer state to arrive at the latches input which passes through the first latch. EN2 goes

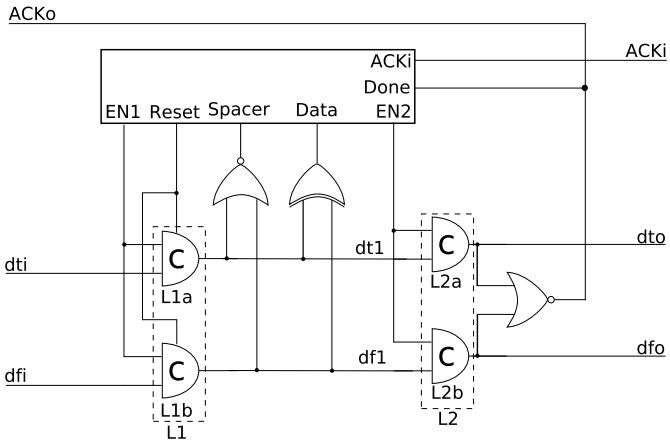


Fig. 5: Double C-element latch

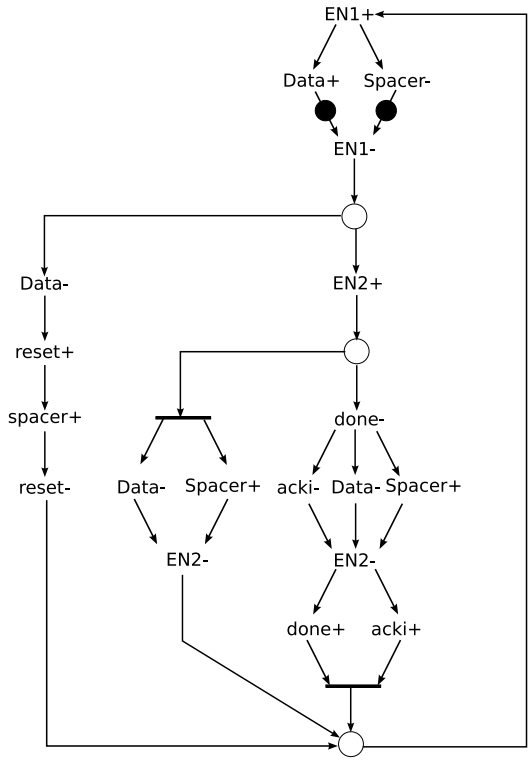


Fig. 6: Double C-element latch control STG

low allows the spacer state to pass, causing done to go high. This in turn causes EN1 to go high and for data to arrive at the input from the previous stage.

Of course there is the possibility that the latching scheme itself introduces more possibilities for a fault to propagate. Indeed as the discussion below shows this does indeed happen but does not cause a visible error. To consider the fault tolerance of the scheme the effect of a fault in each case shown in Table II must be examined. This table allows states in which the latch may pass a fault to be identified. In state 2 the fault dti=1, dfi=1 can be passed to the internal state this fault is referred to as F1 and is the same fault as passed by the standard scheme. In states 3 and 4 a spacer state my be passed to the output, named F2 and F3 respectively. In the case of F1 the latch L1 is reset and the data state is re-sampled till a valid data state is latched. In the case of F2 and F3. The spacer fault passes to the output of the latch but at this point the output is in a spacer state so no change occurs. To remedy this EN2 goes low followed by the data state being re-sampled.

B. Timing Assumptions

This scheme has a number of timing assumptions which must be observed. The time before the data of dti,dfi is evaluated for a fault must be long enough for the outputs of latch L1 to settle after latching. The time between EN1- and EN2+ must be long enough to ensure EN1 is low before EN2 goes high. If both EN1 and EN2 are high at the same time a 11 fault may propagate through the latch into the next stage. All these delay must of be set after layout so the path delays can be taken into account and must include an allowance for process variability effects.

V. Comparison

The three schemes above were compared in terms of overhead when applied to a AES look-up S-Box and a Kasumi S9 S-Box. The results are shown in Table III.

Scheme	**Area**			
	AES (μm^2)	Overhead (%)	Kasumi (μm^2)	Overhead (%)
Muller	119519	N/A	78114	N/A
Duplication	237929	99	155027	98.4
Double C-Element	128692	7.67	88088	12.7

TABLE III: Area Comparsion

The overhead of duplication is high as expect. In both cases the overhead is not 100% as the acki signal was not duplicated. The overhead for the proposed method is low.

VI. Conclusion

A new scheme to stop the propagation of faults through C-Element latches was presented and compared to the method of duplication. The size of the overhead of the new method was found to be small. It is planned to expand the benchmarks to include a more S-Box designs and to compare the schemes in terms of power and latency with and without the existence of a fault. The new method will also be compared with that proposed by Monnet et. al. in [3]. The latency metric is particularly important in order to measure the effect of needing to wait before assessing the validity of input that has been latch into latch L1 and the extra time need to re-sample the inputs when a fault occurs.

References

[1] International technology roadmap on semicondutors. 2005.

[2] P. Hazucha and C. Svensson. Impact of CMOS technology scaling on the atmospheric neutron soft error rate. *IEEE Transactions on Nuclear Science*, 47(6):2586–2594, 2000.

[3] Y. Monnet, M. Renaudin, and R. Leveugle. Asynchronous circuits transient faults sensitivity evaluation. In *11th IEEE International On-Line Testing Symposium*, 2005.

[4] Jens Sparsø and Steve Furber, editors. *Principles of Asynchronous Circuit Design: A Systems Perspective*. 2001.

An Information Redundant Asynchronous Concurrent Error Detecting ALU

M.J. Marshall, G. Russell
School of Electrical, Electronic & Computer Engineering
University of Newcastle upon Tyne, UK
m.j.marshall@ncl.ac.uk, g.russell@ncl.ac.uk

ABSTRACT

As a result of advances in technology and shrinking device dimensions, the occurrence of transient errors is increasing. This together with the concomitant reduction in supply voltages has decreased noise margins, causing system reliability to be reduced, all this at a time when electronic systems are being used increasingly in 'safety critical' applications.
Previous work has demonstrated that an information redundant Concurrent Error Detection (CED) scheme using Dong's Code can be applied efficiently to an ALU within an asynchronous design in which the area overheads incurred were approximately 12% [1]. This paper extends the work to the application of this checking scheme to the integration of a multiplier function into an ALU enabling the advantages of using asynchronous design style, for example power reduction, to be used in a wider arena of DSP applications. The work outlined in this paper has demonstrated that up to a 24% area saving can be made in comparison with similar CED scheme on the multiplier function, and with many parts of the circuit used within other processor functions, large savings are found on the ALU as a whole. This has shown that despite the complexity of multiplier structures, an overhead of 13% may be obtained when implementing an asynchronous self checking ALU with multiplier functionality as opposed to a system without any CED.

1. Introduction

With the ever increasing need for lower power, faster, more efficient designs there is much attention focused on the potential of asynchronous design. However, these do exhibit the unfortunate characteristic of higher error probabilities due to the lack of a global timing clock which leads to incorrect state transitions within both fully and locally asynchronous applications. The nature of transient faults (short duration and random occurrence), render them undetectable by standard test strategies using BIST or scanpath which are only applied periodically. In order to detect transient or intermittent faults, which can manifest themselves as 'soft errors' and 'silent' data corruption, it is necessary to implement some form of Concurrent Error Detection. This allows functional blocks to be tested concurrently within their normal operation, creating a fault tolerant system. With long pipelined functions such as those within DSP structures, the opportunity for data corruption increases as the number of functions increases. Without error checking within the pipeline, silent data corruption may occur, leading to incorrect results.
Many different methods for test and detection of faults within systems have been implemented, ranging in their complexity, cost of manufacture and fault and error coverage. Hardware, time or information redundancy methods may be used, each with its own advantages and disadvantages. Implementation of time redundancy leads to time penalties as operations will take much longer, due to re-computation of results. The cost for a hardware redundant system often outweighs that of the information or time redundant systems making hardware redundancy less popular with chip producers unless extremely high error checking and/or reliability is required. Information redundancy is more often chosen over time redundancy due to the higher speed of operation obtainable.
Unfortunately there is no one way of providing the information redundancy and many chip makers implement their own proprietary designs or use existing Intellectual Property (IP) designs which often lead to poor error coverage and the use of multiple codes within a single chip, leading to inefficiencies, for example in increased area overheads due to the need for code conversions and thus an increased probability of failure due to the larger area.
There are several types of information redundancy codes which are commonly used, namely Berger, Residue, Parity and Dong's Code, all providing error coverage suitable for certain implementations. With the exception of Dong's Code all of the codes mentioned above have a set of arithmetic and logical prediction equations including the functionality of a multiplier. Dong's Code has predictive equations for several ALU functions such as add/subtract and the logical functions rotate/shift [2] but lacks multiplication protection.

This paper studies the implementation of a predictive scheme to allow detection of errors during an asynchronous multiplicative operation on a set of binary numbers using the information redundant code, known as Dong's Code.

2. Berger and Dong's Code

Dong's Code is a modified Berger Code, as such it exhibits some similar characteristics, for example, both are separable codes [3]. This allows the data bits and check bits to be clearly identified and removed if necessary without decryption. Although separable codes do increase the number of bits required to represent an output and thus more storage space to represent the

output value; it has the advantage that some area overheads may be reduced as no encoding and decoding circuit is required. In using Dong's Code a much greater control over the level of error and fault coverage can be obtained, permitting it to be tailored to a given application. Berger Codes, by definition, require $n = \lceil \log_2(m+1) \rceil$ bits for the code, i.e. for a $m = 16$ bit informational word, 5 bits are required. Dong's code is in fact made up of 2 codes, C1 and C2, C1 being the count of the number of zeroes mod (2^k) and C2 being the number of zeroes in C1 mod(2^k). This provides a check on the check bits themselves. The completed Check Symbol Sc, is thus C1+C2 and requires $n = k + \lceil \log_2 k \rceil$ bits, where k is a positive integer value. i.e. for a 3 bit code, 5 bits are required in total. By increasing the value of 'k', the error coverage may be increased, this also increases the area overhead by doing so.

In this paper for the 8 bit implementation of a 2 operand multiplier with 16 bit output and length ($k = 3$) in the first part of the code (C1), and thus 2 bits for the second part (C2) to represent the number of zeroes within C1. This will provide 99.04% error coverage [4], hence will be suitable for many medium error tolerant systems such as a data logging DSP processor.
Initially it may be considered that Dong's Code seems to have no advantages, it would require 3 bits to represent the information data (C1) and then a further 2 bits for the second part of the code (C2). Berger's Code would require 5 bits in total to represent the maximum of 16 zeroes. However, the system savings are later realised through minimisation of the prediction scheme.

Example: X=1100110011111111
Berger Code: Xc(number of zeroes) = 4, so Sc= b00100
Dong's Code: Xc=4, so C1=4Mod(8)= b100,
C2=b10, Sc=C1+C2=b10010

4. Dong's Code Check Symbol Prediction

In order to implement a CED scheme it is necessary to compare the calculated code with some predicted value so that errors may be detected, this is done through a small degree of parallelism. During the calculation of the product a Check Symbol Prediction (CSP) calculation is used to determine the output of the Check Symbol Generation (CSG).
It has been shown [4] that the check bit prediction of Berger Code for the multiplication of two binary numbers using array multipliers is possible, as such a Dong's Code prediction scheme is obtainable.

For the multiplication of 2 binary values X and Y,

$$Sc = nXc + nYc - XcYc - Cc + n \quad (1)$$

Sc: Berger check-bit code
Xc: Number of zeroes in term X
Yc: Number of zeroes in term Y
Cc: Number of zeroes within the carries register
n: Highest number of bits in Xc or Yc

Equation (1) may be modified to provide the Dong's Code equivalent C1 value,

$$C1 = \overline{((XcYc + Cc) Mod\, 2^k)} + 1 \quad (2)$$

This equation may then be implemented into the CSP block within the ALU as shown in Figure 1.

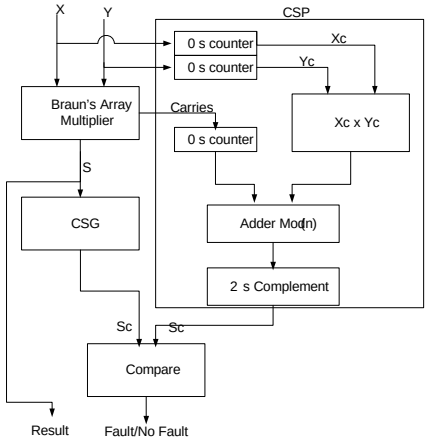


Figure 1: Multiplier error detection operation

The application of this form of error detection on an asynchronous ALU will also provide a faster method for soft error recovery. As transient faults occur the system outputs the fault signal, this error may then be interpreted by the environment in different ways;

- ***Transient error, attempt recovery***
  This method allows the environment outside the ALU to sit and wait for the transient error to pass. Without the concurrent operation of the testing system this transient error would propagate leading to data corruption. With a fully synchronous system the sit and wait method would require full clock cycles to pass before allowing the system to continue, leading to larger delays.
- ***Extreme transient/permanent fault***
  By assuming the system is affected only by transient errors which affect for a short period the system will be prone to lock-up in the case of permanent faults. These may then place the system in a permanent faulted state if using the transient error recovery method at all times. To avoid this, the application of a maximum length recovery time could be implemented to then force a reset (watchdog timer).

5. Results

Previous papers [5] have shown the multiplier function alone may be implemented with a 24% reduction in area by using Dong's Code rather than Berger on simple 8x8 multipliers, and 10% on 16x16bit. The following implementations show different pipelines with different features, using the same technology. The area comparisons are based on Ambit area reports from the Cadence Design Suite on a 0.35µ process.

Presented in this paper is the application of Dong's Code Check Symbol Prediction for array multipliers

into an existing CED 32bit RISC ALU. With the added benefit of a multiplier, an area overhead of 13% has been realised for the implementation of CED to all ALU operations and all ALU register files.

Pipe	Size (μ^2)
Synchronous	$1525x10^3$
Synchronous CED	$1733 x10^3$
Synchronous Multiplier	$1846 x10^3$
Synchronous CED Multiplier	$2100 x10^3$
Asynchronous	$1458 x10^3$
Asynchronous CED	$1693 x10^3$
Asynchronous Multiplier	$1774 x10^3$
Asynchronous CED Multiplier	$2010 x10^3$

Table 1: Asynchronous vs. Synchronous areas

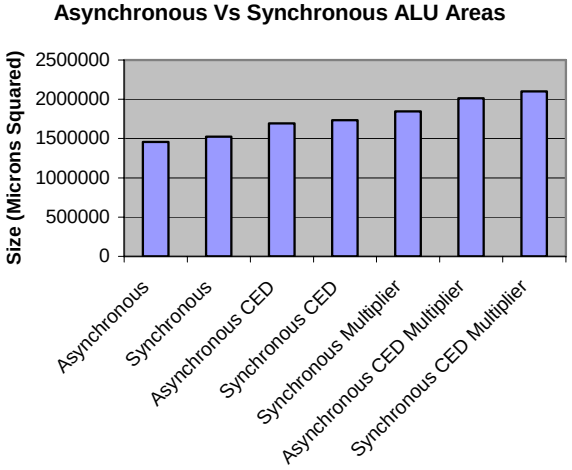


Figure 2: Pipeline comparisons

6. DISCUSSION

This study has focused on a single information redundant code being incorporated into the ALU. By maintaining single codes throughout a system in order to protect register files, and operations, there is the potential for area savings to be made due to the reuse of large sections such as those within the CSP and CSG circuits. All Dong's Code CSP circuits utilise a number of large "zeros counters" and many make use of the "mod n" adders. There is also reuse of the CSG and comparison circuit within every process as shown in the generalised block diagram of the ALU in Figure 3.

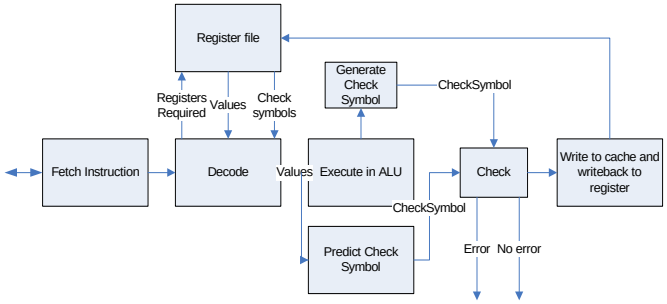


Figure 3: Pipelined process

7. Conclusions

Synchronous system design has become more problematic with issues of clock skew growing. Asynchronous systems require no clock, and no unnecessary power is wasted in generating it, nor distributing it, and by using CED, systems may detect errors as they occur at the cost of area overhead (13%).

Dong's Code and Berger Code are able to detect all single and unidirectional errors, with Dong's Code providing error checking on the check bits themselves to determine errors within transmission or check bit generation. This increased awareness leads to far greater error coverage than that provided by many other codes. The area overhead savings have been found to be 13% for the Dong's Code asynchronous ALU with multiplier functionality, compared to the equivalent synchronous implementation.

The use of Dong's Code has thus been expanded with this implementation of a multiplication Check Symbol Prediction circuit. Thus, Dong's code may now be utilised in a wider variety of implementations, for example an ALU used for DSP applications where multiplication may used extensively. The continuing work of this project involves the study of logarithmic processors and the potential implementation of information redundancy to protect their operation.

Previous work has demonstrated that an information redundant Concurrent Error Detection scheme using Dong's Code can be applied efficiently to an ALU within an asynchronous processor in which the area overheads incurred were approximately 12%. This paper extends the work to the application of this checking scheme to the integration of a multiplier function into an ALU enabling the advantages of using asynchronous design style, to be used in a wider arena of DSP applications. The work outlined in this paper has demonstrated that despite the large structure of an array multiplier and the complexity of check symbol predictions, these savings can be maintained.

REFERENCES

[1] P.D. Hyde, "A Pipelined Asynchronous Self-Checking RISC based Processor", Thesis (PhD), University of Newcastle-upon-Tyne, 2004.

[2] A. Maamar, G. Russell, "Checkbit Prediction Using Dong's code for Arithmetic Functions", *Proceedings 3rd IEEE On-Line Testing Workshop*, Crete, July 1997, pp 254 – 258.

[3] H. Dong, "Modified Berger Codes for the Detection of Unidirectional Errors", *IEEE Transactions on Computers*, Volume C-33, Number 6, June 1984, pp 575 – 575.

[4] J. C. Lo, S. Thanawastien, T. R. N. Rao, "Berger Check Prediction for Array Multipliers and Array Dividers", *IEEE Transactions on Computers*, Volume 42, Number 7, July 1993, pp 892 – 896.

[5] M. Marshall, G. Russell, "An information redundant scheme for online testing of an Asynchronous ALU operation", *Informal proceedings 11th IEEE European Test Symposium (ETS)*, May 2006.

The Royal Academy of Engineering

Newcastle University

Unfolding Models of Asynchronous Systems: Applications to Analysis and Synthesis

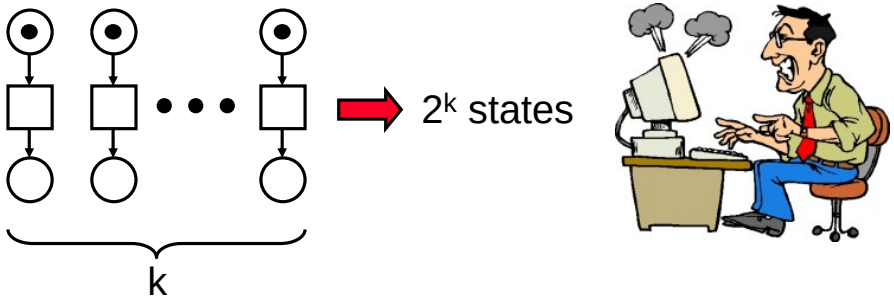
Victor Khomenko

University of Newcastle upon Tyne

Talk outline

- Petri net unfoldings
- Model checking based on unfolding prefixes
  - deadlock checking
  - encoding conflicts
- Beyond model checking
  - resolution of encoding conflicts
  - logic synthesis
- Further developments
  - unfoldings of high-level nets
  - merged processes

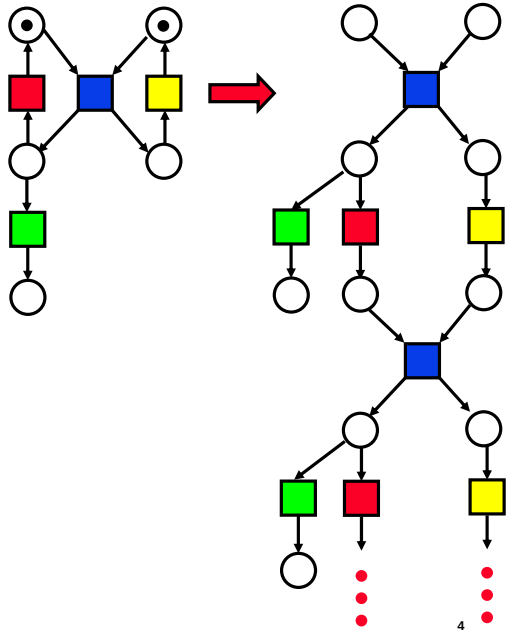
State space explosion problem



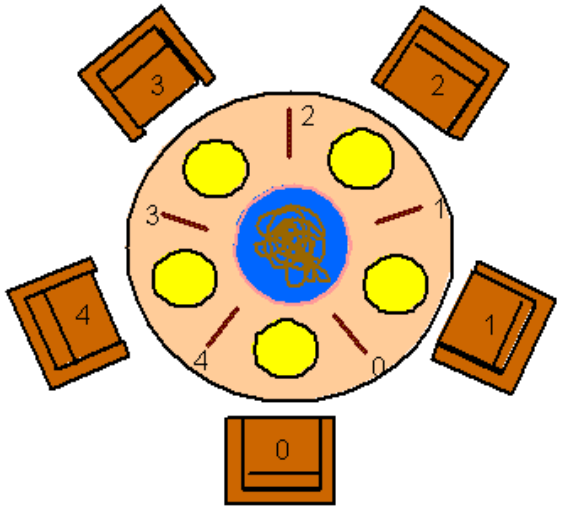
- For efficient synthesis the state space of the system must be explored.
- However, the number of reachable states is often exponential in the size of the spec, particularly when the degree of concurrency is high.

Petri net unfoldings

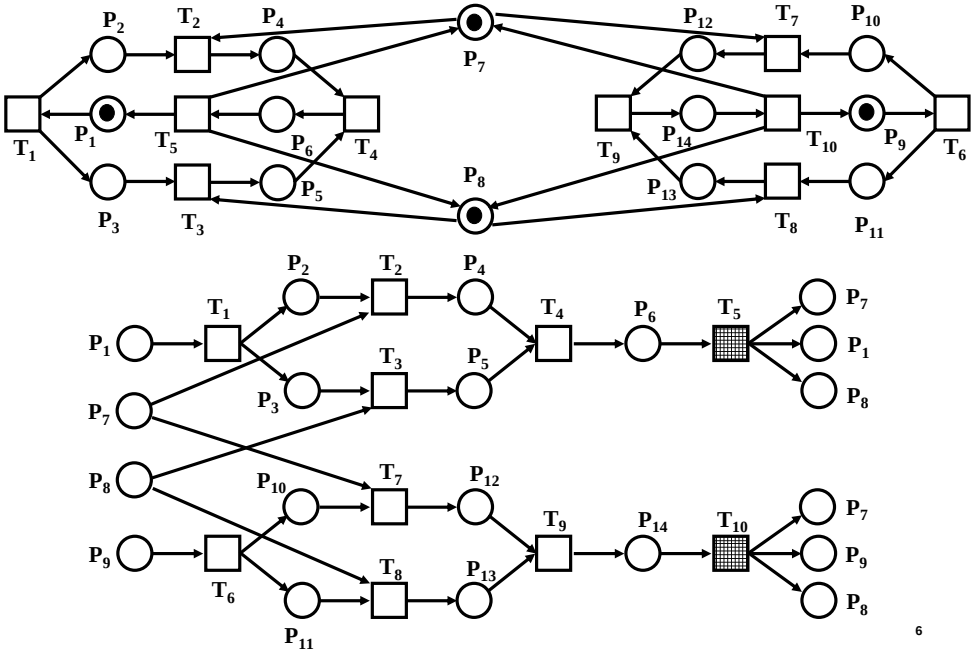
- Represent system states implicitly, using an acyclic net
- Rely on the partial-order view of concurrent computation
- Alleviate the state space explosion problem



Example: Dining Philosophers

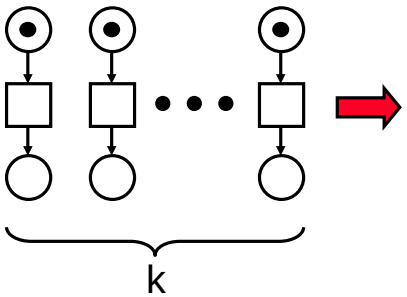


Example: Dining Philosophers



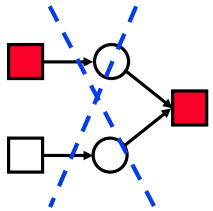
Configurations

- A partial-order analog of traces
- The order of execution of concurrent events does not matter and should not be enforced
  - hence have to explore fewer runs compared to the interleaving semantics

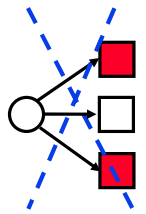


k! traces reaching a deadlock
only one configuration

Structural characterisation

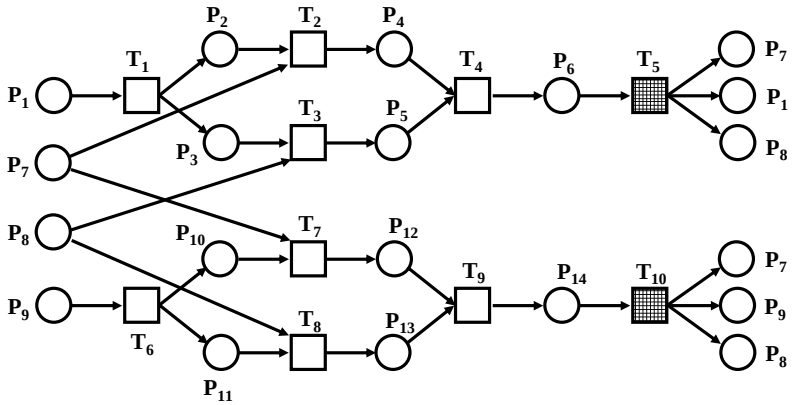


Causality must be obeyed



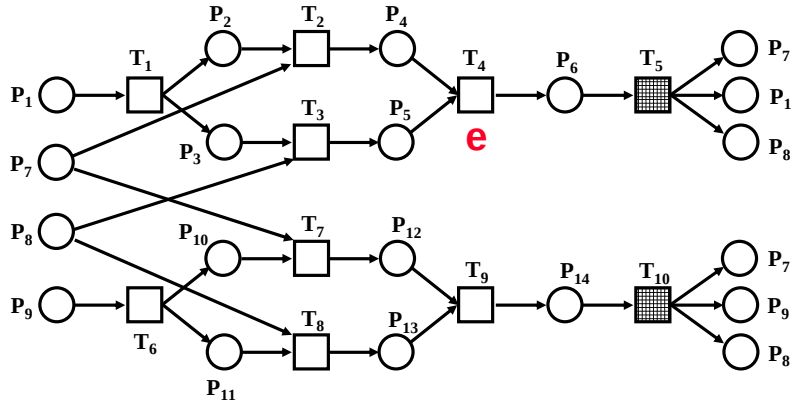
No choices allowed

Example: Dining Philosophers

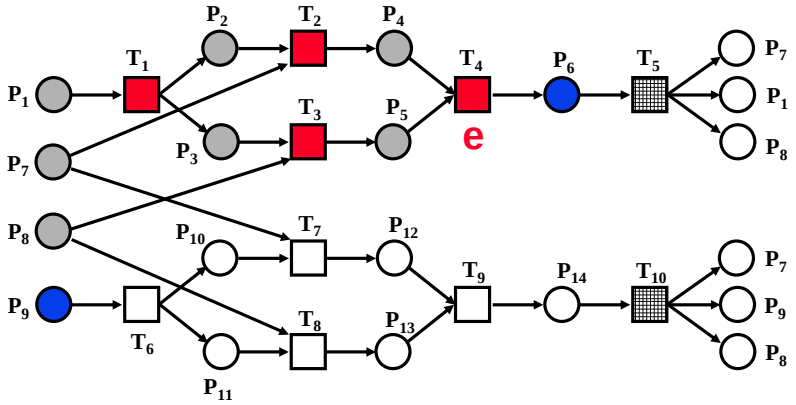


Local configuration

- The *local configuration* [e] of e is the smallest configuration containing e
  - comprised of e and its causal predecessors

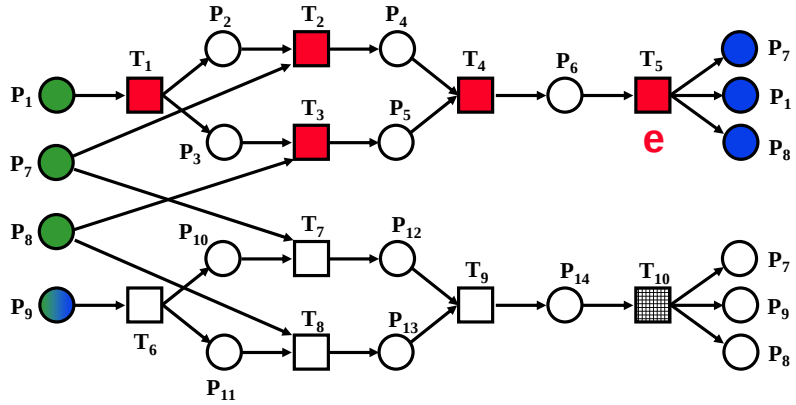


Final marking of a configuration



Cut-off criterion

- An event e is *cut-off* iff there is a [local] configuration C with the same final marking such that C<[e] (< is a certain order partial order on configurations)



The ERV unfolding algorithm

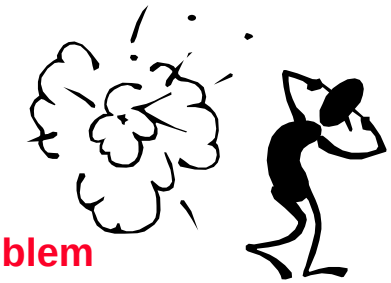
Unf ← places from M_0
pe ← transitions enabled by M_0
cut-off ← ∅
while pe ≠ ∅
  extract e ∈ $\min_<$ pe
  if e is a cut-off event **then** cut-off ← cut-off ∪ {e}
  else
    add e and its postset into Unf
    UpdatePotExt(pe, Unf, e)
end while
add cut-off events and their postsets to Unf

State Graphs vs. Unfoldings

State Graphs:

☺ **Relatively easy theory**
☺ **Many efficient algorithms**

☹ **Not visual**
☹ **State space explosion problem**

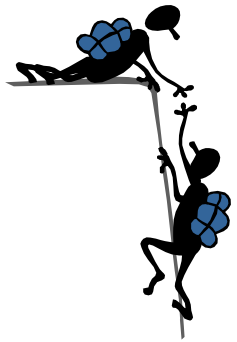


State Graphs vs. Unfoldings

Unfoldings:

☺ **Alleviate the state space explosion problem**
☺ **More visual than state graphs**
☺ **Proven efficient for model checking**

☹ **Quite complicated theory**
☹ **Not sufficiently investigated**
☹ **Relatively few algorithms**



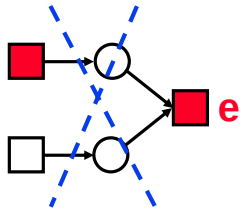
Talk outline

- Petri net unfoldings
- **Model checking based on unfolding prefixes**
  - **deadlock checking**
  - **encoding conflicts**
- Beyond model checking
  - resolution of encoding conflicts
  - logic synthesis
- Further developments
  - unfoldings of high-level nets
  - merged processes

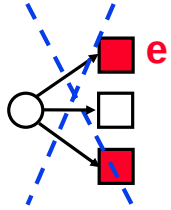
Model checking on PN unfoldings

- **A Boolean expression φ is built using the prefix, such that:**
  - φ is unsatisfiable iff the property holds
  - every satisfiable assignment of φ gives a violation trace
- **φ has the form CONF∧VIOL**
- **Some of the variables of φ are associated with the events of the prefix**

The CONF constraint



If an e is executed than all its [direct] causal predecessors are also executed



If an e is executed than no events in [direct] choice relationship with e can be executed

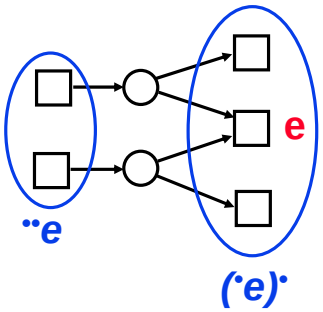
$$\bigwedge_{e}\ \bigwedge_{f\in {}^{\bullet\bullet}e}(e\rightarrow f)\ \wedge\ \bigwedge_{e}\ \bigwedge_{f\in({}^{\bullet}e)^{\bullet}\setminus\{e\}}(\neg e\vee\neg f)$$

The satisfying assignments of CONF correspond to the configurations of the prefix

VIOL: Deadlock

No event is enabled to fire, i.e. for every e:

- **some [direct] predecessor of e has not fired, or**
- **an event in [direct] conflict with e or e itself has fired**



$$\bigwedge_{e}\Big(\bigvee_{f\in {}^{\bullet\bullet}e}\neg f\ \vee\ \bigvee_{f\in({}^{\bullet}e)^{\bullet}}\neg f\Big)$$

The method works for other reachability-like properties as well!

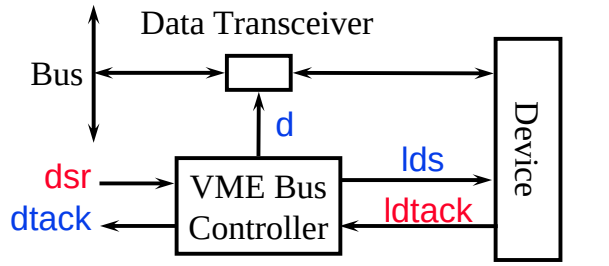
Asynchronous circuits

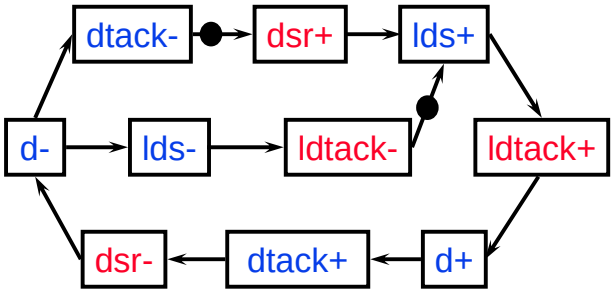
Asynchronous circuits are circuits without clocks

- ☺ Low power consumption
- ☺ Tolerant to process, voltage and temperature variations
- ☺ Low electro-magnetic emission
- ☺ No problems with the clock skew

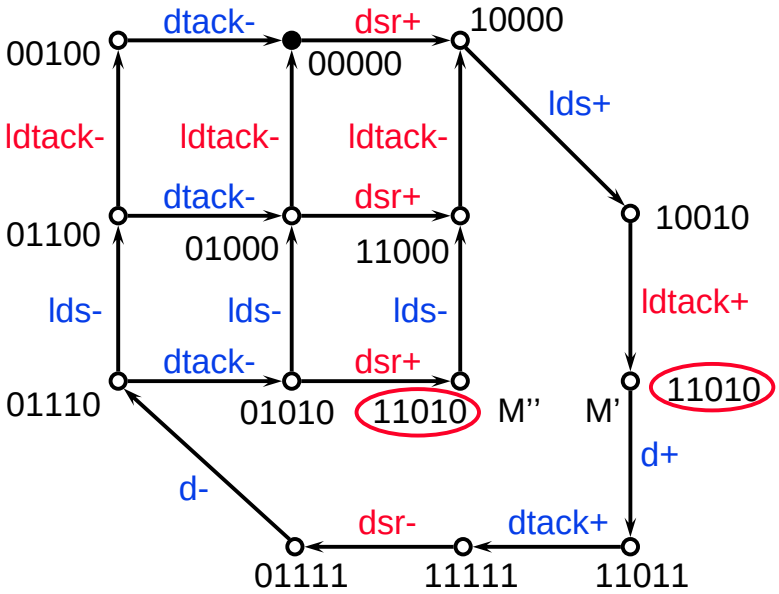
- ☹ Hard to synthesize
- ☹ The theory is not sufficiently developed
- ☹ Limited tool support

Example: VME Bus Controller

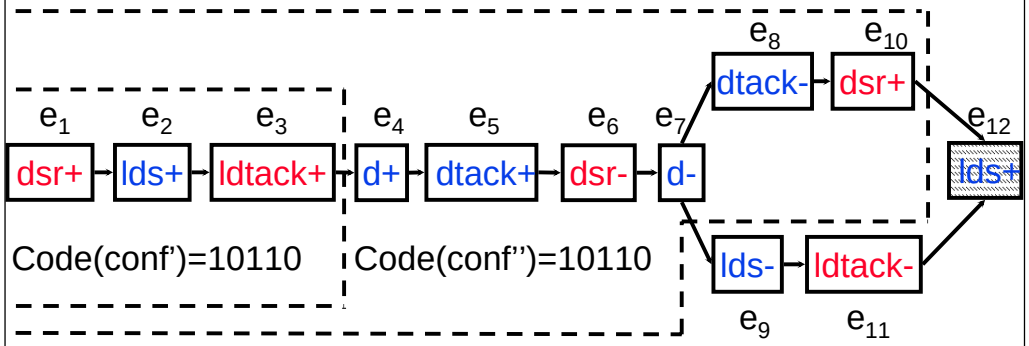




Example: Encoding Conflict



Example: Encoding Conflict



Detection of encoding conflicts

- **A special case of model checking!**
- **φ has the form $CONF_1 \wedge CONF_2 \wedge VIOL$**
- **VIOL is a constraint stating that the two configurations have the same final encodings and enable different sets of output signals**

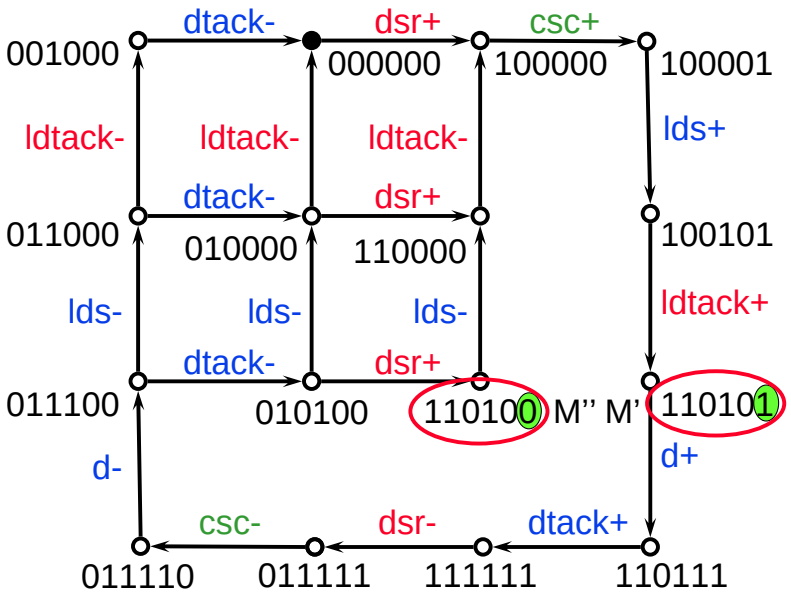
Talk outline

- Petri net unfoldings
- Model checking based on unfolding prefixes
  - deadlock checking
  - encoding conflicts
- **Beyond model checking**
  - **resolution of encoding conflicts**
  - **logic synthesis**
- Further developments
  - unfoldings of high-level nets
  - merged processes

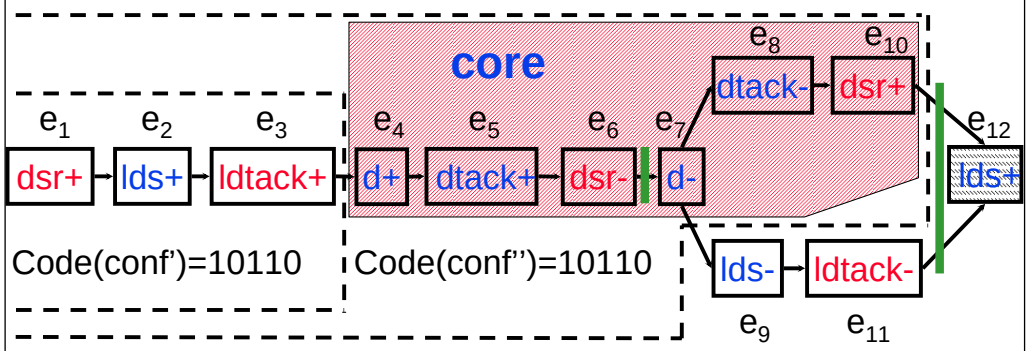
Beyond model checking

Problem: model checking just tells you whether some property holds, but it's not enough for resolution of encoding conflicts and for deriving equations!

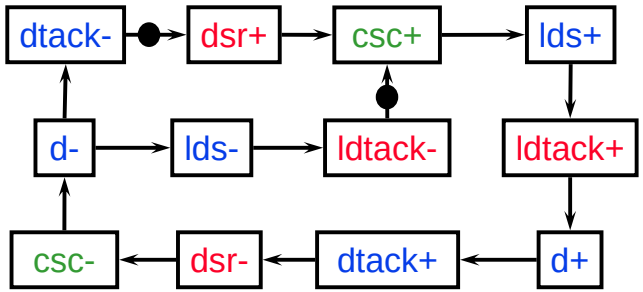
Example: Resolving the conflict



Example: Encoding Conflict

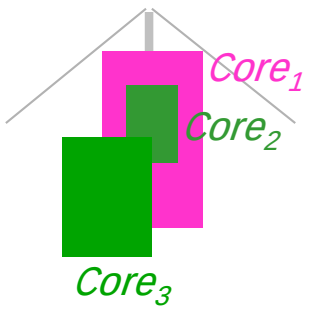


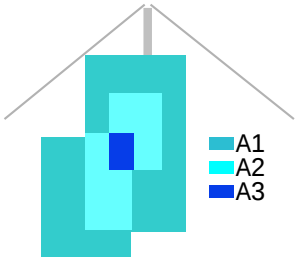
Example: Resolving the conflict

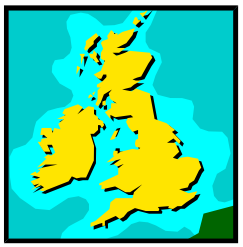


Visualising conflicts: Height map

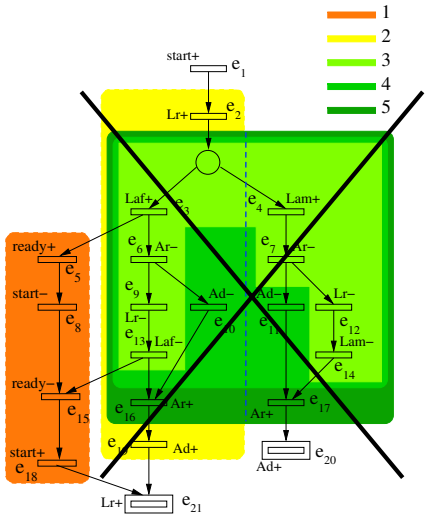
- Cores often overlap
- Highest ‘peaks’ are good candidates for signal insertion
- Analogy with topographic maps



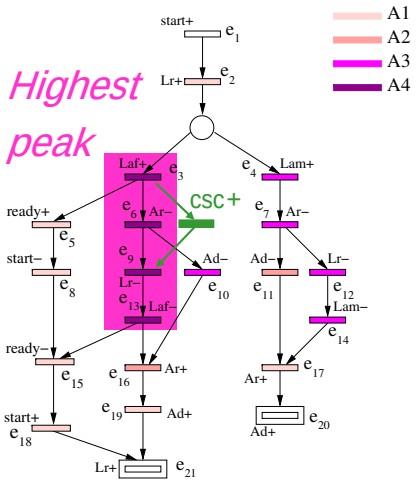




Height map: an example



Core map



Height map

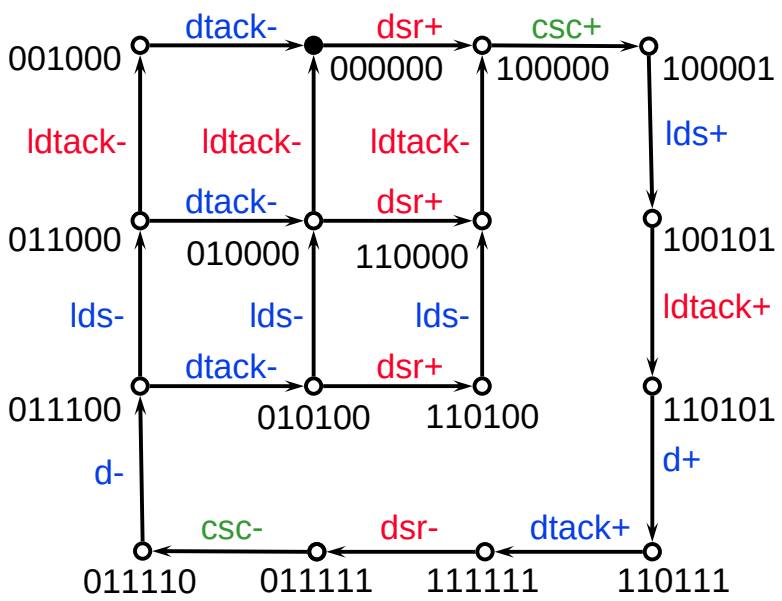
Logic synthesis: Next-state function

- The *next-state function* of each output or internal signal will be implemented as a logic gate in the circuit
- Defined for each such signal z at each reachable state M as

$$Nxt_z(M) = Code_z(M) \oplus Enabled_z(M)$$

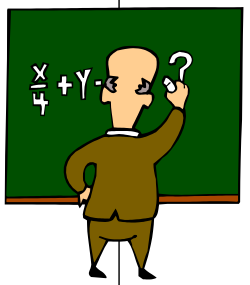
- The value is undefined (‘don’t care’) for unreachable states

Example: Deriving equations

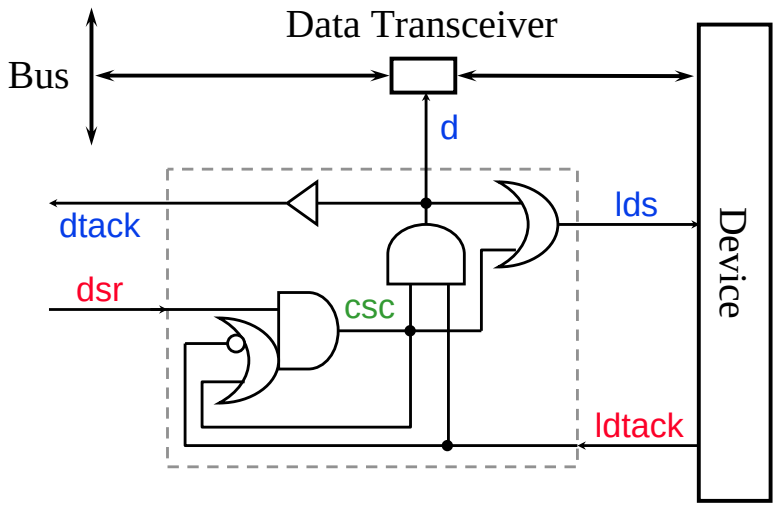


Example: Deriving Equations

Code	Nxt_{dtack}	Nxt_{lds}	Nxt_{d}	Nxt_{csc}
001000	0	0	0	0
000000	0	0	0	0
100000	0	0	0	1
100001	0	1	0	1
011000	0	0	0	0
010000	0	0	0	0
110000	0	0	0	0
100101	0	1	0	1
011100	0	0	0	0
010100	0	0	0	0
110100	0	0	0	0
110101	0	1	1	1
011110	1	1	0	0
011111	1	1	1	0
111111	1	1	1	1
110111	1	1	1	1
Eqn	d	d ∨ csc	csc ∧ ldtack	dsr∧ (¬ldtack∨csc)



Example: Resulting Circuit



Logic synthesis on unfoldings

Challenge: how to do this without building the state graph, and using only the unfolding prefix?

Logic synthesis on unfoldings

- **Problem:** given a prefix and a set X of signals which are known to be a support of the given output or internal signal z, compute the truth table of Nxt_z
- Let $\varphi = CONF \wedge CODE_X$ where $CODE_X$ relates the values of all signals in X with the configuration
- Compute the projection of φ onto X

Need to know how to compute projections!

Example: computing projections

a	b	c	d	e
0	1	0	0	1
0	1	0	1	0
0	1	0	1	1
0	1	1	0	0
0	1	1	0	1
0	1	1	1	0
0	1	1	1	1
1	0	0	0	1
1	0	0	1	0
1	0	0	1	1
1	0	1	0	0
1	0	1	0	1
1	0	1	1	0
1	0	1	1	1

$$\varphi = \overbrace{(a \vee b)}^{a \oplus b}(\neg a \vee \neg b)(c \vee d \vee e)$$

$Proj_{\{a,b,c\}}\ \varphi$

a	b	c
0	1	0
0	1	1
1	0	0
1	0	1

Computing projections

$$\varphi = \overbrace{(a \vee b)}^{a \oplus b}(\bar{a} \vee \bar{b})(c \vee d \vee e)(a \vee \bar{b} \vee c)(a \vee \bar{b} \vee \bar{c})(\bar{a} \vee b \vee c)(\bar{a} \vee b \vee \bar{c})$$

$Proj_{\{a,b,c\}}\ \varphi$

a	b	c	d	e
0	1	0	0	1
0	1	1	0	0
1	0	0	0	1
1	0	1	0	0

UNSAT

Incremental SAT

Talk outline

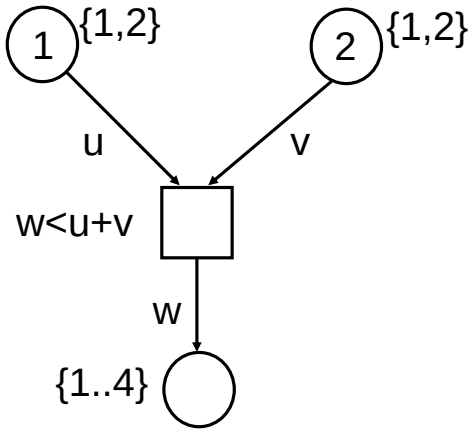
- Petri net unfoldings
- Model checking based on unfolding prefixes
  - deadlock checking
  - encoding conflicts
- Beyond model checking
  - resolution of encoding conflicts
  - logic synthesis
- Further developments
  - unfoldings of high-level nets
  - merged processes

Further developments

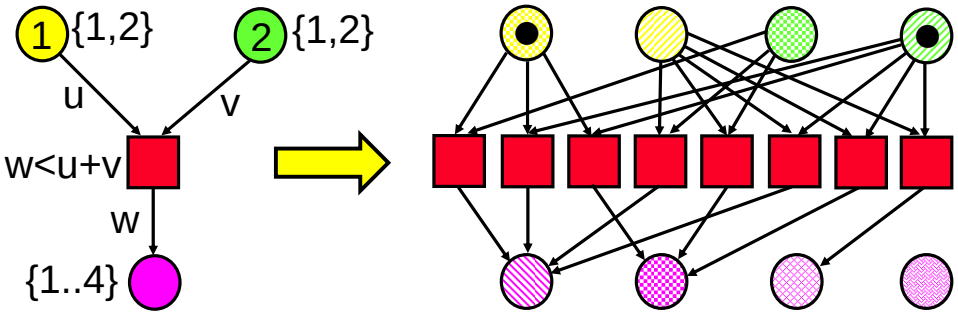
Roadmap:

- **Unfoldings of more complicated then low-level Petri nets models**
- **More concise then unfoldings representations of state spaces**

Coloured (high-level) PNs



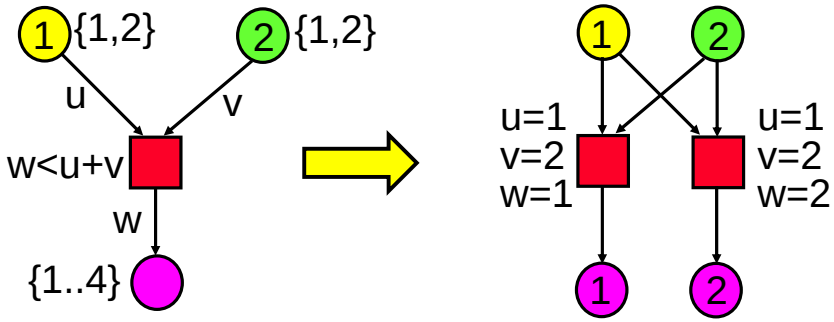
Expansion



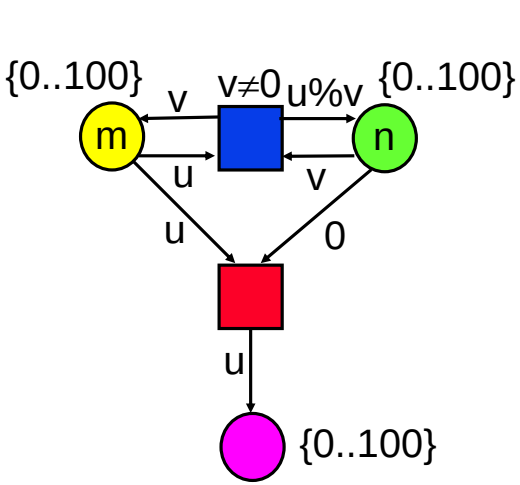
☺ **The expansion faithfully models the original net**

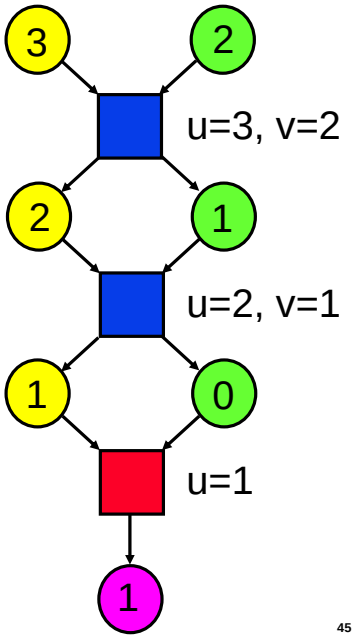
☹ **Exponential blow up in size**

Unfolding

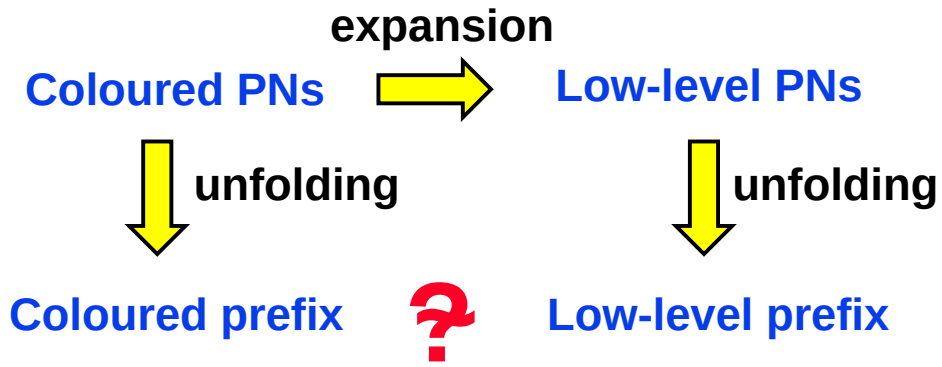


Example: computing GCD

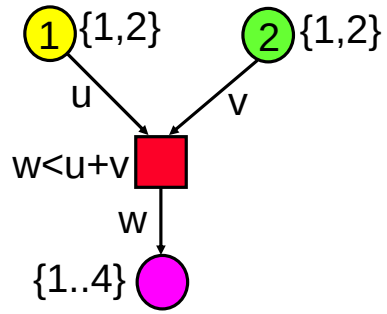


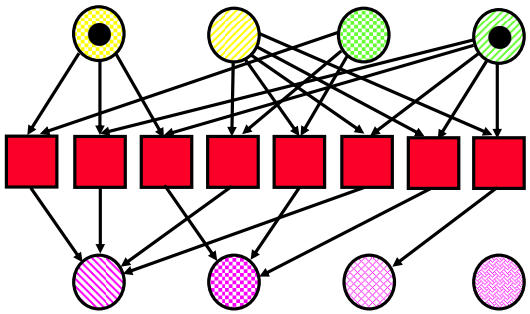


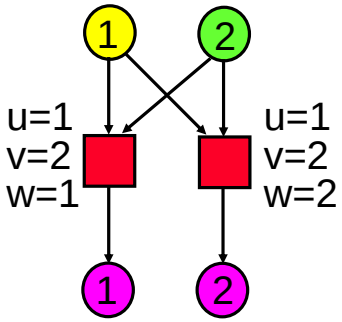
Relationship diagram

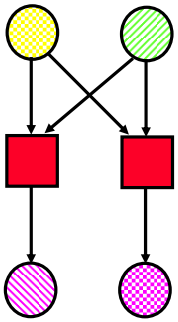


Relationship diagram

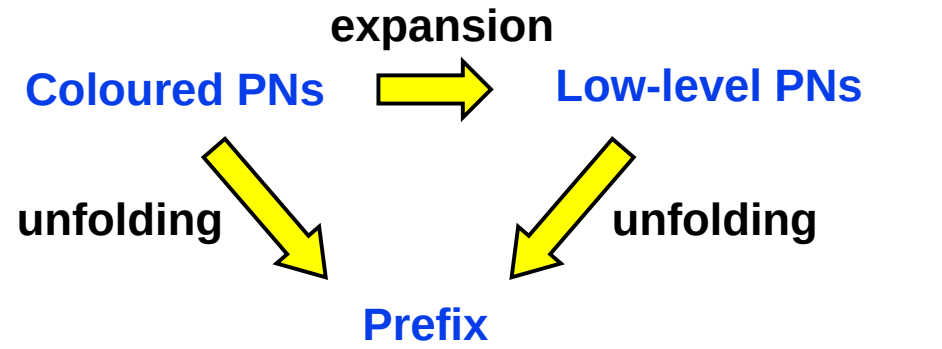








Relationship diagram



Benefits

- ☺ Avoiding an exponential blow up when building the expansion
- ☺ Definitions are similar to those for LL unfoldings, no new proofs
- ☺ All results and verification techniques for LL unfoldings are still applicable
- ☺ Existing unfolding algorithms for LL PNs can easily be adapted

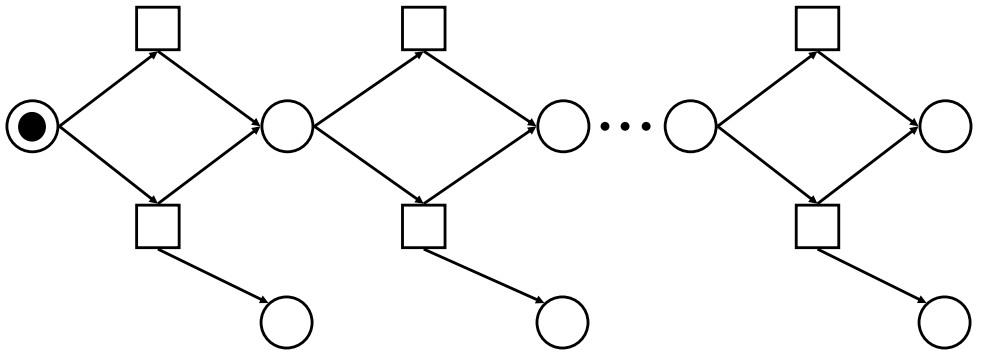
Experimental results

- ☺ Tremendous improvements for colour-intensive PNs (e.g. GCD)
- ☺ Negligible slow-down (<0.5%) for control-intensive PNs (e.g. Lamport's mutual exclusion algorithm)

Merged processes

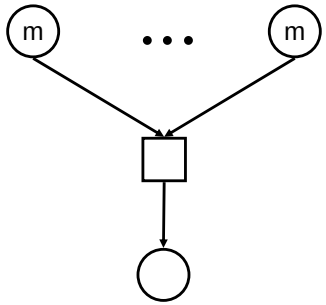
Problem: Unfoldings do not cope well with other than concurrency sources of state space explosion, e.g. with sequence of choices and non-safe PNs

Example: sequence of choices



No event is cut-off, the prefix is exponential

Example: non-safe PN



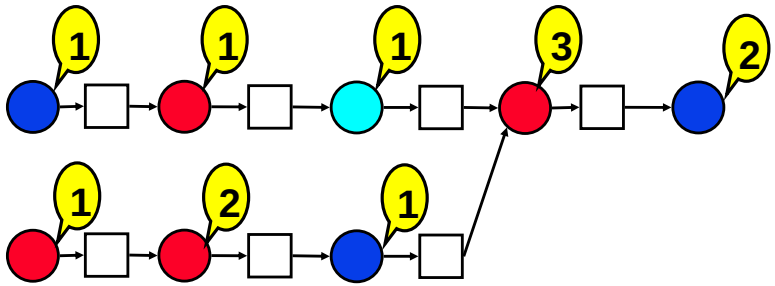
Tokens in the same place are distinguished in the unfolding, the prefix is exponential

Example: non-safe PN

Wanted

A data structure coping not only with concurrency but also with other sources of state space explosion

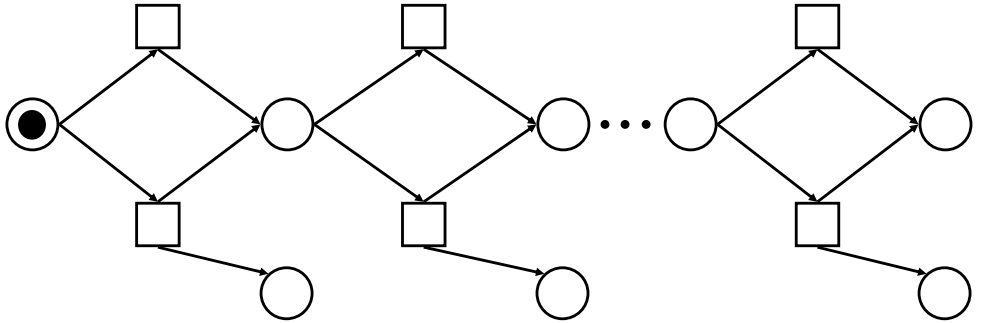
Occurrence-depth

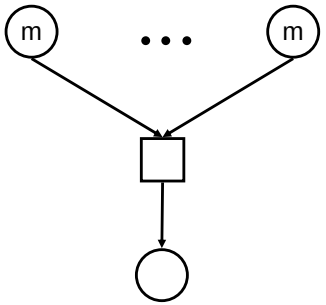


Merged Process:

- ☺ **Fuse conditions with the same label and occurrence-depth**
- ☺ **Delete duplicate events**

Examples





Merged processes of these nets coincide with the original nets, even though unfoldings are exponential!

Advantages

- Merged processes can be used for model checking
- In practice, they are often by orders of magnitude smaller than unfolding prefixes
- In many cases they are just slightly larger than the original PNs
- In some cases they are smaller than the original PNs due to removal of dead places

Thank you!

Any questions?

CSC-Aware STG-Decomposition

Mark Schaefer
Institute of Computer Science, University of Augsburg, Germany
mark.schaefer@informatik.uni-augsburg.de

1 Introduction

Previous attempts to decompose STGs (and to synthesise the resulting components, instead of synthesising the original STG directly) turned out to be quite successful regarding the runtime [VW02, VK05]. While this was even the case for a first naïve implementation, recent improvements of our decomposition algorithm [SVWK06] improved the runtime even more.

Unfortunately, decomposition also reduces the solution space for synthesis, sometimes in a way that the resulting components could not be synthesised due to irreducible CSC conflicts. But even when the components are synthesisable it is often needed to solve CSC with additional signals, though increasing area; this can also happen if the original STG has CSC initially.

In this paper, our latest decomposition algorithm is described, which tackles this problem by reducing the components only to the point where CSC is satisfied by signals already belonging to the circuit instead of solving CSC with new signals. This is an advantage for STGs which satisfy CSC initially; if this is not the case, CSC has to be solved for the STG as well as for the final components.

For the time being, the new decomposition algorithm is merely a proof of concept and in the last section, some possible optimisations are discussed.

The paper is organised as follows: in the next section a brief introduction to our decomposition approach is given. In Section 3 the new algorithm is described and some experimental results are given. Finally, there is a conclusion and an outlook to future work.

Due to lack of space, STGs (and CSC in particular) are not explained; a detailed introduction can be found e.g. in [CKK+02].

2 STG Decomposition

A detailed introduction to our decomposition approach can be found in [VW02, VK05, SV05, SVWK06]; our decomposition tool DesiJ can be found at www.informatik.uni-augsburg.de/en/chairs/swt/ti/research/tools/desij.

Synthesis with STG decomposition works roughly as follows: a partition of the output signals of the given specification STG N is chosen, and the decomposition algorithm decomposes N into component STGs, one for each set in this partition. Then, a circuit is synthesised from each component, and the interconnection of these circuits has a behaviour that conforms to the specification.

This correctness is formally proven in [VW02, VK05] on the level of STGs. Interconnection on the physical level simply means to connect the circuits with a wire for each common signal, i.e. if an output x of a component C_1 is also an input of a component C_2. On the STG level, interconnection corresponds to the ordinary parallel composition for Petri nets.

To describe the decomposition algorithm in more detail, we discuss below the notion of *auto-conflicts*, which plays an important part during decomposition. The second subsection deals with the decomposition algorithm itself.

STGs can model more behaviour than a real-life circuit can show. For example, inconsistent STGs cannot be implemented although they are allowed in principle. Another problematic case are *dynamic conflicts*, i.e. two transitions of an STG enabled under some reachable marking, where firing one would disable the other.

If the conflicting transitions correspond to different input signals then they model a choice made by the environment, and this is not a problem. However, if at least one of the signals is an output, then the specification cannot be implemented as an asynchronous circuit. There are three problematic cases:

1. One transition is labelled with an input edge, the other with an output edge. This conflict is very hard to implement, since both signal edges are independently generated and may occur at the same time. Nevertheless, our decomposition method and our tool DesiJ cover such conflicts, but we will not discuss them here any further.

2. Both transitions are labelled with different output edges. A circuit which can handle such conflicts is called an *arbiter* and cannot be implemented as a purely digital circuit. STGs with such conflicts can also be handled by our decomposition method, which does not introduce new conflicts of this kind. For a detailed discussion see [VW02, VK05].

3. Both transitions are labelled with the same signal edge, a so-called *auto-conflict*. Such a nondeterministic choice can hardly be handled by circuits, and we assume that decomposition is only applied to STGs without auto-conflicts. During

our decomposition algorithm, auto-conflicts could be generated; this is considered as an indication that too many signals were lambdarised in an STG. In this case *backtracking* is performed and a signal is delambdarised, see below for more details.

Observe that conflicts between λ-labelled transitions are ignored.

Auto-conflicts are dynamic in nature, i.e., to detect them one has to generate the reachability graph, which we want to avoid because of their potential exponential size. A much simpler notion is that of a *structural auto-conflict*, where two equally labelled transitions have a common place in their presets. This is a necessary precondition for auto-conflicts and can be checked structurally. Consequently, the decomposition algorithm of [VW02] checks only for structural conflicts, conservatively treating them as dynamic ones.

In the following, we assume that we are given a deterministic, consistent specification N without structural auto-conflicts; first, one chooses a *feasible partition*, i.e. a family $(In_i, Out_i)_{i\in I}$ for some set I such that the sets Out_i are a partition of Out, $In_i \subseteq Sig \setminus Out_i$ for each i and furthermore:

- If two output signals x_1, x_2 are in structural conflict in N, then they have to be in the same Out_i.
- If there are $t, t' \in T$ with $t' \in (t^\bullet)^\bullet$ (t is called *syntactical trigger of* t'), then $l(t') \in Out_i$ implies $l(t) \in In_i \cup Out_i$.

Clearly, for each STG N there is a *minimal feasible partition* Υ_N such that the Out_i are minimal and only necessary inputs are included in In_i.

If we have a feasible partition, we can build another feasible one by adding additional input signals to one of the members or by merging two members (In_1, Out_1) and (In_2, Out_2) to a new one $((In_1 \cup In_2) \setminus Out_1 \cup Out_2, Out_1 \cup Out_2)$.

All possible partitions can be generated by applying these operations repeatedly to Υ_N.

For each member (In_i, Out_i) of a partition an *initial component* is generated from N: in a copy of the original STG N, every signal not in $In_i \cup Out_i$ is lambdarised, i.e. labelled with λ, and the signals in In_i are considered as inputs of this component – even if they are outputs of N.

The following operations are applied to each of these components; this process is called *reduction*:

- *secure contraction* of λ-transitions
- deletion of redundant places
- deletion of redundant transitions
- *backtracking*

We call the first three of these operations *reduction operations*. The reduction of an initial component leads to a component-STG without λ-transitions. Each component-STG is then synthesised, usually by constructing its reachability graph. Very often, adding up the sizes of these graphs gives a number much smaller than the size of the reachability graph of N, in which case the decomposition can be seen as successful. Actually, it might already be beneficial if each reachability graph is smaller than the one of N, in particular for reducing the peak memory usage.

We will now describe the above operations in more detail. The *contraction of a transition* t generates a set of new places $\{(p, q) | p \in {}^\bullet t, q \in t^\bullet\}$ (each one of them inherits the tokens and arcs of its 'inner' places) and removes t, ${}^\bullet t$ and $t^\bullet$ from the net; cf. Figure 1.

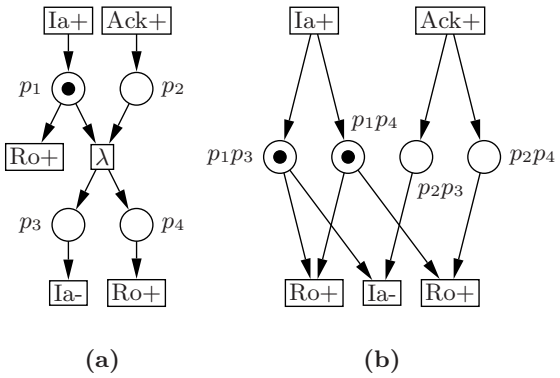


Figure 1: Transition contraction with generation of structural auto-conflict. **(a)** Initial net. **(b)** After contraction of the λ-labelled transition.

Contractions are only performed if they are '*secure*' (implying language preservation) and *no new structural auto-conflict is generated*. It is easy to see that the contraction of a transition t increases the number of places by $|{}^\bullet t| \cdot |t^\bullet| - (|{}^\bullet t| + |t^\bullet|)$.

Redundant places are a subclass of *implicit places* which can be deleted without changing the firing sequences of the STG. The difference is that looking for implicit places requires the reachability graph while redundant places can be detected structurally; hence, we only look for the latter ones during decomposition.

There are two kinds of *redundant transitions*. First, if there are two λ-labelled transitions which are connected to every place in the same way, one of them can be deleted without changing the traces of the STG. Second, a λ-labelled transition t with $W(t, p) = W(p, t)$ for every place p can also be deleted, since its firing does not change the marking and is not visible on the level of traces.

These two operations may seem trivial, but especially the deletion of redundant places is essential for getting small components, since very often the existence of such places prevents further transition contractions. The same is also true to some extent for redundant transitions.

Backtracking means to delambdarise a signal of the initial component, to consider it as an additional input signal and to start reduction anew. This is applied if there are still λ-transitions left but none of the reduction operations can be performed. In particular,

if the contraction of a λ-transition t would generate a new structural auto-conflict, this is considered as an indication that too many signals of a component were lambdarised to produce its output signals appropriately; this can be changed by delambdarising t, i.e. restoring the initial label, and – informally speaking – providing more information to the circuit.

After the last backtracking, when enough signals are added to the initial component, only the reduction operations have to be applied to get the final component. This means that backtracking is only needed to detect these additional signals; if they are known in advance, one can perform decomposition completely without backtracking.

3 The New Algorithm

A *decomposition tree* is a tree with sets of signals attached to the nodes; the tree is traversed starting at the root and the initial STG, in every node the corresponding signals are contracted such that every leaf corresponds to a desired final component; for more details of tree decomposition, see [SVWK06].

The main idea of CSC-aware decomposition is as follows (cf. Figure 2):

1. A decomposition tree as described above is generated and completely traversed in a depth-first-search manner, starting from the root.

2. Every time a node k is entered for the first time, i.e. coming from $parent(k)$, the respective signals $sig(k)$ are contracted.

3. When a leaf l was finished, CSC is checked for the resulting component C_l. If it is satisfied, C_l is saved as final result. Otherwise, the CSC violation traces of C_l are determined and a *solve-task* is generated out of them and associated to $parent(l)$.

4. Every time a node k is entered coming from $children(k)$, every solve-task associated to k is considered separately: the contained violation traces are *inversely projected* to the component C_k. Depending on the result CSC is solved and the corresponding component is generated, or the solve-task is updated and associated to $parent(k)$.

The single steps are now considered in more detail.

The generation of the decomposition tree uses the same algorithm as for tree decomposition, only additional parent pointers are saved for each node (except the root). The traversal of the tree was changed from a recursive pre-order algorithm to the described depth-first traversal.

To perform CSC-aware decomposition more efficiently, the implementation of the STG class in our tool was enhanced with an undo support; instead of saving an intermediate result for each node explicitly, there is only one STG—initially the specification—which is modified when going down in the tree and restored when going up.

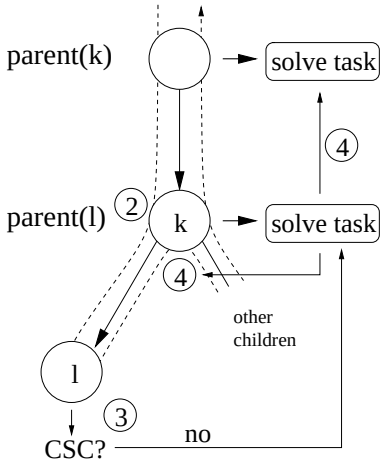


Figure 2: Depth-first-search traversal (dashed line) of a decomposition tree with the new algorithm.

CSC is checked externally with the tools Punf and Mpsat [Kho02, KKY04a, KKY04b], which can return all CSC violation traces for a given STG.

If there are actually CSC conflicts in C_k, the corresponding trace pairs are stored in a solve-task. This data structure contains additionally the outputs of C_k, thus always a proper component can be generated.

Furthermore, there is a counter for the number of updates of a task; if this counter exceeds a given value it is no longer tried to solve CSC internally, but by adding new signals. This prevents a certain task from being moved up to high (producing large components) if the corresponding CSC conflict is there initially.

The solve-task is associated to $parent(k)$ and handled there when the depth-first-search is going up and the modifications in C_k were undone, thus allowing the inverse projection of the violation traces.

Let v_l and v_k be traces of C_l, C_k respectively. Trace v_k is an inverse projection of v_l if $v_l = v_k|_{Sig_l}$. Of course, there are many different inverse projections of v_l; our algorithm looks for the shortest one, which can be proved to be unique.

Given the trace pair (v_l^1, v_l^2) which leads to a CSC violation, the shortest inverse projections of both traces (v_k^1, v_k^2) are calculated. If now $codeChange(v_k^1) \neq codeChange(v_k^2)$, the corresponding CSC conflict was possibly destroyed. Additionally, it might be possible that the additional signals caused new CSC conflicts, and therefore it is checked for CSC externally again.

If there are actually new conflicts, a new solve-task is generated and associated to the respective parent If, on the other hand, CSC was solved the respective component is saved as final result.

The new algorithm was applied to some benchmark examples, the results can be found in Table 1. The value in the Petrify column denotes the time (in seconds) used by the tool Petrify (by J. Cortadella) for

STG	Petrify	DesiJ
2pp.arb.nch.03.csc.g	1	1
2pp.arb.nch.06.csc.g	14	2
2pp.arb.nch.09.csc.g	116	4
2pp.arb.nch.12.csc.g	≥ 300	10
2pp-wk.03.csc.g	1	1
2pp-wk.06.csc.g	9	2
2pp-wk.09.csc.g	31	3
2pp-wk.12.csc.g	≥ 300	18
3pp.arb.nch.03.csc.g	4	1
3pp.arb.nch.06.csc.g	134	3
3pp.arb.nch.09.csc.g	≥ 300	7
3pp.arb.nch.12.csc.g	≥ 300	22
3pp-wk.03.csc.g	1	1
3pp-wk.03.csc.g	31	3
3pp-wk.03.csc.g	≥ 300	7
3pp-wk.03.csc.g	≥ 300	22

Table 1: Results of the benchmark examples. Time values are given in seconds.

synthesis, the value in the DesiJ column denotes the time used by DesiJ for decomposing the STG plus the time used by Petrify for synthesising the components. In every case a complex-gate implementation was derived. If Petrify was not able to derive equations within 5 minutes, synthesis was aborted, denoted by '≥ 300' entries.

One can see that DesiJ leads in every case to a much faster synthesis; most of the time was used for decomposition; synthesising all components usually took less than 1 second. Compared to tree decomposition, the runtime was increased by about 10%.

4 Conclusion

We introduced a new algorithm for STG decomposition which tackles one of the most important problems of our decomposition approach: deriving implementable components more often. Although in most cases only a small part of the components was not implementable, this was enough to prevent the synthesis of the original specification in some cases. Furthermore, when possible the CSC conflicts of the derived components are now 'solved'with signals already belonging to the STG instead of using additional signals.

Still, there are possibilities for optimisation. At the moment, if a CSC conflict was resolved, the final component will contain all signals whose contractions were undone. Since more additional signals might lead to more new CSC conflicts, a future implementation will only keep the needed signals, i.e. the ones which actually resolve the conflict. For this, it will be needed to redo some decomposition steps.

If this is not desired, it is possible to produce a component representing all descendent components of a node, i.e. all outputs assigned to the leafs below the current node are merged to one output set and produced by the STG assigned to the current node. This is very similar to tree aggregation, described in [SVWK06], except that the reasons for this procedure are different. A disadvantage of this method might be, that adding outputs to a component can easily increase the number of CSC conflicts.

It could also happen, that more than one solve-task is associated to a node during the depth-first search and it might be an advantage to combine these tasks, esp. in combination with the first optimisation proposal.

Acknowledgements: I would like to thank Victor Khomenko for providing me his tools Punf and Mpsat and for pointing me to the concept of undoing STG modifications.

References

[CKK+02] J. Cortadella, M. Kishinevsky, A. Kondratyev, L. Lavagno, and A. Yakovlev. *Logic Synthesis of Asynchronous Controllers and Interfaces*. Springer, 2002.

[Kho02] V. Khomenko. Punf *Documentation and User Guide. Version 6.01*. Manual, 2002.

[KKY04a] V. Khomenko, M. Koutny, and A. Yakovlev. Detecting state coding conflicts in stg unfoldings using sat. *Fundamenta Informaticae*, 62(2):1–21, 2004.

[KKY04b] V. Khomenko, M. Koutny, and A. Yakovlev. Logic synthesis for asynchronous circuits based on Petri net unfoldings and incremental sat. In Canada Kishinevsky M. and Ph. Darondeau, editors, *ACSD 2004*, pages 16–25, 2004.

[SV05] M. Schaefer and W. Vogler. Component refinement and CSC solving for STG decomposition. In Vladimiro Sassone, editor, *FOSSACS 05*, Lect. Notes Comp. Sci. 3441, pp. 348–363. Springer, 2005.

[SVWK06] M. Schaefer, W. Vogler, R. Wollowski, and V. Khomenko. Strategies for optimised STG decomposition. In *ACSD 06*, 2006.

[VK05] W. Vogler and B. Kangsah. Improved decomposition of signal transition graphs. In *ACSD 2005*, 2005.

[VW02] W. Vogler and R. Wollowski. Decomposition in asynchronous circuit design. In J. Cortadella et al., editors, *Concurrency and Hardware Design*, Lect. Notes Comp. Sci. 2549, 152 – 190. Springer, 2002.

Validation of an Asynchronous Synthesis Back-End

Nitin Gupta, Doug Edwards
School of Computer Science, The University of Manchester, Oxford Road, Manchester, M13 9PL, UK.

{nitin.gupta,doug.edwards}@cs.man.ac.uk

Abstract

When synthesising a design, validation must be performed on the synthesisable modules to ensure the gate-level modules behave as expected. Existing synchronous design flows are mature and ensure the gate-level netlist matches the register-transfer-level (RTL) specification. Asynchronous synthesis flows do not have this level of confidence and require alternative validation methods to ensure correctness of the synthesisable modules. These methods improve the quality of asynchronous design flows and elevate the confidence in synthesised designs. The work here presents a simulation-based method to validate synthesisable asynchronous handshake circuits. The method uses a behavioural simulation model to identify validation errors in handshake component designs. Tools have been integrated with the Balsa design flow to demonstrate the effectiveness of this method in identifying validation errors in handshake component designs.

1. Introduction

Validation is the process of ensuring a design behaves as expected. For synthesis design flows, this process usually involves ensuring the gate-level design matches a behavioural-level specification. Validation of synchronous designs consists of performing functional and timing validation. Functional validation is performed pre-synthesis at the register-transfer-level (RTL) to ensure the logic of the behavioural model matches the initial specification. Timing validation is performed post-synthesis at the gate-level to ensure timing constraints are met between registers or latches. Synchronous design flows have a level of confidence in the synthesis process and ensure the synthesised gate-level netlist matches the behavioural specification. Thus functional validation of the gate-level netlist is not performed. Asynchronous design flows are still maturing and do not follow synchronous design flows. Correctness of the behavioural model does not ensure correctness of the gate-level model. Functional and timing validation must be performed on the gate-level design to ensure correctness of the synthesisable gate-level modules and increase the level of confidence of asynchronous synthesis flows.

The method presented here focuses on performing validation of asynchronous circuits in a modular way that allows back-end designers of the synthesis flow to validate individual handshake component designs and easily integrate them into the synthesis flow. The method relies on the model of handshake circuits to simplify the validation process by abstracting away the specification of communication protocols for each channel. Although this method is intended for back-end designers, the method can be used by circuit designers to perform timing validation of entire designs.

2. Handshake Circuits

Handshake circuits can be specified using a language that easily describes the functionality of the circuit [1][3][4]. These circuits can be compiled into a behavioural handshaking model and synthesised into a gate-level netlist without requiring the designer to specify the communication protocol for each element. Handshake circuits can be constructed independent of the implementation style (i.e. the handshake protocol and data encoding). Detailed explanations of the various asynchronous handshake protocols and data encodings can be found in [11][13]. This work focuses on using single-rail handshake circuits [11] that implement four-phase handshaking protocols, but applies to other implementation styles.

3. Validation of Handshake Components

Validation ensures the implemented circuit is behaving correctly as expected, ensures design requirements are being met, and also helps to debug design errors. In order to validate a handshake component, a number of checks must be made to ensure correctness [2]. These checks are based on analyzing the behaviour of each communication channel of the implemented handshake component to ensure no unexpected transitions occur. Because handshake circuits follow specific protocols, the behaviour is fixed for each channel and can easily be validated. Below is an explanation of the various critical errors that must be validated.

3.1. Handshake Protocol Violation

A handshake protocol violation occurs when transitions on the request and acknowledge signals of a channel are not sequenced correctly. This type of error is considered a functional validation error. Correct sequencing of control signals ensures data operations are sequenced in the correct order. Figure 1 illustrates the correct and incorrect sequencing of four-phase handshake protocols.

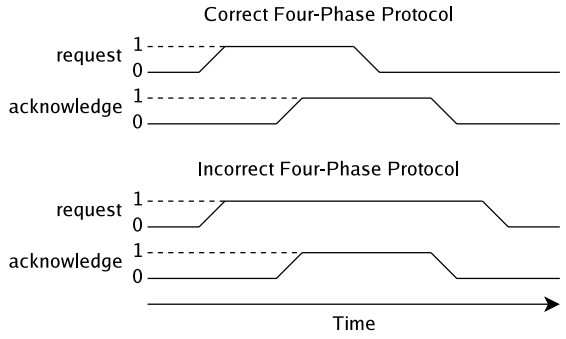


Figure 1: Illustration of handshake protocol violation

3.2. Bad Data

A bad data error occurs when data is undefined or incorrect at start of the data validity period. The data validity period is the time required for data to remain stable, and is signalled by transitions on the request and acknowledge signals. During this period, the handshake component assumes the data is stable and correct, and thus operates on the data. Figure 2 illustrates the minimum validity period needed for common data validity protocols of four-phase single-rail handshake circuits [11], based on whether the sender of data actively starts the communication (push channel) or passively waits for the communication (pull channel). Figure 3 illustrates when a bad data error would be flagged if data is still changing at the start of the validity period, but this error should also be flagged if stable but unexpected data is detected at this point.

3.3. Data Validity Violation

A data validity violation occurs when data is stable and correct at the start of the validity period, but changes before the end of the validity period. Figure 3 illustrates this violation for the broad protocol. As explained above, if data changes during the validity period, the handshake component may operate on incorrect data. Data validity violations typically appear as timing errors when delays are incorrectly matched, but actually are functional errors in the design or composition of handshake components.

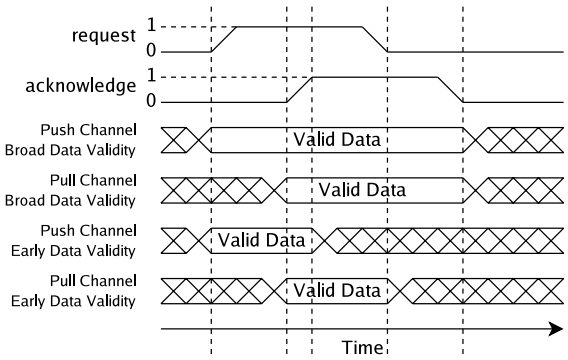


Figure 2: Common data validity protocols for four-phase single-rail handshake circuits

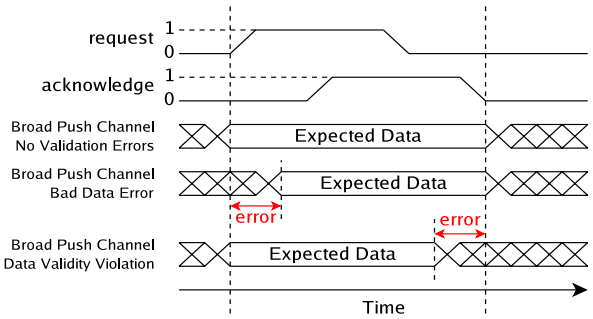


Figure 3: Illustration of bad data errors and data validity violations

4. Simulation-Based Validation

Because validation requires checking that implemented circuits behave as expected, simulation-based methods are common. They usually involve validating the implemented design against a behavioural model. The method here uses a similar approach, but relies on the properties of synthesised handshake circuits to simplify the approach.

Previous methods of asynchronous validation rely heavily on specifying communication models and signal transition graphs, rather than taking advantage of the abstraction provided by handshake circuits. Karlsen [8] and Vanbekbergen [12] present methods that require the designer to specify state graphs for each communicating element, which can be a time consuming process as designs become larger. Abstracting away the communication protocols is desired to allow the designer to focus more on the functionality of the design. The methods presented by Davies [2] and Furber [5] are similar to the method presented here and can be applied to handshake circuits, but do not actually perform behavioural validation. The methods rely on a brute-force approach that simply checks bundling constraints on the simulated design, rather than validating against a behavioural model to ensure correctness of the

design. The process of validation should be automated for synthesised handshake circuits since these circuits are very deterministic and follow specific protocols.

The method here relies on validating handshake circuits generated by Balsa [1][3], an asynchronous handshake circuit synthesis tool, but can be adopted for other design flows. Balsa takes a circuit description and automatically compiles it into a set of interconnected handshake components. These handshake components are completely deterministic, with the exception of the Arbiter handshake component (see Section 4.1). This results in each channel following an exact behaviour given a specific test pattern. The Balsa design flow incorporates a behavioural simulator that simulates the compiled handshake circuit design. The resulting behavioural simulation gives the exact sequence of events on each channel that should occur for the synthesised circuit, essentially yielding behavioural model that can be used during validation. If each channel in the synthesised design is monitored independently, then the entire circuit can be validated by performing an equivalence check on each channel between the synthesised netlist simulation and the behavioural simulation to ensure that each channel is correctly sequencing transitions on the handshake signals and the data.

The above validation method relies on sufficient coverage of the test patterns provided to the simulators to exercise the various states of the design. The design may validate for the given test patterns, but if the test coverage is limited, the circuit may still fail. As with all simulation-based methods, this method relies on good test coverage to be effective in validating whether a design works.

4.1. Arbitration

If a circuit requires an arbitrated choice, Balsa allows the designer to place an Arbiter handshake component to arbitrate the choice. This component is the only component that introduces non-determinism in a Balsa handshake circuit design. In order to deal with this problem, the non-determinism must be removed from the simulation. The proposal is to force the simulations to follow each other in order to determine whether the behavioural model and netlist design behave the same. The netlist simulation is forced to make the same arbitrated choices as the behavioural model while stalling the other choice if that input request arrives first. This will result in the netlist simulation making the same choice as the behavioural model, but only for the purpose of validating the behaviour. Once the design has been validated, the arbiters can assume their original behaviour.

5. Results

To demonstrate the effectiveness of the method above, validation tools were integrated into the Balsa design flow. They were used to identify functional validation errors in Balsa handshake component designs. Two synthesis back-ends were chosen for the tests, the broad and early implementations. The broad back-end implements a single-rail broad data validity protocol while the early back-end [6] implements a single-rail early data validity protocol. A simplified version of a synthesised ARMv5-compatible SPA core [10] was used as the test subject (called the nanoSpa). The Hello World program was chosen for the simulations because it activates a large number of channels and executes in a reasonable time. The program performs at least one handshake on a majority of the channels (as measured by the validation tools). Because the validation method above validates the functionality of handshake components, tests that explore a large state space are needed.

Upon running the program on the nanoSpa, validation errors were detected in a number of places. A major functional validation error in the composition of Balsa handshake components was identified upon running the validation tools on a nanoSpa implementation that used the conventional style option for a PassivatorPush handshake component. The PassivatorPush component is responsible for transferring data between two independent blocks in the design. The problem occurs when the output of a PassivatorPush component is connected to a Fetch component as illustrated in Figure 4. The single-rail broad implementation of these components allowed the data validity protocol to be violated due to a mismatch of delays within the Fetch component. The gate-level implementation allowed changes on incoming data to propagate to the output channel of the PassivatorPush before the Fetch component finished the handshake on its output channel. Because the Fetch component was implemented only using wires, the composition of the PassivatorPush and the Fetch caused a data validity violation on the Fetch component's output channel, causing incorrect data to be processed. The error was subsequently fixed by placing data storage within the PassivatorPush component to maintain the data validity protocol and recognising this composition in the Balsa compiler. Another solution to this error was to allow the output of the Fetch to complete a full four-phase handshake before acknowledging the activation channel by using an S-element [11].

Another major error was detected in the composition of CaseFetch and Fetch components (see Figure 4). The CaseFetch component is responsible for transferring input data from a particular channel based on an index. The composition of these two components caused the same data validity violation as above, but unlike the error above, this error was implementation specific. The error only occurred when the design was synthesised using the Xilinx back-end. The Xilinx implementation requires a different gate-

level implementation of the CaseFetch component due to the lack of keeper inverters (a state-holding element used in asynchronous designs). The CaseFetch implementation removed data from outgoing channels as soon as the outgoing acknowledge reset. This caused a data validity violation along the output channel of the Fetch component because of the wire-only implementation of this component. Data storage can be placed in the CaseFetch component to resolve the issue, or an S-element placed in the Fetch component, similar to the solution above.

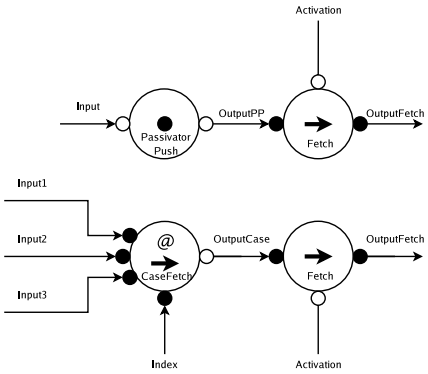


Figure 4: Hazardous handshake circuit compositions

A data validity violation was detected in the early implementation of the Variable component. The Variable component assumes exclusive writes and reads, but due to the early protocol, a write-after-read hazard may occur. When these two operations are sequenced, the return-to-zero of the read may overlap with the writing to the Variable. This may cause a race to occur, causing the output of the Variable to prematurely change. The error is fixed by recognizing this behaviour in the compiler and ensuring exclusivity, or adding an exclusive element in the Variable component.

6. Conclusions

A method of performing validation of asynchronous handshake circuits has been presented. The method takes advantage of the abstraction provided by handshake circuits and suggests performing validation at the channel level. A synthesised ARM core was used as a test subject to demonstrate the effectiveness of this method in identifying validation errors. Once these errors were identified, these parts were fixed and validated independently, then integrated back into the synthesis flow. Future work will involve running the validation suite on other designs to find and debug design errors in Balsa handshake components and improve the quality of the Balsa back-ends.

The method presented here adds value and increases confidence in the Balsa design flow. The tools mentioned above integrate with this flow, allowing back-end users to validate designs of handshake components as well as validate choices made by the compiler. Specific compositions of handshake components may cause validation errors, suggesting an error with the compiler. Thus the presented method can be used to test the compiler and increase confidence in the way it composes handshake circuits.

Although the tools integrate with the Balsa design flow, the methods above can be generally applied to other design flows that implement handshake circuits, such as the Timeless Design Environment flow provided by Handshake Solutions [7]. The method above relies on the abstraction and determinism of handshake circuits, allowing the method to be adapted to various design flows. As long as a behavioural simulation of the channel communication can be acquired and the channel activity extracted from the gate-level netlist simulation, the method above can be generally applied to other synthesis flows.

7. References

[1] A. Bardsley. Implementing Balsa Handshake Circuits. Ph.D. Thesis, Department of Computer Science, The University of Manchester, 2000.

[2] R. M. Davies, J. V. Woods. Timing Verification for Asynchronous Design. In Proceedings of EURO-DAC 96', pp. 78-83, September 1996.

[3] D. A. Edwards, A. Bardsley. Balsa: An Asynchronous Hardware Synthesis Language. In The Computer Journal, Vol. 45, No. 1: pp. 12-18, British Computer Society, 2002.

[4] C. Farnsworth, D. A. Edwards, J. Liu, S. S. Sikand. A Hybrid Asynchronous System Design Environment. In 2nd Working Conference Asynchronous Design Methodologies, May 1995.

[5] S. Furber. Validating the AMULET Microprocessors. In The Computer Journal, Vol. 45, No. 1, British Computer Society, 2002.

[6] N. Gupta. Synthesis of Asynchronous Circuits Using Early Data Validity. In Proceedings of VLSI Design 2005, pp. 799-803, January 2005.

[7] Handshake Solutions. TiDE - Timeless Design Environment. URL: http://www.handshakesolutions.com.

[8] P. A. Karlsen, P. T. Roine. A Timing Verifier and Profiler for Asynchronous Circuits. In Proceedings of ASYNC 99', pp. 13-23, April 1999.

[9] A. M. G. Peeters. Single-Rail Handshake Circuits. Ph.D. Thesis, Technische Universiteit Eindhoven, Eindhoven, The Netherlands, 1996.

[10] L. A. Plana, P. A. Riocreux, W. J. Bainbridge, A. Bardsley, S. Temple, J. D. Garside, Z .C. Yu. SPA - A Secure Amulet Core for Smartcard Applications. In Microprocessors and Microsystems, 27(9): pp. 431-446, October 2003.

[11] J. Sparso, S. Furber. Principles of Asynchronous Circuit Design. Kluwer Academics Publishers, 2001.

[12] P. Vanbekbergen, A. Wang, K. Keutzer. A design and validation system for asynchronous circuits. In Proceedings of DAC 95', pp. 725 - 730, June 1995.

Blame Passing for Analysis and Optimisation

Charlie Brej
Dept. of Computer Science, The University of Manchester, Oxford Road, Manchester, M13 9PL, UK.

cbrej@cs.man.ac.uk

Abstract

Potentially, asynchronous circuits can execute faster than their synchronous counterpart because of their average-case, rather than worst-case, performance. In practice, such an advantage is difficult to achieve. A major reason is the difficulty in identifying timing-critical regions of the circuit and analysing the results of changes to the system. The problem arises because static critical path extraction tools used by synchronous designers do not work with asynchronous circuits.

This paper introduces a novel, pragmatic dynamic timing analysis approach to determine bottlenecks in asynchronous circuits. This approach evaluates the behaviour of a circuit within a specific test-bench designed to exercise the circuit in a manner typical of its final application.

Extracted information can then be used to determine which optimisations should be applied, and where those optimisations should be applied. Circuit behaviour information can also be fed back to the designer to allow circuit bottlenecks to be visualised.

1. Introduction

There are many circuit design methodologies which do not use a global clock as a timing reference to mark the completion of operations[1]. Of these, the most relevant in this paper are circuits where the timing of each operation is not bounded but rather is implicit in the data encoding, and in particular circuits with Delay Insensitive (DI)[1] data encodings. Circuits with data dependant timing (e.g. operand dependant matched delays) or data dependant control sequences (in systems made with languages such as Balsa[2]) are also amenable to the approach described in this work, but will not be considered in depth.

Circuits with DI data encodings can have non-uniform timing which is dependant on the operation executed. This is different from synchronous circuits where every operation's execution time is bounded by the predictable clock period. This bounding leads to predictable timing which allow circuit analysis to take place in a static manner. In synchronous systems this takes the form of critical path extraction[4]. The predictability of synchronous systems, and the bounding which the clock provides, naturally leads to systems with worst case performance, irrespective of the pattern of data processed.

Static timing analysis has become the method of choice for synchronous circuit analysis as it has the advantage of high speed of analysis and complete coverage of all significant paths. The lack of simple timing references across an asynchronous circuit can make static analysis difficult. This paper attempts to use simulation and dynamic analysis to exploit the potential for average case performance[3] in asynchronous circuits, where data-dependent timing exists.

One of the design methodologies which tries to exploit average case performance is *early output* logic[5]. In this paper early output circuits will be used to demonstrate the dynamic timing analysis method and the optimisation system.

1.1. Early Output Logic

Early output logic attempts to increase performance of a system by first decreasing the latency of each stage: Through the use of 1-of-n delay insensitive codes, the completion of computations can be determined through the use of completion detection logic on the data outputs of a stage rather than an estimation of stage timing using a worst case delay model. Bit-level pipelining allows the generation of partial results which can be forwarded to the next computational stage while the remainder of the outputs are still being processed. In cases where the inputs which have arrived to a function are sufficient to generate an output, the output generation is not synchronised with the arrival of the remaining inputs. The output is generated in parallel with the gathering of the inputs to the stage. This allows *early output* [5] generation, yet correctly acknowledges all inputs to the stage even if they were late (and so unnecessary for generating the output).

Figure 1 shows a segment of an early output circuit. The communication is done across 4 wires: request zero (R0) and one (R1), validity (V) and acknowledge (A). The early output OR gate is constructed from an AND/OR pair which generate the two data output signals. The validity output from the gate is formed by gathering all validity inputs. The latch cannot acknowledge until the validity becomes high. It asserts its validity output once it is outputting data becomes valid. This style of early output circuit construction is described in more detail elsewhere[5].

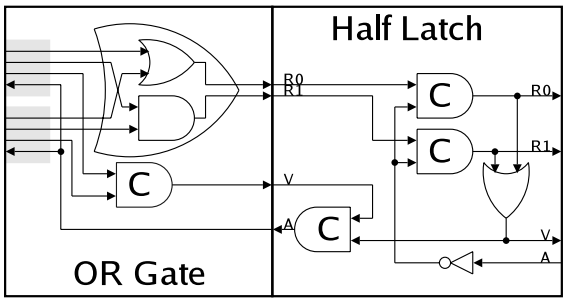


Figure 1: Example early output circuit segment

1.2. Asynchronous Circuit Construction

Most asynchronous circuits are constructed in a manner very similar to that of synchronous circuits. The circuit is composed of computational logic which takes inputs and generates outputs based on those inputs. Latches are used to store data and keep it stable while it is being processed by the combinatorial logic. The main differences in the numerous asynchronous design methodologies come from the use of differing handshaking protocols and data encodings used to co-ordinate transfer of data between latches. Each approach has its advantages and disadvantages. In particular, it is advantageous to ensure that control signalling happens with the same set of signal transitions for each transfer. The power and speed attractive two-phase protocols[6] make this difficult. The use of a 'reset' phase with four-phase protocols[8] leads to simpler circuits, but considerable effort is requires to hide the latency of the reset phase by overlapping it with other circuit activity. Encodings such as four-phase 1-of-4 encoding are popular as the circuits produced are simple and the energy efficiency of the code is good[7].

1.3. Asynchronous Circuit Properties

Early output logic tries to tackle the overheads of the four-phase 1-of-n codes when used in combinatorial logic. Unfortunately, due to the use of the four-phase protocol there remains the reset period problem. A generally accepted method of reducing the effect of the reset period is to doubling the pipelining in the system while keeping the number of tokens the same. This allows half of the logic stages to compute while the other half resets, ready to accept new values.

Fine grain pipelining is not always benefial. It can lead to an increase in the latency of data flowing through the pipeline. Other optimisations such as C-element tree balancing improve response time for each input equally. This often does not take into account the case where inputs arrive in sequence and so balancing the tree can shift the last input to arrive from a position where it was close to the output to a position further from the output.

In order to determine where these optimisations should be applied, the circuit's performance must be analysed when performing 'typical' operations.

2. Static Timing Analysis

There are some static methods which can be adapted to asynchronous circuits.

2.1. Slack matching

Slack matching [9] allows a crude balancing of the level pipelining between two paths with the same start and end points. This method adds additional pipelining latches into the path with the lower pipelining. This ensures that at the start of the fork the two paths are capable of accepting an equal number of data tokens and a stall due to one pipeline being full becomes less likely. This system makes many assumptions such as a equal execution time of each stage, a bundled data system (no bit level pipelining) and no data dependant delays. Another limiting factor is only optimisation which can be applied using this method is pipelining latch insertion.

2.2. Critical path extraction

Synchronous circuits use static timing analysis to extract the critical path and the optimisations rely on making this shorter.

The extraction of the critical path from synchronous designs uses an algorithm which finds the route and the length of the critical path [4]. This process is made simpler because of the assumption implicit in the use of clocked latches that all inputs are applied at the same time.

The algorithm marks the time of arrival of data at each point in the circuit. This is done by determining the latest arriving input to each gate and marking the output time as that time plus the delay of the gate. The outputs of the latches after the active clock edge are marked as occurring at time zero. Once this has been performed on all signals in the circuit, the signal with the latest arrival time can be found and its route between latches can be determined.

This method has several limitations: combinatorial logic loops are not permitted due to the cyclic dependencies produced, and the critical path can include more than one mutually exclusive path which gives a critical path which cannot occur.

This method is sufficient for simple synchronous circuits. Unfortunately most asynchronous circuits do not have predictable and cyclic timing and the static timing approach is not applicable.

3. Blame Passing

A method to analyse asynchronous circuits is crucial to allow the asynchronous engineer to tackle system bottlenecks. As this cannot easily be done statically, it must be performed dynamically.

3.1. Simulation

The basis of the dynamic timing analysis approach is the ability to simulate the examined circuits. In order to observe realistic operation of the unit, the circuit must be placed into a test-bench emulating the environment in which the unit would be used. Because the delays and sequencing of the operations are data dependant, the test-bench must accurately reflect the environment, otherwise the optimisations applied will be optimising the circuit to execute operations or react to environment stimuli sequences which may never occur.

The absolute accuracy of the simulator used is not important (as long as relative delays are reasonably consistently represented) and any level of simulation between behavioural models and post-layout transistor-level analogue simulations could be used. In this paper, an example fixed delay gate level simulator has been used to demonstrate the methodology. A custom gate level simulator was implemented as it is fast and it does not rely on external tool suites to generate satisfactory results for all components.

Once the circuit and the test bench have been loaded into the simulator, the simulation begins with the release of the reset signal. The simulation then continues until the benchmark has been completed. The completion of the benchmark can be signalled by raising a specific signal or it can be time bound and instead of recording the time taken to perform a set number of operations, the number of operations executed in the set amount of time is recorded.

The simulation performs two tasks: 1) measuring the performance of a proposed circuit and 2) extracting information about its behaviour in order to improve the performance further. Even inaccurate simulators, where the exact delay of each component is not known, can be used to extract reasonable comparative performance results, giving a good idea if an optimisation would have a positive or a negative effect.

3.2. Slowest Path Extraction

This paper introduces the concept of a *slowest path*. The slowest

path follows the actual sequence of transitions which accumulated into the delay of the system during the full benchmark. This is different approach from determining the critical path in a synchronous system because the analysis required to find the slowest path need not be exhaustive. Such a restricted analysis allows it to rapidly observe average case performance (rather than worst case). A slowest path is allowed to pass through any unit as many times as is required. This enables the approach to extract the path from long multi-cycle operations and thus also observe the signal interactions in latches as well as logic. The method of extracting the slowest path is loosely based on static timing analysis.

In static timing analysis each wire in the system is marked with the arrival time of the data in a worst case scenario. Once the wire with the latest arriving time has been found the critical path can be extracted. This can be done in a single pass over the circuit (marking signal times can be done at the same time as identifying the currently most critical path). For the sake of simplicity, in the following example we will presume this is done using another pass.

To extract the critical path with a known end point, a path back from that point towards the previous latch for the gate with the latest arrival point at the end point must be found. This path passed back through gates with the latest input arrival times until the output of a latch is reached. The path follows a theoretical sequence of transitions which could happen and so require the clock to have a period longer than the delay of the critical path.

In the dynamic timing analysis, the end point of a simulation run is the benchmark completion signal. In fixed time simulations any signal which transitioned towards the end of the simulation run can be picked. This end point is the last point in the slowest path. The previous point in that path can be determined by finding which was the last input to arrive to the transitioning gate. If this input would have transitioned sooner the operation would have taken less time, and so this input bears the 'blame' for the delay of the circuit. Blame passes from gate input to gate input back through the path (hence "blame passing simulation") until the initial signal is reached (usually the release of the reset).

Unfortunately, if only a single time is recorded for each gate, the cyclic nature of the slowest path will cause that value to be overwritten on each cycle of the simulation. Instead, the proposed technique generates the slowest path forward rather than in reverse during the simulation. As the simulation executes, each transition of a gate is recorded, along with its cause, as the output of the gate could become a part of the slowest path. Should its transition not cause any subsequent gate outputs to change, the transition is counted as a *dead end*. This can now be forgotten as it can not form a part of the slowest path. The recorded transitions keep a reference counter in order to allow their removal should all transitions caused by them have reached dead ends. Discarding dead ends prevents the simulation memory footprint from growing out of control.

3.3. Slowest Path Analysis

The slowest path in any simulation represents the sequence of transitions which accumulated to the complete delay of the simulation. This shows the exact points where the optimisations should be applied as applying optimisations in areas not passed through by the slowest path would not effect the path and the operation will still take the same amount of time. There are exceptions to this rule: optimisations could cause a unit, not on the path, to become slower and become a part of the path, causing the whole system to operate slower. The other exception is in gates where a number of inputs which transition can cause the output to transition. Here the easiest target to focus on is to optimise the sequence of events which led to the transitioning of the first input which triggered the gate. It is possible to also shorten the slowest path by generating a new path through an optimised unit which feeds one of the other inputs which could trigger the transition of the gate before the original first input reaches it.

The optimisation to be applied to the units passed though by the slowest path can be determined by observing the route of the path through known constructions.

4. Optimisations

To demonstrate a number of optimisations, a simple example circuit was designed. The circuit takes a number from an internal constant source and decrements it (storing the result in a register) until it reaches zero, at which point it reads a new number from the constant source. Figure 2 shows the design which consists of a constant source (Const) which feeds a number to the unit each cycle, a register (Reg) which holds the current value, a decrementer (Dec) which reduces the number by one, an OR gate which tests for the number being equal to zero and a multiplexer which picks either the new value or the external constant to be written to the register.

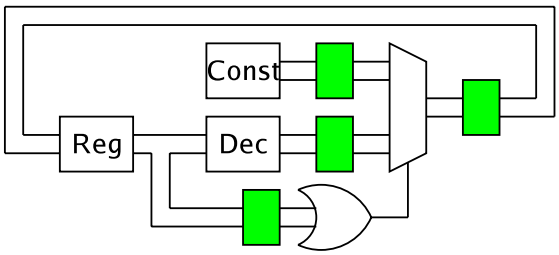


Figure 2: Decrementer circuit

The design was heavily pipelined to allow parts of the circuit to reset in parallel with others processing. The shaded blocks in the figure show the placement of half latches which increase the pipelining of the design. In addition to these, the decrementer was *vertically pipelined* to a single bit level (a half-latch placed on the carry path between each bit slice). This may seem excessive but these latches should be treated only as possible latch locations as any latches which restrict the performance of the design can be removed through one of the optimisations.

The circuit was simulated and its slowest path extracted. The simulation is set to run for a fixed time of 100 000 gate delays, after which a random signal which transitioned in the last time-slice is taken as the end point of the slowest path. The signal can be randomly picked as no matter which signal is chosen, the path, within a small number of gate delays, converges into the same route as with any other signal selected.

One of the methods the designer can use to observe the slowest path (in order to find the bottleneck in the system) is to annotate that path onto the schematic used to design the circuit or a diagram which represents the design. In figure 3 the slowest path is placed on top of a representation of the design. The arrows represent the transitions in slowest path. As the simulation executes many operations of the unit, the paths often take the same route a number of times. In the figure, the thickness of the arrow represents the number of times a particular gate crossing had occurred. Not visible

on the diagram is the distinction between the rising and falling transitions.

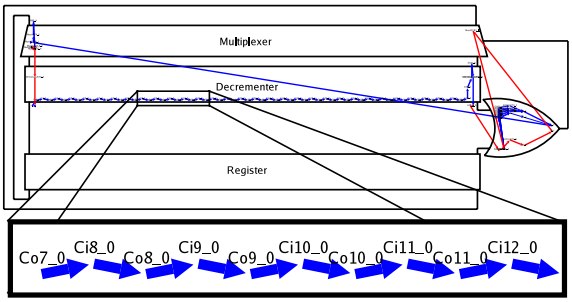


Figure 3: Slowest path in the decrementer design

The zoomed segment in the figure shows the a part of a sequence of transitions which occupy the majority of the slowest path (77%). These are falling transitions along the carry chain of the decrementer.

In this benchmark the constant, which is loaded and decremented, is large (2^{32}-1) with the simulation time comparatively small and so the decrementer never has a long carry chain dependency. The use of early output logic allows a fast generation of a result as the carry signals can be generated locally rather than needing to propagate the full length of the unit.

The generation of the result is not the bottleneck in this benchmark. Instead, the circuit takes a very long period of time to release the signals on the carry chain despite the fact the chain is broken up into small one-bit segments. The root of the problem is the construction of the half latches on the carry chain which prohibit their outputs from dropping while their inputs remain high (after the acknowledge has been applied). This dependency causes the full carry chain to reset sequentially from the bottom and ripples the release of the data signals through the entire unit. This problem can be remedied using an early-drop latch[5]. This latch drops its data outputs upon receiving an acknowledge even if the data inputs remain high.

4.1. Early Drop Latch

The application of optimisations can be described by tables. A positive effect of applying an optimisation can be predicted through the observation of a frequently occurring sequence in the slowest path passing through the element to be optimised. This path is shown in the “Pos” column in each optimisation table. As each optimisation has a possibility to cause a lengthening of the slowest path, the “Neg” column depicts the path which, if observed in the pre-optimised design, is likely to cause the optimisation to decrease the performance of the system. The “Apply” column in each optimisation table presents the optimisation to be applied. Figure 4 shows the table for the early-drop latch optimisation.

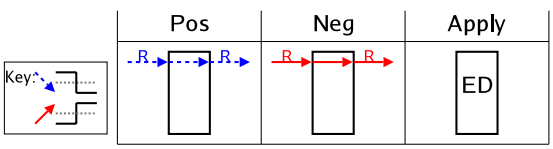


Figure 4: Early drop latch optimisation

The early drop latch releases the request (data) signal as soon as the acknowledge has been released rather than waiting for the request on the input to fall. This optimisation is particularly effective in circuits with the slowest path passing through many latches on the falling data signal transitions (an example of which is the carry chain reset in the decrementer example). The pattern to be matched for the optimisation to be effective is the down-going transition (dashed arrows) of a request out signal being dependant on the request in signal. Replacing the latch with an early drop latch would allow the release of the request out signal to be done concurrently (before the request in signal is released).

An early drop latch does have an additional delay on the rising transitions (solid arrows) of the data signal propagation and should not be used in situations where this frequently occurs in the slowest path.

4.2. Latch Removal

As mentioned before, the number of latches placed in the example design is high and many of them will have a negative effect on the performance of the design by adding latency to the slowest path. Removing a latch can reduce the cycle time by two gate delays (if the latch is on the slowest path in both the set and the reset periods). The danger in doing this is the latch may have been adding pipelining crucial to make the system free flowing. There is little way to determine which latches add pipelining which is useful to the system from the slowest path and this is why the table in figure 5 has that entry missing.

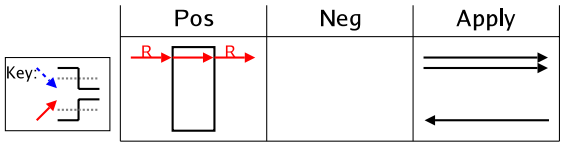


Figure 5: Latch removal optimisation

Instead the optimisation system has to rely on a simulation of the system along with the proposed optimisation to determine if it has a positive effect.

4.3. Latch Insertion

In situations where insufficient number of pipelining latches were placed in the design the optimisation system can spot where a latch is separating two regions which are unable to store two different tokens due to the latch not providing enough decoupling. Only once the stage in front has completed its phase can it allow the stage behind to enter its next phase e.g. the stage in front must complete its reset phase and accept the data from the stage behind before the stage behind can enter the reset phase and release the data. Such situations can be avoided by inserting an additional pipelining latch where the stage behind can commit its data (and enter the reset phase) before the stage in front is not ready to accept it. This can be seen in figure 6.

The danger of inserting latches is the addition of latency. Should the slowest path pass through the latch data signals, the path will become one gate delay longer for every pass.

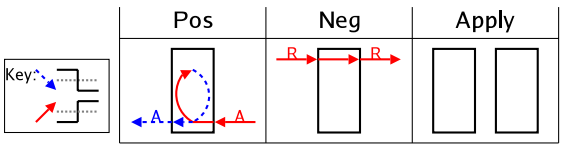


Figure 6: Latch insertion optimisation

The combination of latch removal and latch insertion effectively reproduces the effect of slack matching. The areas concentrated on by the slack matching techniques [9] (unbalanced pipelines) show up in the slowest path as recommendations to remove latches from

the over-pipelined path and to insert latches in the under-pipelined path.

4.4. Anti-Token Latch

In early output circuits, it is often possible to generate the result of a function without the presence of all inputs. Unfortunately, the late unnecessary input must be synchronised with and acknowledged to correctly group inputs. The anti-token latch [5] allows a stage to acknowledge an input which has not arrived yet. The latch then effectively holds an anti-token which propagates back through the pipeline and removes the undesired token.

The slowest path in situations where a stage waits for the token to arrive before acknowledging it passes through a latch from the data input arriving to the validity output rising, signalling the latch is ready to accept an acknowledge. The anti-token latch asserts the validity signal before any data has arrived which exposes it to receive an acknowledge before it holds any data to remove. Here an anti-token is formed and the stage becomes free to process a new set of inputs. Figure 7 shows the table for the anti-token optimisation.

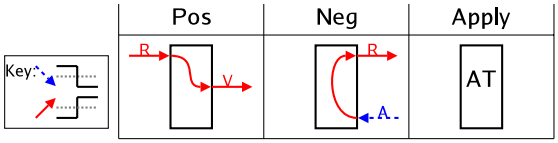


Figure 7: Anti-token latch optimisation

The anti-token latch is larger and slower in a number of transition sequences than an ordinary half latch. The negative effect box in the figure shows just one of the routes which if exist in the slowest path would gain an additional gate delay.

5. Results

To demonstrate the effectiveness of these optimisations they were applied to three circuits and the performance improvement due to each one was recorded. The performance of the original early output design is presented (labelled “Early None” in the graphs), along with the performance after the latch insertion and removal optimisations (Early Half), early drop latch optimisations (Early Drop) and anti-token optimisations (Early Anti). To give a good comparison of the performance of the resultant circuits they are compared with the synchronous equivalent (Synchronous) for which the timing is determined by extracting the critical path. The delay of the latching element and margins for clock jitter are not factored in.

Also presented is a DIMS implementation generated from the same design specification (DIMS None). The DIMS design cannot take full advantage of the early-drop and anti-token latch optimisations but the latch removal and insertion rules apply equally to this design style and the result of these optimisations is also presented (DIMS Half).

Each benchmark was run for 100 000 gate delays and the number of operations executed in that time was recorded.

5.1. Decrementer Benchmark

The decrementer circuit has already been shown, and forms one of the circuits upon which the optimisations will be demonstrated. Because the circuit has behaviour dependant on the data being processed, it is benchmarked with two different internal constant values. The ‘Zero’ benchmark sets the constant to zero, which causes it to continuously reload the constant. The ‘Full’ benchmark decrements from the maximum (32 bit) integer which causes it to never load the number form the constant. The circuits were not only benchmarked for use with the differing constant values but also optimised with them. The results for the circuit are given in figure 8.

The insertion and removal of latches optimisation in this circuit removes many of the latches in the carry path since they do not add to the pipelining of the circuit and instead add latency. This yields a 50% improvement in performance in both circuits. In the ‘Full’ benchmark, another large increase in performance is gained through the use of early-drop latches. The same effect was not seen in the ‘Zero’ benchmark as it does not suffer from the same problem. Instead, a lot of additional performance was gained through the use of anti-token latches which were able to pass anti-tokens to the decrementer once the value was loaded from the constant. This decreased the reset time of the decrementer which was a bottleneck in the performance.

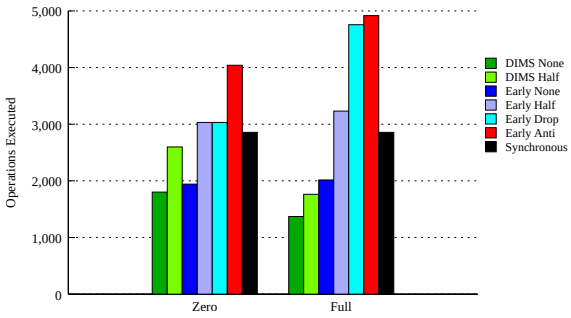


Figure 8: Decrementer benchmark results

5.2. GCD Benchmark

The GCD benchmark determines the greatest common denominator of two numbers. To keep the design simple, the numbers are restricted to 8 bits. The design comprises two dividers which only generate the remainder while discarding the result of the division, two registers to record the current number pair being worked on, a comparator which determines either of the two numbers have become zero and a pair of internal constants. The unit loads a pair of numbers from the constants and then repeatedly divides them by each other each time recording the remainder. Eventually one of the numbers reaches zero and the result is the other number. In this benchmark, the result is discarded and a new pair of numbers is loaded from the constants. The two modes of operation the design is worked on are: two numbers in the Fibonacci sequence, and two zeros. The Fibonacci sequence numbers (223 and 144) require a large number of operations before the result is generated and a new set of numbers is loaded. The zero test loads new numbers on each cycle as the greatest common denominator of two zeros cannot be determined.

The results of the optimisations on this circuit are shown in figure 9. The zero benchmark received a reasonable increase in performance due to the use of anti-token latches. These were effective at removing the results of the unnecessarily executed divisions and allowing the circuit to progress to the next phase. Because the placement of half latches was good, little performance gain is attributed to the removal and insertion of half latches.

5.3. CPU Benchmark

The CPU benchmark uses the datapath of an open source microprocessor [10]. The control signals are attached to pseudo

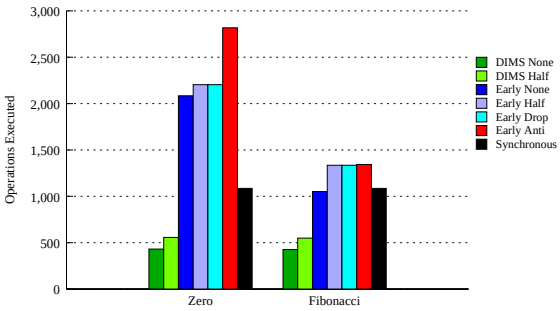


Figure 9: GCD benchmark results

random number generators which cause it to execute random instructions. The memory stage is formed from a delay which is triggered once all address inputs are present. The result of all memory operations is always zero. The delay of the memory stage is either zero for the 'Zero' benchmark or 50 gate delays for the 'Long' benchmark.

The results are shown in figure 10. Because the circuit was already relatively balanced and tuned, in the Zero benchmark none of the optimisations had a great effect. The DIMS circuit gained a reasonable performance increase due to the latch removal. In the Long benchmark, the anti-tokens were able to keep the design executing during memory accesses, the results of which were not requested by the register forwarding multiplexers. Instead, the circuit continued to execute while allowing the memory access to perform a delayed write to the register bank. The anti-token latches placed in the register bank effectively generated a register locking system where the registers which were not being written to generated anti-tokens on their inputs and continued with the next cycle of operation.

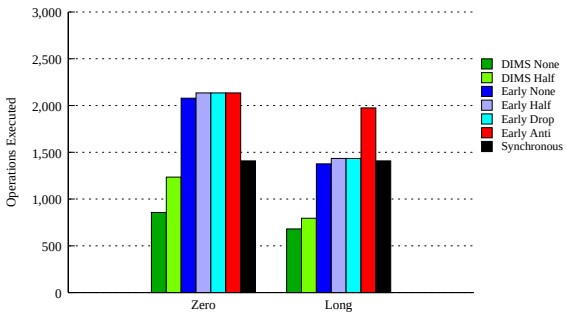


Figure 10: CPU benchmark results

6. Conclusions

Blame passing dynamic timing analysis offers an insight into the operation of a system which allows designer to make decisions about the circuit based on actual system behaviour, rather than making educated guesses about the effect of each alteration. The optimization system automates this process and allows poorly designed circuits to be balanced and offers even good designs additional performance with its use of advanced latch designs.

The blame passing extensions to the custom gate level simulator increased the simulation execution time by 30%. This is relatively small and allows the system to simulate, optimise and re-simulate in short cycles (about 3 seconds per cycle for each of the example designs).

The optimisations performed on the example designs showed cases where designers would be unaware of the real bottlenecks or of possible optimisations. The decrementer circuit was very inefficient due to its long reset time. Engineers tend to concentrate on the processing periods rather than reset periods and so an inefficiency like this would be easily overlooked. On the CPU example circuit, the addition of a register locking scheme by the designer would require a lot of additional work. A simple version of this was constructed by the optimisation system with no designer input. The optimisation system, by replacing a small number of latches, managed to construct a conceptually complex scheme which increases the system performance.

6.1. Future Work

The optimisation system is currently very specific. Only early output and DIMS designs which have been specified in a custom netlist format are allowed. Only a few optimisations have been specified and applied and the simulator can only execute in a fixed gate delay level setup. Future extensions to the system will allow different design methodologies such as bundled data pipelines (such as Micropipelines[6]) and non-pipelined approaches (such as handshake circuits [12]) to be exploited.

The simulator will be extended to read more accurate delay models of components and allow extraction of the slowest path from the event logs of other simulators. Additional optimisations will be added to the current set (stage retiming [11] and tree reshaping).

7. References

[1] J. Sparsø and S. Furber, "Principles of Asynchronous Circuit Design", Kluwer Academic Publishers, 2001, (ISBN 0-7923-7613-7)

[2] A. Bardsley, "Implementing Balsa Handshake Circuits", Ph.D. Thesis, University of Manchester, 2000.

[3] S. B. Furber, J. D. Garside, S. Temple and J. Liu. "AMULET2e: An Asynchronous Embedded Controller", Proceedings of Async 97, pp. 290-299, IEEE Computer Society Press, 1997.

[4] R. B. Hitchcock, G. L. Smith, D. D. Cheng, "Timing Analysis of Computer Hardware", IBM Journal of Research and Development, Vol. 26, 1, pp. 100-105, 1982

[5] C.F. Brej, "Early Output Logic using Anti-Tokens", Twelfth International Workshop on Logic and Synthesis (IWLS 2003), May 2003.

[6] I.E. Sutherland, "Micropipelines", The 1988 Turing Award Lecture, Communications of the ACM, Vol. 32, No 6, pp 720-738, January, 1989.

[7] W.J. Bainbridge, S. Furber, "Delay Insensitive System-on-Chip Interconnect Using 1-of-4 Data Encoding", Proceedings Async 2001, pp. 118-126, IEEE Computer Society Press, March 2001.

[8] D.E. Muller, "Asynchronous logics and application to information processing", Switching Theory in Space Technology, Stanford, University Press, Stanford, CA, 1963.

[9] Andrew M. Lines. Pipelined Asynchronous Circuits. MS Thesis, Caltech-CS-TR-95-21, 1995.

[10] C.F. Brej, "Yellow Star: A MIPS R3000 microprocessor on an FPGA", 2001

[11] S. Hassoun, C. Ebeling, "Architectural Retiming: An Overview", TAU95, November 1995.

[12] K. van Berkel, "Handshake Circuits - An Asynchronous Architecture for VLSI Programming", 1993.

Completion Detection Optimisation based on Relative Timing

A. Mokhov, D. Sokolov, A. Yakovlev
School of Electrical, Electronic and Computer Engineering, University of Newcastle
{andrey.mokhov, danil.sokolov, alex.yakovlev}@ncl.ac.uk

***Abstract*— This paper presents an algorithm for efficient distribution of completion detection blocks in a dual-rail self-timed circuit to ensure correct computation of the completion signal. Layer-wise optimisation technique is used with the width of layers selected so as to satisfy timing constraints and use the least possible number of completion detection blocks. The presented algorithm is implemented in a tool for asynchronous design flow.**

I. Introduction

Asynchronous design is commonly accepted nowadays to be better theoretically than synchronous design in terms of speed, power consumption, electromagnetic noise etc [1]. However, asynchronous design is hard to implement efficiently in practice that stops industry to accept it as a standard. One of the reasons for that is absence of a stable CAD-supported design flow. Furthermore, most of today asynchronous implementations suffer from large area and power overheads caused by mechanisms ensuring correct functionality of the circuits. The size of the mechanisms (such as completion detection logic) sometimes exceeds the size of the logic part of the circuit that eliminates any possible benefit of using asynchronous design.

Modern automated design flows often separate data and control paths synthesis [2]. Data path synthesis based on the NCL-D [3] architecture has quite a straightforward principle of work: each gate in it has its own built-in completion detection mechanism. This mechanism requires large area overheads. To improve it a more recent NCL-X architecture was introduced in [4]. Addition of signals *go* and *done* helped to decrease the area and power overhead but still each gate had to be equipped with a completion detection block to guarantee speed independence.

At the same time it has already been demonstrated that control path in asynchronous circuits can be designed using the idea of relative timing [5]. This helps to simplify the logic substantially (leading to speed and power consumption improvement) without violating the circuit functionality. While the importance of relative timing in control logic synthesis has been addressed in [5], the question about its use in data path is still open. Of course the application of conventional bundled data techniques can be considered as an example of relative timing, but this is clearly an overkill in many situations as worst case constraints prevent from exploiting data-dependence in delays as well as put large safety margins to compensate parametric variability [6]. Initial attempts to use the idea of relative timing to improve data path synthesis more gradually from speed independent design were made in [7]. This has led to techniques such as *path-wise* and *layer-wise optimisations* which are discussed in depth in Section III.

This paper presents an algorithmic development of these techniques and their automation in combination with accurate timing analysis, which gives excellent results. The presented technique reduces the number of completion detection blocks significantly that leads to decreasing area overheads. And this naturally also decreases power consumption and increases speed.

II. Background

There are two main approaches to asynchronous data path design: *bundled data* and *completion detection*. The difference between them is the way they produce signal *done* which tells the environment that combinational logic has finished computation and produced valid outputs.

A. Bundled data

The bundled data approach uses a delay buffer to produce signal *done* (see Figure 1).

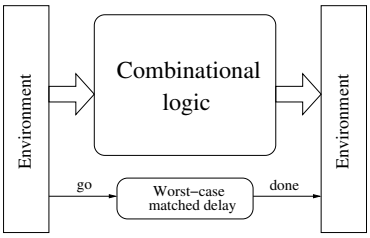


Fig. 1. Bundled data

The environment produces signal *go* to confirm that it has set valid inputs for combinational logic. The buffer delay should be selected so that for any input transition the combinational logic has finished computation before signal *done* is produced. Such a delay is called the *worst case delay* of a circuit. It should be emphasised that there are two main aspects of worst case delay: one is based on data-dependency, the other on parametric variability. So the matched delay should be sufficient to exceed the circuit computational time for any input data under any process and physical variations. The use of the worst case delay leads to an intrinsic disadvantage

of the bundled data approach: circuit performance is fixed to the worst case computational time regardless of actual input values and variations. This kills major benefits of asynchronous design - average case performance and the ability of automatic adaptation to physical properties. However, the bundled data approach is widely adopted because standard synchronous single-rail combinational logic can be used in data path without any modifications such as conversion to hazard-free logic etc. It should also be mentioned that in spite of worst case performance the bundled data approach is better than fully synchronous design in terms of speed because timing constraints are localised within a relatively smaller part of the whole system.

B. Completion detection

Completion detection methods use additional logic to detect that the circuit has actually finished computation and produce signal *done* without conservative overestimation of completion time. Completion detection is better in terms of handling variability because it does not make any assumptions on process and/or physical variations. Some of the completion detection methods are not fixed to worst-case performance and thus can exploit full speed potential of asynchronous design. But the key disadvantage of the methods is that they need a *completion detection network* and standard single-rail components cannot be directly used as they do not provide any completion information. This leads to area and power overheads and various optimisations are needed.

III. Completion Detection Optimisations

To reduce overheads of completion detection different optimisation techniques have been proposed. A method based on *partial acknowledgement* has been developed in [8]. This technique leaves a circuit to be speed independent and thus very robust under process and environmental variations.

An initial idea to use relative timing for optimisation of data path was indicated in [7]. This leads to *path-wise* and *layer-wise approaches.* They are investigated in depth in our present paper. The methods use relative timing information and therefore the resultant circuit is no longer speed independent and less tolerable to variations but timing constraints are put only on gates. This eliminates the data-dependent aspect of worst case and minimises the effect of parametric variability.

A. Path-wise optimisation

This method assumes that the longest delay of circuit stabilisation is determined by the circuit's critical path. Therefore it is enough to put completion detection blocks only on the outputs of the gates on the critical path. Unfortunately the assumption of the method is incorrect in general case as for some input transitions shorter paths can have longer delay of stabilisation.

B. Layer-wise optimisation

This technique is another possible optimisation of NCL-X architecture and is based on the following observation. A few gates just near the circuit outputs can be left without completion detection because multi-input C-element that is used to produce signal *done* is very slow. If its switching delay is larger than the delay of propagation of a codeword (and spacer) through several gates of combinational logic which produce circuit outputs, then the completion detectors for these gates are redundant. Following the same idea, after a layer of logic with completion detectors it is again possible to skip completion detection in a layer of gates whose cumulative delay is less than the delay of C-element.

A straightforward way is to keep the width of layers constant throughout the circuit but in fact careful timing analysis of a given circuit allows the width to be increased from layer to layer and eventually obtain substantial reduction to the number of used completion detection blocks. This paper presents an algorithm for refined layer-wise optimisation that uses layers whose width can potentially grow in geometric progression.

IV. Layer-wise optimisation refinement

To start the description of the algorithm two timing functions should be introduced. Let $f(C)$ denote the maximum time for a *circuit* C to stabilise on all possible input data after all inputs of C become valid. Also let $g(C)$ denote the minimum time for the *circuit* C *outputs* to stabilise on all possible input data after all inputs of C become valid. Note that $f(C)$ is responsible for the stabilisation of all the internal gates of C while $g(C)$ is responsible for the outputs of C only. Both functions should take *early propagation effect* into account. The minimum delay of the multi-input C-element will be denoted as Δ_C. Functions f and g as well as the value Δ_C are calculated given the real delay intervals of gates in the circuit C and implementation of multi-input C-element. As the number of inputs of C-element is not defined before the run of the algorithm (and therefore we cannot determine Δ_C) we first start the algorithm with number of C-element inputs to be equal to the number of dual-rail gates in the circuit which is the exact upper bound given by the NCL-X approach. After having calculated the optimised number of completion detection blocks we rerun the algorithm with corrected Δ_C and so on until this iterative process converges to the optimal Δ_C value.

Now it is possible to derive the constraint for the width of the first layer L_0 (layers are counted from the right side of the circuit). Its width is determined by the following inequality:

$$f(L_0) \leq \Delta_C$$

It is quite easy to understand the reasoning behind: the first layer L_0 has no completion detection blocks attached to it and therefore it has to stabilise completely before the completion signal from the previous layer propagates through the C-element (see Figure 2 for clarification). Notice that the multi-input C-element works in parallel with the last layer L_0 and thus its delay is not added to the overall circuit stabilisation delay.

Now we can derive a bit more complex constraint for the second layer L_1:

$$f(L_1) \leq \Delta_C + g(L_1)$$

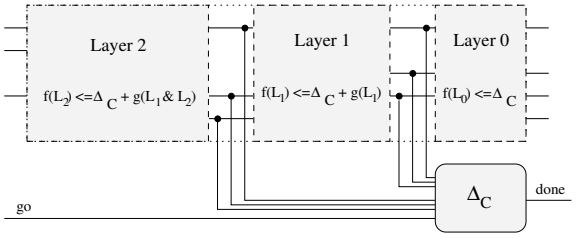


Fig. 2. Refined layer-wise optimisation

Here layer L_1 must stabilise not later than its outputs stabilise ($g(L_1)$ term) and completion detection from them passes through the C-element (Δ_C term). A naive way to compute the constraints for the next layer L_2 would result in:

$$f(L_2) \leq \Delta_C + g(L_2)\,,$$

which is similar to layer L_1. But in fact the constraint for L_2 width can be relaxed in the following way:

$$f(L_2) \leq \Delta_C + g(L_1 \cup L_2)$$

The reasoning behind this is that layer L_2 must stabilise not later than all outputs of L_1 stabilise ($g(L_1 \cup L_2)$ term) and the completion detection from them passes through C-element (Δ_C term). It is hard to realise now why there should be completion detection blocks after layer L_2 if they are not mentioned in *any* constraint at all. The reason is that these blocks are used not for correct computation of completion signal for layer L_2 but rather for correct computation of that for L_1 as they behave like signal *go* for this layer and guarantee correctness of either completion signal from L_1 or completion signal from L_2.

The general inequality for layer L_n can be now easily derived by induction:

$$f(L_n) \leq \Delta_C + g(\bigcup_{k=1}^{n} L_k) \qquad (1)$$

The above inequality implies that the width of layer L_n is not less than the width of L_{n-1} and can potentially be proportional to the sum of the widths of the previous layers. The latter comes from the fact that:

$$\max_{k=1}^{n}\{g(L_k)\} \leq g(\bigcup_{k=1}^{n} L_k) \leq \sum_{k=1}^{n} g(L_k)$$

The exact growth factor is determined by particular circuit but it can be estimated to be greater than one. That can optimise the number of used completion detection blocks significantly especially for large circuits.

V. Algorithm for layer-wise optimisation

Algorithm for the refined layer-wise optimisation is shown in Algorithm 1. Its complexity is $O(nL + m(\log(n) + n_o))$, where n is number of dual-rail gates in the circuit, m - number of wires, L - number of obtained layers and n_o - number of outputs of layer L_1 . It can be assumed (and that was the case for the circuits in our benchmarks) that in an average circuit L and n_o are proportional to $\log(n)$ and therefore the algorithm complexity can be rewritten in a more compact way: $O(m\log(n))$.

Algorithm 1 Refined layer-wise optimisation

```
Given : G - set of dual-rail gates of the circuit
Result: L - set of layers for completion detection

n=0; // current layer
while(G ≠ Ø)
{
  Let Q = ⋃_{k=1}^{n} L_k;

  <Select dual-rail gate x ∈ G such that:>
    1) all outputs of x are in Q ∪ L_0;
    2) f(x ∪ L_n) − g(x ∪ Q) is minimised.

  if f(x ∪ L_n) ≤ Δ_C + g(x ∪ Q) then
  {
    <Add x to the current layer L_n>;
    <Update functions f and g for gates in G
    that belong to fanin of x>;
    <Delete x from G>.
  }
  else
  {
    <Update functions f for all the gates in G>;
    n++; // Layer L_n is finished.
  }
}
```

VI. Experimental results and discussion

The presented algorithm was implemented in a tool to be used within the existing design flow for data path synthesis [7]. It has been tested on several real and artificially created dual-rail circuits modelled using AMS-0.35μ gate library. The experimental results are shown in Table I. The benchmarks were selected to show the effect of the proposed technique on circuit optimisation for different types of circuit structure. We used a collection of different AES S-box implementations, several small circuits (aes_multiplier, ISCAS85 benchmark circuits C-449, C-1908) and huge circuits generated specially and having different repetitive structures (1024 bit ripple-carry adder, inverter matrix and triangle) which were useful as a stress timing test for our tool.

As can be seen from the table the algorithm returns significantly fewer completion detection (CD) blocks than conventional NCL-X. Coupled with the fact that for a smaller number of CD blocks a multi-input C-element occupies much less area it yields an average of 80% in CD area savings. This save of CD area can result in the whole circuit area reduction of 40% on average. Note that these results are extremely *pessimistic* in sense that we used the intervals of gate delays in a very conservative way: we took minimum and maximum delays of a gate under all physical and fanout conditions. So, for example, a maximum delay of an inverter was more than four times higher than the minimum one. But these margins can sometimes be reduced taking into account a real gate working mode.

The algorithm works better on circuits with repetitive structure such as adders, comparators etc. However, the circuits that have more complicated structure, with paths of various lengths, slow down the growth of layers thus leading to an

TABLE I

EXPERIMENTAL RESULTS

module	CL gates	CL trans	conventional NCL-X			optimised NCL-X			save		CD extra delay	comp. time (ms)
			CD count	CD trans	total trans	CD count	CD trans	total trans	CD trans	total trans		
1024_bit_adder	8188	63456	2048	20502	83958	11	130	63586	99%	24%	0.2%	1753
aes_multiplier	342	2258	171	1738	3996	63	654	2912	62%	27%	37%	21
sbox_computed	928	4628	425	4274	8902	86	886	5514	79%	38%	9%	100
sbox_kasumi	688	3156	344	3466	6622	98	1002	4158	71%	37%	34%	54
sbox_no_pipeline	516	3084	257	2598	5682	50	518	3602	80%	37%	11%	54
sbox_oc_balanced	1166	6672	582	5846	12518	170	1722	8394	71%	33%	26%	132
sbox_oc_unbalanced	960	5776	480	4822	10598	153	1554	7330	68%	31%	26%	114
spinv_matrix_10000	20000	40000	10000	100034	140034	190	1926	41926	98%	70%	0.9%	3045
spinv_triangle_100	10100	20200	5050	50530	70730	647	6486	26686	87%	62%	10%	1162
circuit_1908	614	3646	307	3098	6744	79	806	4452	74%	34%	22%	62
circuit_449	480	2896	232	2338	5234	65	670	3566	71%	32%	30%	34
average									**80%**	**40%**	**19%**	

TABLE II

AREA COMPARISON OF LAYER-WISE OPTIMISATION WITH OTHER TECHNIQUES

module	NCL-D	RDL	PA	NCL-X	optimised NCL-X					
					pessimistic		medium		optimistic	
					count	save	count	save	count	save
sbox_computed	14180	6860	5386	8902	5514	38%	5442	39%	5050	57%
sbox_kasumi	11020	5800	4540	6622	4158	37%	3898	41%	3722	56%
sbox_no_pipeline	9804	4663	3500	5682	3602	37%	3534	38%	3270	58%
sbox_oc_balanced	36644	15804	9924	12518	8394	33%	8122	35%	7646	61%
sbox_oc_unbalanced	35720	15071	9188	10598	7330	31%	6958	34%	6682	63%
circuit_1908	11828	5737	4120	6744	4452	34%	4240	37%	3848	57%
circuit_449	8094	4171	3326	5234	3566	32%	3310	37%	3126	60%
average						**35%**		**37%**		**59%**

almost constant width of layers. Nevertheless in the worst of the observed benchmarks the algorithm uses only about 36% of completion detection blocks in comparison with NCL-X design - see aes_multiplier. It should be mentioned that this circuit is the smallest amongst the benchmarks and this might be the reason for the worst result: the width of layers did not have a chance to grow enough as only 3 layers were obtained. Another interesting observation is that the results of the algorithm are very much dependant on the initial single-rail implementation of the circuit. For example 5 different implementations of the same AES S-box gave results that differ greatly in the number of used layers (4 vs. 11) and the percentage of CD blocks reduction (68% vs. 82%). This shows that to achieve better results it might be useful to tune synthesis tools to work together with our tool.

Table II shows the comparison of the obtained results with NCL-D, RDL, partial acknowledgement (PA) and NCL-X designs in terms of area. Note that in addition to pessimistic results from table I we added *medium* and *optimistic* columns which are obtained by shrinking the gates delay margins in 30% (medium) and 70% (optimistic). One can see that even pessimistic results are much better than NCL-X and comparable with PA method.

Careful observation of the structure of the obtained distributions of CD blocks shows that the layer-wise technique has intrinsic limitations and no further improvements are possible. The only way to get results of new quality would be, instead of layering, to solve the problem in general, i.e. to select sets of acknowledged gates without strong geometrical bias. But this, at present, remains a difficult problem if one seeks for a computationally efficient algorithm.

VII. CONCLUSIONS

A new algorithm and tool for dual-rail data path optimisation is presented. The key feature of the technique is that it is based on relative timing characteristics of circuit paths and makes use of delays in signal propagation through layers and a multi-input C-element. The presented algorithm for layer-wise optimisation is quite fast that allows it to be used for large circuits. It is very pessimistic however and here is the main direction of the future research.

REFERENCES

[1] J. Sparsø and S. Furber, *Principles of Asynchronous Circuit Design: A Systems Perspective.* Kluwer Academic Publishers, 2001.

[2] D. Sokolov and A. Yakovlev, "Clock-less circuits and system synthesis.," in *IEE Proceedings, Computers and Digital Techniques*, 2005.

[3] K. Fant and S. Brandt, "Null conventional logic: A complete and consistent logic for asynchronous digital circuit synthesis," in *Proc. Int'l Conf. Application-Specific Systems, Architectures, and Processors (ASAP 96)*, 1996.

[4] A. Kondratyev and K. Lwin, "Design of asynchronous circuits using synchronous CAD tools," *IEEE Transactions on Computers*, 2002.

[5] K. Stevens, S. Rotem, S. Burns, J. Cortadella, R. Ginosar, M. Kishinevsky, and M. Roncken, "CAD directions for high performance asynchronous circuits," in *Proceedings of Design Automation Conference (DAC'99)*, pp. 116–121, 1999.

[6] "International technology roadmap for semiconductors (ITRS'05)," 2005.

[7] D. Sokolov, *Automated synthesis of asynchronous circuits using direct mapping for control and data paths*. PhD thesis, University of Newcastle upon Tyne, 2005.

[8] Y. Zhou, D. Sokolov, and A. Yakovlev, "Cost-aware synthesis of asynchronous datapath based on partial acknowledgement," tech. rep., University of Newcastle upon Tyne, 2006.

Comparative Analysis of Stuck-at Test Generation for Asynchronous Speed Independent Circuits

D.P.Vasudevan
D.P.Vasudevan@sms.ed.ac.uk
School of Informatics,University of Edinburgh, Edinburgh,EH9 3JZ, UK

A.Efthymiou
aefthymi@informatics.ed.ac.uk
School of Informatics,University of Edinburgh, Edinburgh,EH9 3JZ, UK

Abstract— Automatic Test Pattern Generation for asynchronous circuits have been considered one of the primary areas to be probed for advancing asynchronous design research. Absence of global clock in these types of circuits makes testing difficult. This paper analyzes the stuck at fault test generation of the asynchronous speed independent circuits based on two different approaches namely scan latch insertion and state transition graph based test generation. Preliminary steps involved in each approach are briefed and their effective test figures are compared.

I. INTRODUCTION

Synchronous circuit design has been considered the standard for industrial practice due to the availability of advanced CAD tools and testing strategies available. At deep submicron levels, global clock synchronization, power consumption and noise factors are affecting the design performance. Asynchronous circuit based design is gaining its momentum currently over its synchronous counterpart. On the other hand, asynchronous circuits need thorough research on CAD tool development for the whole design flow with test generation. Clock-less design paves certainly an alternative and effective way for efficient design with less power, less noise and without clock synchronization problem. Asynchronous designs are further classified into speed independent, delay insensitive, quasi delay insensitive circuits [1], [2]etc. Thus it has different models and architectures to be designed with and each of them has its own circuit models and delay assumptions. Significant efforts have been taken to develop CAD tools for synthesis of asynchronous circuits which lead to several tools available for the same like Petrify[9], Tangram [3] etc.,. Currently, very few tools are available for test generation for asynchronous circuits. Testing is essential for the designed systems, as the fabrication and component aging will cause defects in the designs. This paper deals with the analysis of two approaches for test pattern generation of asynchronous circuits. The first approach is using a symbolic method based on state traversal, while the second one is based on an adaptation of the well-known scan insertion technique.

Section 2 introduces the basic concepts of Petri nets. Section 3 describes the State Transition Graph (STG) based automatic test pattern generation. Section 4 describes the test pattern generation based on the scan insertion technique. Section 5 gives a comparison of test generated by two approaches for a number of small benchmarks. The paper is concluded at section 6.

II. PETRI NETS

A Petri net [6] is a compact model to represent concurrent systems. A Petri net is a quadruple $N = \{P, T, F, m_0\}$, where P is a finite set of places, T is a finite set of transitions, $F \subseteq (P \times T) \cup (T \times P)$ is the flow relation, and m0 is the initial marking. A transition $t \in T$ is enabled at marking m_1 if all its input places are marked. An enabled transition t may fire, producing a new marking m_2 with one less token in each input place and one more token in each output place. A free choice Petri net (FCPN) where the value changes on input, output or internal signals of the specified circuit are the interpretation of the transitions.

A. Signal Transition

STG is an interpreted FCPN introduced by Chu [4] for representing asynchronous control circuits. It is a quadruple $\{T, P, F, m_0\}$, where T is a set of transitions described by *a* x {+, -}, where *a*+ represents a 0 to 1 transition on signal *a* and *a*- represents a 1 to 0 transition, P is a set of places which can be used to specify conflict or choice. F represents flow transition relation between transitions and places: $F \subseteq (T \times P) \cup (P \times T)$. M_0 is the initial marking. An example of STG is shown in Fig. 1.

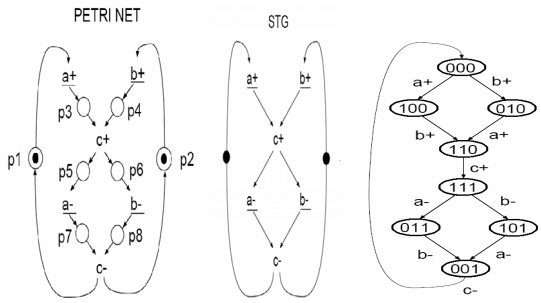


Figure 2 Petri net, STG and State graph for C- element

B. State Graph

A state graph [6] (Fig. 2) is a finite automaton given by $G = \langle A, S, T, \delta, s_o \rangle$, where $A = A_I \cup A_O$ is the set of input and non-input (output and internal) signals such that $A_I \cap A_O = \varnothing$, T is a set of signal transitions, each transition can be represented as $(+a_i,j)$ or $(-a_i,j)$ for the j-th $0 \rightarrow 1$ or $1 \rightarrow 0$ transition of signal a. $\delta : S \times T \rightarrow S$ is a partial function representing the transition function such that if $\delta(s,t) = s'$, then

signal t is said to be enabled and it takes the system from state s to s'. $s_0 \in S$ is the initial state. Each state in the state graph is labeled with a binary vector according to the signal values of the system at that state.

III. SYNCHRONOUS TEST PATTERNS FOR ASYNCHRONOUS CIRCUIT TEST GENERATION

This section briefs the approach of automatic test pattern generation used in [7]. It proposed a testing strategy with following features:

- The behavior of the asynchronous circuit is modeled as a synchronous finite state machine.
- Test patterns are generated using symbolic technique from the modeled FSM.

Test patterns can be synchronously applied to the asynchronous circuits and faults are made available at the output. An asynchronous circuit in this approach is modeled as an interconnection of gates and delay elements. The delay model used here is unbounded gate delay model [5].

A. Definitions

A state graph (SG) is a pair <S,E>, where s is the set of states and $E \subseteq S \times S$ is the set of edges (transitions).

A circuit state graph (CSG) is a 7-tuple <S,E,P,G,S_o,λ_P,λ_G> where <S,E> is a State Graph, P = {p_1....p_m} is the set of primary inputs, G={g_1....g_n} is the set of gates and $S_0 \subseteq S$ is the set of initial states. The labeling functions λ_P: S$\rightarrow \{0,1\}^m$ and λ_G: S $\rightarrow \{0,1\}^n$ map each state s with binary vector consisting of the values s of primary inputs and gates respectively. The next state of a circuit under unbounded gate delay model depends on its present state. A gate is said to be excited if its output differs from the function it implements and stable other wise. A next state function δ: SxG$\rightarrow$ S can be defined for each gate. Function δ(s, $g_{i)}$ returns either the state reached by switching the output of g_i if it is excited or s if g_i is stable. A transition relation, R relates pairs of predecessor/ successor states. If state s' is an immediate successor of state s, it will be assumed that both states are in relation R, denoted sRs' or $(s, s') \in R$.

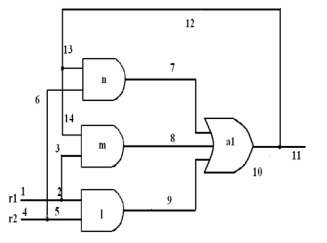


Fig. 3 Majority gate based C-element

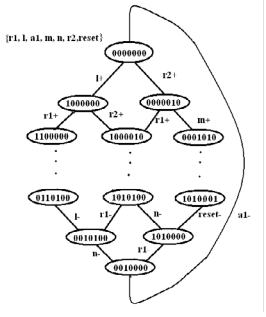


Fig. 4 State graph for the circuit model of Fig. 3

By using the next state function of each gate, the transition relation associated with circuit gates were defined as:

$$R_\delta = \{(s,s') \in S \times S \mid (s \text{ is stable} \wedge s = s') \vee (\exists g_i \in G) \text{ such that } s' = \delta(s, g_i) \neq s)\}$$

For each pair $(s,s') \in R_{\delta,}$ if s is stable, its successor is the same s, otherwise the successor is obtained by switching an excited gate. The transition relation associated to input signals were defined as follows:

$$R_I = \{(s,s') \in S \times S \mid s \text{ is stable} \wedge \lambda_p(s) \neq \lambda_p(s') \wedge \lambda_G(s) = \lambda_G(s')\}$$

Thus the transition relation of the circuit in test mode is defined as $R = R_I \cup R_{\delta.}$

B. Synchronous Abstraction of the Circuit State Graph:

To calculate the synchronous abstraction of the testable Circuit State graph, the pairs of states (s,s') such that s' is reached from s at the end of the test cycle is defined. Each pair has an associated input pattern based on the different values of inputs in s and s'. Set of all these pairs were called Test Cycle Relation (TCR). For practical reasons it was assumed that the circuit must settle in at most k transitions. The k-step test cycle relation (TCR^k) represents the pairs(s,s') distant atmost k transitions. TCR^k for a given CSG in test mode <S,E,P,G,S_o,λ_P,λ_G> is defined as:

$$TCR^k = \{(s,s') \in S \times S \mid \exists\ s_1.....,s_k \text{ such that } s\ R_I\ s' \wedge ({}^k\Lambda_{i=2}\ s_{i-1}\ R_\delta\ s_i) \wedge s_k = s'\}.$$

The next step involved removing invalid pairs of states. Vectors causing non-confluence are detected if pairs (s, s') and (s, s^N) such that s' and s^N have the same input values exist. Patterns producing oscillation or unacceptably long test cycles are found if s' is unstable. The k-Confluent Stable Stable State Graph, denoted as $CSSG^{k,}$ is formed by those pairs in TCR^k that present neither non-confluence nor cause the circuit to be unstable after k transitions. Formally it was defined as, $CSSG^k$ = $\{(s,s') \in TCR^k \mid s' \text{ is stable} \wedge \exists (s,s') \in TCR^k$ such that $[s' \neq s^N \wedge \lambda_l(s') = \lambda_l(s^N)]\}$. Thus each one of the $CSSG^k$ 'S nodes represents a stable state. An arc between two nodes s and s' exists if s' stable and the only state reachable from s in at most k transitions by applying some input pattern.

An example to show the approach of the above theory is given below using the C-element, implemented by a majority gate, shown in Fig. 3. The C-element shown is a model with two input signals r1 and r2 and four gates. The circuit state graph modeled for this circuit is a 7 tuple <S,E,P,G,S_o,λ_P,λ_G>, where <S,E> is a State Graph, P = {r1,r2,reset} is the set of primary inputs (the reset signal is added by the testify tool which initializes any memory element in circuit), G={l,m,n,a1} is the set of gates and $S_o \in$ S is the set of initial states. The labeling functions λ_P: S $\rightarrow \{0,1\}^3$ and λ_G :S $\rightarrow \{0,1\}^4$ map each state s with a binary vector consisting of the values s of primary inputs and gates respectively. Thus the elements of set, S (set of reachable states) has a binary vector of length 7. Totally 128 states forms the set S. The reachable states can be calculated by using a symbolic traversal algorithm like the one used in [10]. The set $E \subseteq S \times S$ for this circuit is obtained by enumerating over the 128X128 states. The next state function for each gate defined for this circuit are (δ_l: Sx l $\rightarrow$ S), (δ_m: S x m$\rightarrow$ S), (δ_n: S x n$\rightarrow$ S), (δ_y: S x y$\rightarrow$ S) which are operated over the gates l, m, n and y respectively. From this circuit state graph model and next state functions, the transition relation $R = R_I \cup R_{\delta,,}$ which forms a set of stable state pairs are obtained. Then synchronous abstraction involving computation of TCR^k and

$CSSG^k$ is made. The state graph evaluated for this circuit model is as shown in the Fig. 4. Testify generated 34 edges which form the transition relation between the states. For clarity, only part of the state graph is shown. After several iterations, the set of stable state pairs are ready for the test generation. With above obtained set of stable states, test pattern generation was performed using three phases: fault activation, state justification and state differentiation. Detailed briefing on these three phases can be obtained from [7]. The test generation is carried out using Random TPG and Ternary simulation [7]. The stable state pairs picked for test generation for this circuit are (s1, s127), (s127, s1), (s2, s3), (s127, s89), (s64, s65), and (s127, s22). The encoded binary codes on these state pairs correspond to the test patterns covering 24 fault sites were generated. The test patterns obtained for this circuit are (0000001, 1111111), (1111111, 0000001), (0000010, 0000011), (1111111, 1011001), (1000000, 1000001), (1111111, 0010111). The size of the test pattern was 7, which is equal to the size of the binary encoded state variables in the state pairs. 12 patterns were generated for 24 faults. To validate the approach several benchmarks synthesized by petrify were tested and the results are analyzed in section V.

IV. SCAN INSERTION BASED TEST PATTERN GENERATION

This section briefs the test pattern generation based on scan latch insertion [8]. Asynchronous circuits can be represented as combinational blocks with feedback loops as shown in Fig. 5.a. Effective test pattern generation involves breaking these feedback loops and insert scan latches at these loops, thus making it completely combinational. Level sensitive latches are used as it restores the asynchronous operation during the normal mode of operation by keeping them transparent. The loops may be global or local feedback loop. In test mode, the asynchronous circuit operates synchronously with the scan latches are fed in with test patterns and the outputs are scanned out as shown in Fig. 5.b.

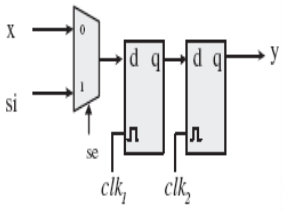


Fig. 6 LSSD Latch [8]

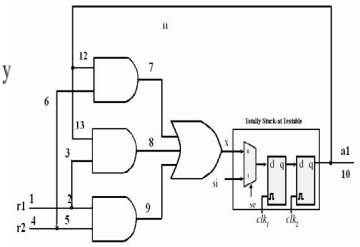


Fig. 7 C-element with the LSSD Latch

The LSSD scan design [8] is shown in Fig 6. It was designed with a 2:1 multiplexer and two latches and operates using 2 phase level sensitive clocks. The signals 'x' and 'y' provide the path for normal operation of the circuit. The signals si and y form the test mode path. This design is fully stuck-at testable. Several optimized circuits [8] are possible for the scan latch design inserted at the feedback loop of the C-element. The simplest and robust scan design is shown here. The scan mode is used for several cycles to apply the test patterns to the scan latches. The scanned output reveals the potential faults in the design. To illustrate this approach, again a majority gate based C-element is considered. The circuit consists of 2 input signals r1 and r2 with the output signal a1. Thus the LSSD Latch is inserted at the node 10 to break the feedback [8]. The modified circuit is shown in Fig. 7 The test generation for the modified circuit can easily be carried out by using standard test pattern generation tools. This is an important aspect of this method since such tools are fast, reliable and produce high-quality test patterns.

The above approach can be automated on the whole and is summarized as follows:

1) Read in the design net-list
2) Remove local loops by adding scan latches for each C-element (if present).
3) Break the global feedback loops
4) Insert the proposed scan latch at the feedback loop points
5) Generate the modified net-list of the original design file with local and global loop scan insertion.
6) Apply the net-list to the ATPG tool to generate the test patterns.

The fault coverage obtained over different benchmarks by using this method in comparison with that obtained using the symbolic technique is discussed in the next section.

V. COMPARISON OF RESULTS

This sections compares the results of the two approaches described earlier obtained by applying them to few benchmarks synthesized from petrify which is widely used in the asynchronous community [9]. The fault coverage and test patterns based on first approach was generated using the tool testify [7] which is developed from the same approach.

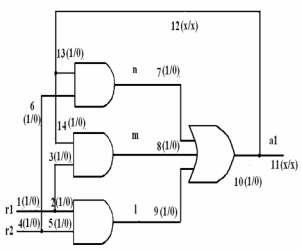


Fig. 8 Fault sites for the C-element covered by testify

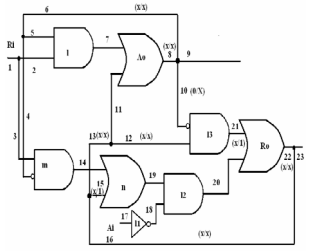


Fig. 9 Faults covered by testify for *half*

For the C-element, the faults covered by testify are 24 out of 28 faults as shown in Fig. 8. As it is evident from the figure, testify generated tests based on the primary input and the gates. So it could not detect the faults at the nodes 11 and 12 which are represented by (x/x). Even though, the test at 10 covers the fault 11, it does not cover fault at 12. The output of the gate a1, node 10 was taken into account as a single node which comprises of nodes 10, 11 and 12. But the fan-out nodes (13 and 14) from 12 are considered test nodes as they form the input for the gates n and m respectively.

Testify generated 12 test patterns of length 7 covering 24 fault sites in the circuit. The test patterns should be applied synchronously to stabilize the circuit at each pattern interval. Similarly for the benchmark *half* (Fig. 9), the faults covered by testify are only the inputs and outputs signals of all the gates. For this benchmark, even 5 more faults at input/output fault sites namely 10(0/x), 12(0/1), 15(x/1), 21(x/1) were not

detected by testify. Other intermediate node fault sites include 6(x/x), 13(x/x) 16(x/x), 22(x/x). Testify generated 24 patterns of length 11. From the above discussed results, it is evident that any new test generation algorithm to be developed should focus on testing the intermediate nodes which will be overseen by the circuit models which are modeled with only the input and output signals of each gate. Table 1 gives the fault coverage obtained for several benchmarks using testify.

Table 1: Fault coverage using symbolic technique

Testify	No of patterns	No of Vectors	Fault Coverage	No of faults	
				Total	Detected
C-element	10	7	85.71%	28	24
half	11	5	67.39%	46	31
chu172	6	8	100.00%	26	26
hazard	5	8	100.00%	28	28

Table2: Fault coverage using scan insertion technique

Scan insertion	No of patterns	No of Vectors	Fault Coverage	No of faults	
				Total	Detected
C-element	6	7	92.86%	28	26
half	6	7	100.00%	46	46
chu172	7	10	100.00%	36	36
hazard	12	7	100.00%	46	46

The fault coverage and test patterns based on the second approach was generated by cutting the global loops manually and by inserting the scan latch at the feedback paths. After inserting the latches, the netlist was fed to the Synopsys Tetramax ATPG tool to generate the test patterns and calculate the fault coverage. Table 2 gives the fault coverage for the same benchmarks and summarizes the test patterns generated using the scan insertion method. The difference in the total number of faults compared to the previous approach is due to the addition of scan latches, which increases the number of primary inputs and fault sites. Test pattern generated using the first method seems to be expensive in terms of number of test patterns and provides lower fault coverage than the second method. Also it generates longer test vectors compared to that of scan insertion approach. With the increase in test vector and number of pins the test patterns can be further reduced by using partial scan design instead of full scan. It also reduces the area overhead due to these scan latches. Another advantage of the second approach is that currently available synchronous test pattern generation tools can be used to generate test patterns, thereby makes this approach for testing asynchronous circuits industrially feasible.

VI. CONCLUSIONS

The test pattern generation for the asynchronous circuits using two different approaches, was discussed in this paper. The basic steps involved in each method were discussed. The scan latch overhead used in the scan insertion method provides a better fault coverage compared to the symbolic technique. But for area sensitive designs symbolic technique based test pattern generation seems to be promising with compact design. To balance the area overhead introduced by the scan insertion approach, partial scan based design can be introduced, which reduces the number of memory elements inserted and also the area overhead. As the continuation of this analysis, future work will be focused on developing algorithms for a partial scan insertion based test pattern generation. To improve the test generation based on STG, the fault model used should be improved from stuck-at fault model to transition fault model which improves the total number of faults detected. Several global feedback breaking algorithms and synchronous sequential testing based algorithms have to be adopted to develop algorithm for asynchronous sequential test generation.

REFERENCES

[1] Charles E. Molnar, Ting-Pien Fang, and Frederick U. Rosenberger. Synthesis of delay-insensitive modules. *In Henry Fuchs, editor, Proceedings of the Chapel Hill Conference on VLSI*, pages 67--86, 1985

[2] Alain J. Martin. Programming in VLSI: From communicating processes to delay- insensitive circuits. I*n C. A. R. Hoare, editor, Developments in Concurrency and Communication, UT Year of Programming Series*, pages 1--64. Addison-Wesley, 1990.

[3] K.van Berkel, J. Kessels, M. Roncken, R.Saeijs, and F.Schalij. The VLSI-programming language Tangram and its translation into handshake circuits. *In Proc. European Conference on Design Automation*, pages 384-389, 1991.

[4] T.-A. Chu. Synthesis of self-timed control circuits from graphs: An example. In Proc. of the IEEE International Conference on Computer Design, pages 565--571, October 1986.

[5] J. A. Brzozowski and C.-J. H. Seger. *Asynchronous Circuits*. Monographs in Computer Science. Springer-Verlag, 1995.

[6] A. Yakovlev, L. Lavagno, and A. Sangiovanni-Vincentelli. A unified signal transition graph model for asynchronous control circuit synthesis.In *Proc. of the IEEE/ACM International Conference on Computer Aided Design*, pages 104–111. IEEE Computer Society Press, Nov. 1992.

[7] O. Roig, J. Cortadella, M. Pena, and E. Pastor. Automatic generation of synchronous test patterns for asynchronous circuits. In *Proceedings of the 34th Design Automation Conference*, pages 620–625, Anaheim, CA, June 1997.

[8] F. te Beest, A.Peeters, A multiplexer based test method for self-timed circuits, Asynchronous Circuits and Systems, 2005. ASYNC 2005. Proceedings. 11th IEEE International Symposium on Volume , Issue , 14-16 March 2005 Page(s): 166 – 175

[9] PetrifyTool: http://www.ac.upc.es/~vlsi/petrify/petrify.html

[10] [10] J. R. Burch, E. M. Clarke, D. E. Long, K. L. McMillan, and D. L. Dill. Symbolic model checking for sequential circuit verification. IEEE Trans. on CAD, 13(4):401–424, 1994.

On-FPGA Communication: An Opportunity for GALS?

Terrence S.T. Mak, Peter Y.K. Cheung, Pete Sedcole

Department of Electrical and Electronic Engineering
Imperial College London
{t.mak, p.cheung, pete.sedcole@imperial.ac.uk}

On-FPGA communication is important to provide high bandwidth and reliable data transfer between coarse-grained modules, and is therefore fundamental to overall FPGA-based system performance. Recently, pre-fabricated coarse-grained modules including microprocessors, DSP units and memory modules are immersed into the fine-grain programmable fabric to provide significant improvements in computation speed, hardware area as well as hardware configuration time. However, as the number of the coarse-grained modules increases, the available communication bandwidth between these modules becomes a critical concern in system design. Furthermore, capacity of FPGA is increasing rapidly and it is becoming more difficult to distribute a global clock signal across entire chip in a single clock cycle. It is essential to partition FPGA into multiple clock tiles and provide reliable asynchronous channels for communication across multiple clock domains.

Currently, bit-level interconnect-fabrics are the only available resources to create point-to-point communication between both Look-Up-Tables (LUTs) and coarse-gained modules. As a result, on-FPGA communication architectures can only be implemented by joining the short wire segments together using programmable switches. These programmable switches will significantly contribute to overall circuit delay, power consumption, hardware area and will introduce extra performance overhead. There were several attempts to improve the interconnect efficiency by modifying the bit-level interconnect architecture, such as by removing some of the switches and introducing hardwired junctions, introducing extra interconnects to decrease the number of hubs between neighborhood logics and implementing other conventional hardware layout techniques, such as buffer insertion and transistor sizing. Nonetheless, bit-level interconnect-fabric is still the sole resources for implementing on-FPGA communication architectures.

Alternatively, to improve the on-FPGA communication efficiency, we can embed a dedicated communication infrastructure with multiple-bit or even packet-based communication channels into the FPGA fabrics. The embedded communication network would provide high performance and asynchronous communication between coarse-grained modules. FPGA and the communication network will be properly interfaced with pre-fabricated network switches, which will also provide efficient data arbitration and routing for the communication network (See Figure 1). The embedded communication network would also provide reliable asynchronous communication between different FPGA synchronous tiles with different clocking frequency to form a FPGA-based Globally Asynchronous Locally Synchronous (GALS) system.

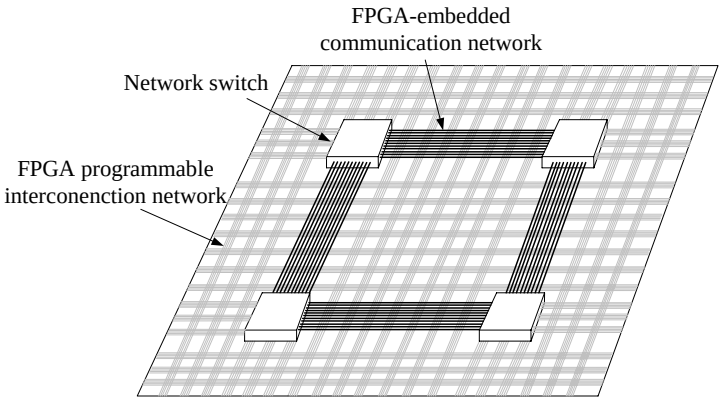


Figure 1. On-FPGA communication architecture consists of the FPGA-embedded communication network and the FPGA programmable interconnection network

An important aspect of the FPGA-embedded communication network design is to determine its topological structure, which is fundamental to the computational performance and hardware resources utilization. The topological structure design is complicated by large potential applications of FPGA. Data traffics are difficult to predict with the configurable hardware fabrics and software contents. A proper formulation and systematic evaluation methodology for the embedded communication network design is crucial. Furthermore, the design of the asynchronous/synchronous interface at network switches and the FPGA-based GALS implementation will be challenging issues. In this talk, we will first present our recent work on the FPGA-embedded communication network topology design based on a tile-based FPGA communication model. Then we will discuss some open questions regarding to the FPGA-based GALS architectures.

Metastability in FPGA Devices

N.Minas, D.J.Kinniment, G.Russell, A.Yakovlev
Newcastle University, UK
{Nikolaos.Minas,David.Kinniment,G.Russell,Alex.Yakovlev}@ncl.ac.uk

Abstract

Many papers have been published in the area of metastability and synchronization in digital systems, mainly describing techniques for minimising the effects of metastability in synchronisation and arbitration. With the recent use of FPGA devices and their extensive use in a wide range of applications, some of the synchronization techniques used do not apply. The area around these devices have been left mainly unexplored as to what happens when an FPGA is in metastability or what information about the device can be extracted by measuring metastability. This paper is a work in progress, which presents findings so far from our measurements in two FPGA from different vendors. It mainly demonstrates the phenomenon of metastability in FPGA devices and how metastability affects bistable elements, which are critical components of Asynchronous Circuits.

1. Introduction:

FPGAs have been used for many years. Although, the advantages of using such a device are well known, they were mainly used as prototyping platforms. In recent years, FPGA complexity has progressed to a point where System-on-Chip (SoC) designs can be built on a single device. The number of equivalent gates and features has increased dramatically to compete with capabilities once offered only through ASIC devices. With designs becoming more complex, a single clock will be very difficult to accurately distribute across the entire system. To avoid the problem of clock distribution, multi clock domain systems have been designed. Although this solves the problem of clock distribution, they create the problem of synchronisation of data between blocks of different clock frequencies.

In the case of data transfer between two blocks of different clock frequencies, the data crossing the new clock domain will be considered as asynchronous; this can cause a violation of the set-up and hold time requirements of the storage element, which can then lead into metastability. A simple way of decreasing the probability of failure is to use a synchronous circuit shown in Figure 1, consisting of two flip-flops in series; the idea behind this is that even if one of the two flip flops goes into metastability, the next one will catch the correct logic output in the next clock cycle.

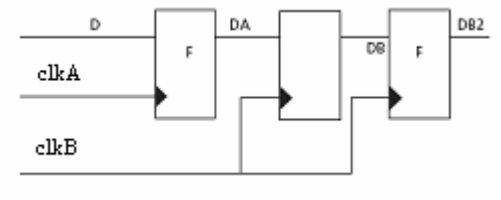


Figure 1. **Two Flip-Flop Synchroniser**

Although the synchroniser above has a good MTBF (Mean Time Between Failure) if the clock period is long, this is not always the case, and often more robust circuits are needed, in order to improve latency and thus the performance of the overall circuit. Designing synchronisers for FPGA based circuits can be difficult due to the limited choice of components that can be implemented in this technology. Semiat and Ginosar [1] successfully designed a series of synchronisers for FPGAs. However, our work will not concentrate on synchroniser designs, but on extracting metastability characteristics from different FPGA families and vendors, and designing on-chip timing circuits to measure accurately the changes in the propagation delays of the storage elements.
Section 2 describes the experimental set-up used for taking measurements in the two FPGAs used. In section 3 the measurements taken and the results obtained are analysed, also a bistable element is constructed using NAND gates in the Xilinx device to demonstrate the effects of metastability in circuits with feedback loops passing through LUTs (Look Up Tables). Finally in section 4 we present our conclusions.

2. Experimental set-up:

In order to extract the metastability characteristics of the test devices, the circuit configuration [1,2] used as shown in Figure 2. The FPGA devices used were the FLEX10K20 and the Virtex-II from Altera and Xilinx respectively.

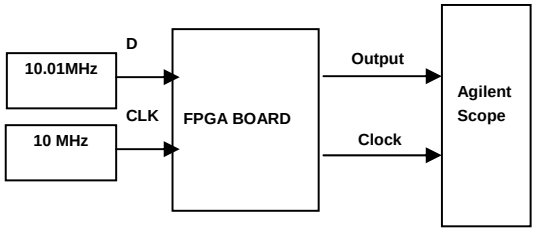


Figure 2. **Experimental set-up**

Two asynchronous oscillators are used to drive the D and clock inputs of a D-type edge triggered flip-flop under test. With a slight difference in the frequency of the two oscillators, 10.01MHz for data and 10MHz for the clock, the rising edge of the clock may, or may not produce a change in the output Q of the Flip-Flop. According to Kinniment et al [3], metastability can only be observed if the D input is different in successive clock edges.
Since the two oscillators are not locked together, all overlap times between 0 and 100ns cycle time for data and clock are generated with equal probability. In order to observe the delay due to metastability, the change of the Q output from low to high is used to trigger the recording of each clock rising edge for a potential metastable event. The only events that can be observed are those events where clock and data overlap by less than the difference of the two oscillators periods (<100ps), since they are the only events to produce a change in the Q output. These events are then presented as a histogram of the number of events collected against the time from the Q output to the clock. The events that result in deep metastability, and thus in long propagation delays are very rare. These kinds of events require less than a 100ps overlap, which occur every 1000 clock cycles. However, not all of the metastable events will be collected; this is because the oscilloscope used to capture the events, a 54850 series Agilent Infinium oscilloscope, has a significant dead time between successive measurements.

3. Measurements:

This section presents the measurements taken from two FPGAs, first the Altera FLEX10K FPGA is tested, followed by the Virtex 2. All the measurements were obtained using an Agilent technologies 54855A Infinium oscilloscope. It is useful to note the importance of these measurements, since FPGA vendors rely mostly on metastability detectors such as those described in [4, 5, 6], which can be inaccurate and in most cases produce results which are misleading. Measurements of metastability characteristics were also taken from a bistable element constructed from gates in the Xilinx Virtex 2 FPGA.

3.1 Metastability in FLEX10K:

Using the circuit configuration outlined in section 2 the metastability behaviour was collected over a period of 4 hours. Our results are shown in Figure 3, where the oscilloscope is in colour grade mode, so that the density of traces at a particular point is represented by the colour of the pixel at that point on the display. A histogram of the trace density along the horizontal line is also shown in this figure, which represents the number of events passing through the pixels concerned. The change of the Q output is used to trigger the recording of each clock rising edge for a potential metastable event.

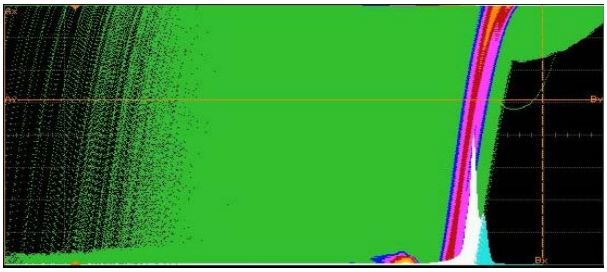


Figure 3. Metastable behaviour in Altera Devices

However, to observe more details of the characteristics of the device and to extract the values of τ (resolution time constant) for the different regions, the histogram of the waveform depicted earlier need to be plotted in a semi-log scale, as shown in Figure 4,

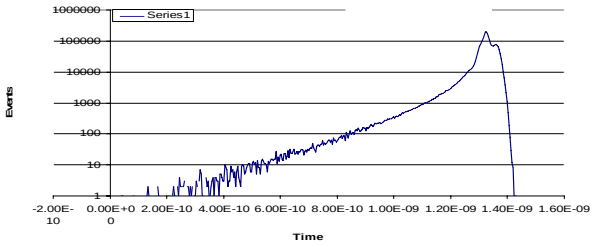


Figure 4. Histogram of Altera FPGA

In Figure 4 the X-axis represents time from a triggering Q output back to the clock edge and therefore increasing metastability time is shown from right to left. From the histogram the value of τ for the metastable region can be measured. The slope of the metastable region starts at about 350ps and ends at 950ps. In this instance τ is about 120ps and is obtained from the reciprocal of the slope of the histogram. This particular device is quite slow and metastable events can be observed better than in faster devices as we will see in the next section.

3.2 Metastability in Virtex 2:

The Virtex 2 is a faster device than the Altera. Thus it was expected to observe similar results to that shown in Figure 3, however with less frequent occurrence of metastable events, the experiment ran for a bit longer, in an attempt to capture more events.

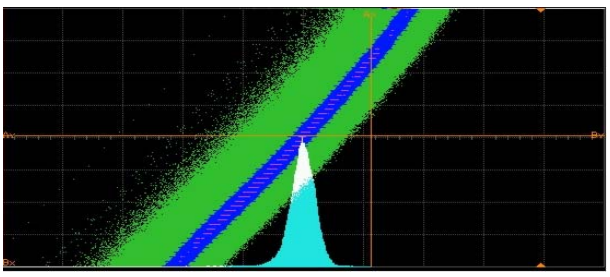


Figure 5. Metastable behaviour in Xilinx Devices

When the histogram was plotted the graph of Figure 6 was observed, which is completely different from the Altera device histogram. One explanation is that the two oscillators have become locked together. By reducing the period difference of the two oscillators, we subsequently reduce the number of normal transition events until they disappear completely at the point where the difference of the two periods is smaller than the setup/hold time window. Consequently what is observed is quite deep metastability where the two edges only differ by a small amount of jitter.

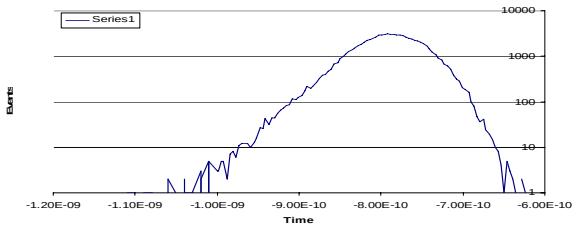


Figure 6. Histogram of Xilinx FPGA

In this instance we can observe from the histogram that the Xilinx device has a faster resolution than the one tested before. In this case, τ is in the range of 30 to 40ps. However in order to directly compare the two devices, the two oscillators need to be unlocked.
The experiment was subsequently rerun, however in this instance the difference in the period of the two oscillators was increased from 100ps to about 500ps. The sensitivity level of the trigger was high. The metastability behaviour captured in Figure 7 was observed. As it can be seen, the output this time looks more like the one expected earlier, where although metastable events are rare, the spread of the metastable behaviour is still obvious.

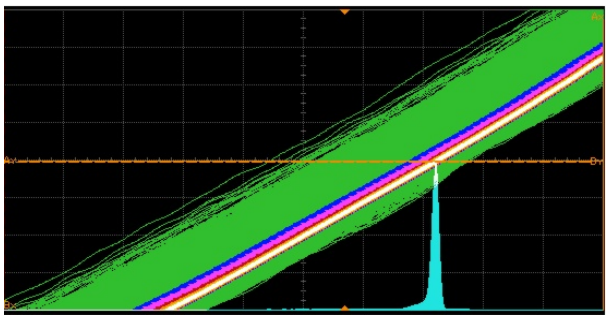


Figure 7. Effects of frequency change in metastable behaviour

From the histogram the value of τ is about 50ps, which verifies that the Xilinx device is faster than the Altera device tested, as shown in Figure 8. The anomaly in the histogram between 250ps and 270ps is similar to the results of the Miller effect, described by Dike and Burton [2], which is a coupling effect occurring across the gate-drain junction of one of the device transistors, which mainly occurs in jamb latches.

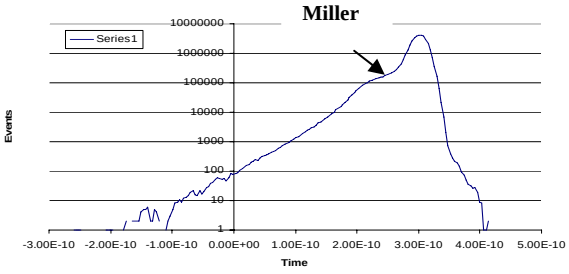


Figure 8. Histogram of Xilinx Devices with increased frequency

3.3 Metastability in a Bistable Element:

A master-slave D type edge triggered Flip-Flop was constructed using NAND gates in a Xilinx FPGA, as shown on Figure 9, to observe the phenomenon of metastability in a bistable element. The master and the slave were placed in two different sectors within the same Logic Block, but very close to each other.

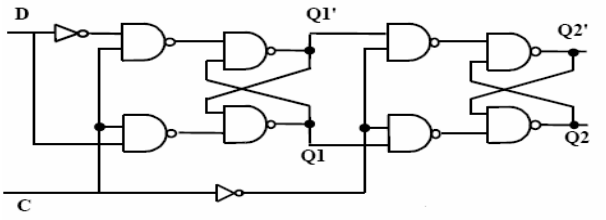


Figure 9. D type Flip-Flop with NAND gates

The Experimental setup was similar to that used in previous sections, with the data and clock oscillator having a period difference of 100ps. The routing delays play a significant role in this experiment, since the routing of the design is up to the software, hence the feedback loop can have large delays, thus allowing us to observe the effects of oscillation due to metastability. It is expected that the values of τ will be much larger than those in the Flip-Flop modules of a Xilinx FPGA. The metastability behaviour captured for the D type Flip-Flop can be shown in Figure 10.

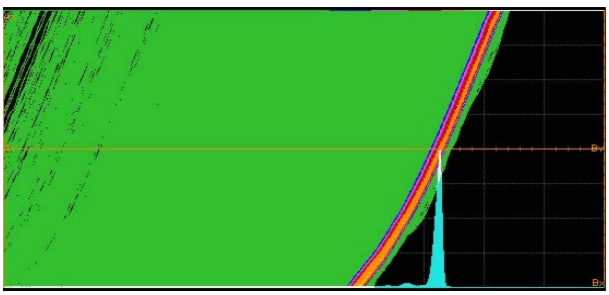


Figure 10. Metastable behaviour in a NAND gate based D type Flip-Flop

From the histogram shown in Figure 11, it seems that there is a damped oscillation in the deterministic region, suggesting that the feedback loop is unstable. Also the metastable region is presented much longer than previously thought. To calculate the value of τ, the histogram was plotted in the same manner as before. As it can be seen from the histogram, the value of τ is in the order of nanoseconds, which is much larger than the values calculated earlier for both the Altera and the Xilinx FPGAs. This demonstrates that circuits with feedback loops that pass between LUTs can exhibit oscillation. Bistable elements are generally used in asynchronous circuits to design mutexes and arbiters, because there are no standard mutex or arbiter cells in FPGAs, the method of cross-coupled gates is used to design such circuits.

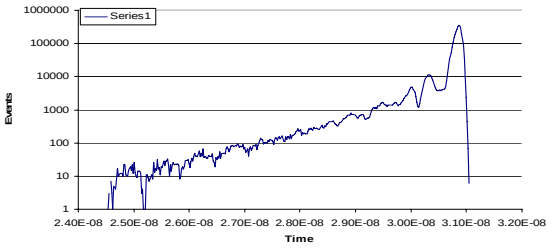


Figure 11. Histogram of a NAND gate based D type Flip- Flop

Normally a metastability filter would be used on the output to eliminate the metastable event. However, these filters cannot be easily designed in FPGAs, since Programmable logic devices lack the elements needed. An earlier attempt has been made to design a mutex by Semiat and Ginosar [1], but because of the likehood of oscillation taken into account this may not work, since it relies on an exclusive OR arrangement to filter the oscillation, but such circuits are prone to short high outputs when both inputs go from high to low, or the reverse. Another way of designing a mutex is to use the well behaved latches of the Xilinx devices, where as it was shown previously have a faster resolution times. This area is currently under research and it will be investigated thoroughly.

4. Conclusions and future work:

This paper demonstrated the effects of metastable behaviour in two FPGAs from two vendors. The results for the Altera FPGA, taking into account the process technology and the Logic Block design were as expected. However, the Xilinx FPGA was more resilient to metastability. The first experiment showed that the two oscillators were locked together, as a result true metastability was observed. However, for directly comparing the two FPGAs, the two oscillators needed to be unlocked, and for this reason the period difference of the two oscillators was increased.

By directly comparing the values of τ for both devices, it was verified that process technology plays a significant role in the resolution time of the Flip-flops. To demonstrate the effects of metastable behaviour in a bistable element a master-slave D type Flip-Flop was constructed from NAND gates in the Xilinx FPGA. The results observed from the histogram showed significant oscillation times before the signal becomes stable and also the value of τ was much larger. Bistable elements are mainly used in asynchronous circuits, to design arbiters and mutexes. By observing the metastable behaviour in the bistable elements, it has been demonstrated that when a bistable element is made out of gates it is more prone to metastability. Consequently a solution is been investigated to design mutexes and arbiters using bistable elements found in the FPGA rather than gates.
Currently architectures for the on-chip measurement and observation of metastability behaviour are been investigated, due to limitations of off-chip measurements using a digital oscilloscope.

5. References:

[1] Y. Semiat, R. Ginosar, "Timing Measurements of Synchronisation Circuits," Proceedings of the Ninth International Symposium on Advanced Research in Asynchronous Circuits and Systems, 2003, pp. 68-77.
[2] C.Dike, E. Burton, "Miller and Noise Effects in a Synchronizing Flip-Flop," IEEE Journal of Solid State Circuits, Vol. 34, No. 6. pp. 849-855, June 1999.
[3] D.J.Kinniment, K.Heron, G.Russell, "Measuring Deep Metastability," IEEE International Symposium on Asynchronous Circuits and Systems, 2006.
[4] P. Alfke, B. Philofsky, "Metastable Recovery," Application Note, Xilinx Corporation, August 10, 1996.
[5] Altera Corporation, "Metastability in Altera Devices," Application Note, May 1999.
[6] Lattice Corporation, "Metastability in Lattice Devices," Technical Note, March 2004.

Asynchronous Timing in the Survivor Memory Unit of a Viterbi Decoder

Wei Shao and Linda Brackenbury
APT Group, School of Computer Science, University of Manchester, Oxford Road, M13 9PL
Email. {shaow,lbrackenbury@manchester.ac.uk}

***Abstract*— The Viterbi algorithm is a frequently used convolutional error correcting code particularly used in digital transmission systems. In the decoder, the Survivor Memory Unit (SMU) determines the most likely output to have been sent by the encoder at each timeslot. To obtain this decode data, the SMU keeps a history over many timeslots of the most likely paths through the decode trellis denoting the possible state transitions. While the SMU enters data synchronously, the tracing back through the history to find the best output at the earliest timeslot is best performed asynchronously. Furthermore, while most asynchronous timing requires handshakes or timing included in the data, the asynchronous approach described here for the parallel backtrace operation has no handshake overhead and hence offers better power and performance characteristics. This is confirmed for post-layout simulation on a $0.18\mu m$ process which uses only 70% of the power of a previous handshaking asynchronousdesign.**

***Index Terms*— low power Viterbi decoder, trace back, survivor memory unit.**

I. Introduction

Digital communication systems are now pervasive in and an integral part of everyday living for an ever widening set of functions. As a result, the market for such digital equipment has become huge in recent years and is continuing to grow rapidly. Central to the successful use of such systems is the requirement to receive and correctly decode transmitted information. For portable devices there is a further requirement that the decoding be energy efficient and optimised; this is a result of the relatively small battery capacity.

Error correct codes improve the reliability of communication channels by detecting and correcting errors. Convolutional codes are often used here because the encoder output depends on both the input data and previous input bits, unlike the block codes where there is no dependence on the previous input history. This interlocking of the data over k bits provides convolutional codes with better error correcting properties than possible with block codes [1], [2]. For this reason, convolutional codes are widely used and the Viterbi algorithm is particularly adopted because it efficiently implements the maximum-likelihood decoding of a continuous data stream [3]. The constraint length, k, is a fundamental property of convolutional codes and relates both to the obtainable bit error rate and the complexity of the decoder logic. Unfortunately as the constraint length increases, the complexity of the decoding circuitry increases exponentially since the number of possible encoder states is $2k-1$. This limits the constraint length in most domestic applications to seven or less.

Internally, the Viterbi decoder comprises three blocks as shown in Figure 1. The Branch Metric Unit (BMU) computes the Hamming distance between the received input symbols and the data to be expected by a particular encoder state. The Path Metric Unit (PMU), adds the branch metric to the existing state metric, compares pairs of metrics, and selects the smaller metric which becomes the next metric for a state; this is depicted in the trellis diagram in Figure 2 which assumes that the encoder generates two output symbols for each input bit. The Survivor Memory Unit (SMU) keeps a history of the winner for each metric-pair comparison in the local winner memory over many timeslots and determines the decoded output from the earliest timeslot. Depending on the design, the SMU may also receive information about state(s) having a minimum weight from the PMU, e.g. the state "00" at the end of the trellis shown in Figure 2, indicating optimum point(s) from which to commence a search through the history information; starting from a known point reduces the number of timeslots needed to be stored by the SMU and hence reduces the power required.

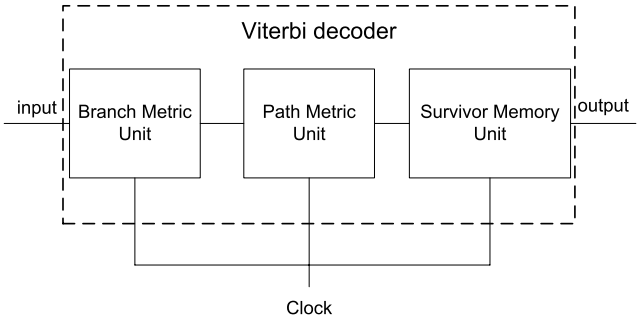


Fig. 1. Viterbi decoder block architecture

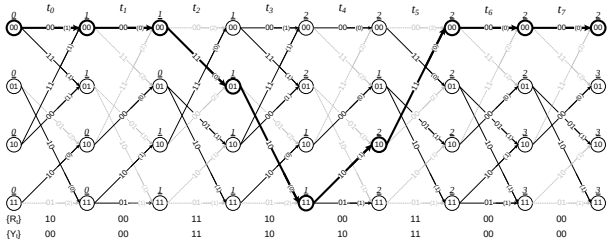


Fig. 2. The trellis structure shows the Viterbi decoding process of a R=1/2, k=3 code.

II. Timing in the SMU

Tracing back is fundamentally a recursive updating process where the traceback recursion estimates the previous encoder state S_{n-1} according to the current state S_n where

$$S_{n-1} = S_n[m-2:0]d_n^S \tag{1}$$

for the common radix-2 trellis. d_n^S is the one-bit decision winner from the comparison in the PMU and is read from the local winner memory located by state index S_n and time index n; the previous state S_{n-1} is obtained by simply removing the most significant bit of S_n and appending d_n^S as the least significant bit. Due to recursion, a trace back process can not be pipelined and this restricts the throughput. Furthermore, since a minimum length of five times the constraint length is required for a trace back to achieve high decoding accuracy, significant memory is required for the local winner memory; this incurs a considerable area and power overhead.

The input to and the output from the decoder is generally synchronised to an external clock. Thus data is input and removed at regular intervals and hence the operation of the BMU, PMU, the placing of data into and the removal of data from the SMU are all essentially synchronised to the clock. However, the tracing back through the timeslot history can be either synchronous [4], [5], [6] or asynchronous [7].

III. Handshaking asynchronous trace back

With asynchronous handshaking timing a trace back, implementing multiple back trace pointers is relatively simple and has far less control and memory overhead compared to a synchronous implementation. For these reasons, tracing back asynchronously is preferred. Figure 3 shows a schematic of the circular handshaking trace back mechanism used in the self-timed design in [7]. Each

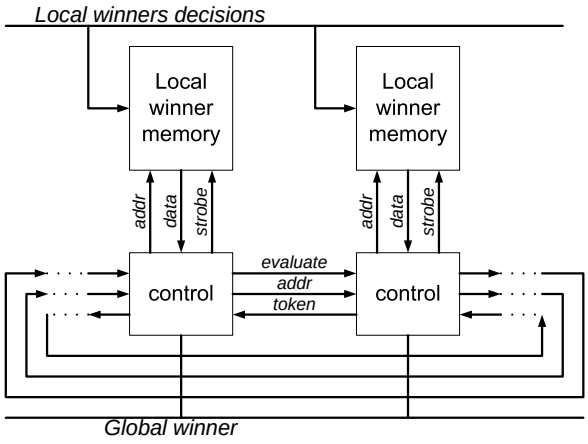


Fig. 3. The SMU architecture of the asynchronous design from [7].

timeslot comprises a local winner register plus control logic. A rotating token indicates the logic to receive the next PMU output and having loaded the winner memory, the token moves to the adjacent slot logic. At this point, the loaded timeslot issues an evaluate signal to initiate the trace back. A significant feature of the system is the use of a global winner indicating that the PMU has located a single minimum path metric. This is used as the starting point for a trace back. The path can now be reconstructed at each successive stage with the output address from a slot generated from its local winner information and incoming address, according to equation 1. Handshakes between adjacent slots control the transfer of data during back trace. Back trace(s) can therefore run asynchronously, as fast as the handshakes will allow. Furthermore, a variable and unknown number of back traces, determined by the number of paths requiring correction, can be running concurrently. This ability of the asynchronous design to automatically adapt to the error conditions gives it a flexibility that is absent in a synchronous design. Transition activity is minimised by storing the global winners generated by the PMU and retiring the back trace as soon as the stored global winner agrees with the generated address. Starting the back trace from a known point also enables the number of timeslots to be significantly reduced to close to the theoretical minimum of five times the constraint length. This small amount of memory is impossible to realise in synchronous designs.

IV. New no-handshaking asynchronous trace back

Although the asynchronous design described in the last section has considerable advantages over a synchronous approach, it too has drawbacks. The modest control logic for the handshake logic consumes power and also affects performance. Moreover, a trace back has to be forcibly stopped if it is in danger of running into the slot where data is about to be placed; this is resolved with an arbiter which introduces uncertainty and potential metastability to the design [7].

Handshakes are required to transfer data safely between sequential elements. If the sequential logic is removed and the trace back mechanism only comprises combinatorial logic then an asynchronous trace back can be performed without handshakes. In this case, the system needs to guarantee that the decoded bit to be output can be resolved correctly before the slot is loaded with new PMU data. The combinatorial trace back block is shown in Figure 4. The local winner data is held on flip flops and all its output bits are

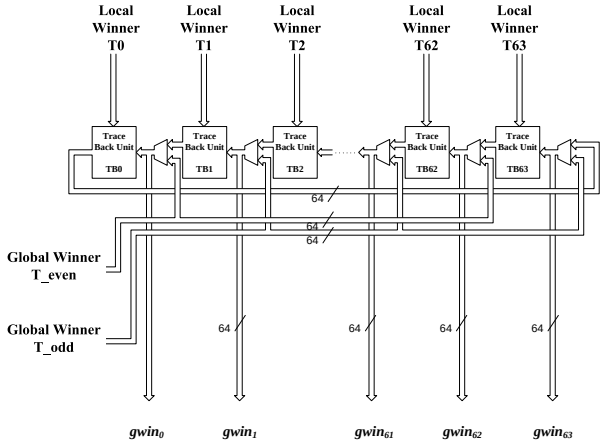


Fig. 4. Trace back path of the new *SMU* design.

fed to its associated trace back block and as shown it is a 64-slot system. Each trace back unit, TB_i is a direct implementation of one stage of the radix-2 trellis so that the global winner at time slot T_i is constructed from the local winner decisions and the global winner at time slot T_{i+1}; note that the global winner comprises a single bit per state. Each trace back stage consists of a trace back unit and a multiplexer; the multiplexer selects the global winner from either the PMU or the preceding trace back unit.

A trace back on all 64 paths is started at each time slot by selecting the new global winner from the PMU. Although trace backs are initiated synchronously, they run asynchronously thereafter. A major feature of this trace back path structure is that it needs no pointers or handshakes to control the progress of

trace backs. They run naturally on the trace back path through the combinatorial logic until they merge with a previous trace back or reach a time slot being updated by new global winners.

A. Timing considerations in new trace back

There are four potential timing problems arising from operating without sequential logic:

1) Switching of the multiplexers in the trace back stages: The multiplexer selection signal and the global winner from the PMU are synchronised to the updating of the local winner memory. However, the global winner from the predecessor trace back unit is asynchronous to this. If the multiplexer selection were to be switched just as a new output were generated then spurious transitions would be propagated down the trace back chain. To avoid this, global winners from the PMU update even trace back units on even timeslots, and odd trace back units on odd timeslots. This overlapping enables trace backs to be started when the adjacent global winner has become established and so avoids unnecessary switching transition propagation and avoids additional uncertainty in determining the bit to output. The timing of the multiplexer switching is shown in Figure 5. As can be seen, the selection of Sel_i

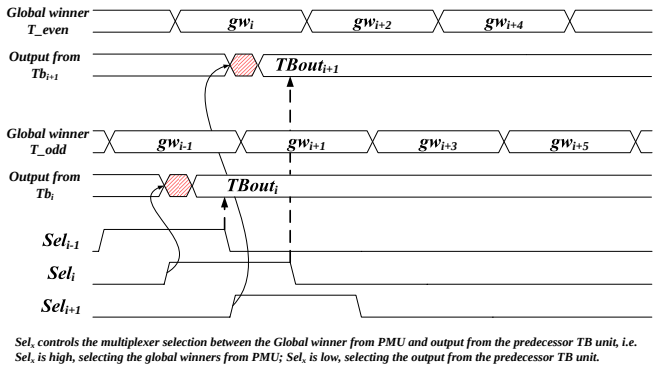


Fig. 5. The timing of the trace back path multiplexer.

changes only when the input $TBout_{i+1}$ from the the predecessor trace back unit to the multiplexer is steady.

2) Output synchronization: The global winner signals propagate asynchronously with respect to the synchronous clocking of the decoded bit from the oldest timeslot into the output flip flop. Therefore, changes of data at or near the clock may cause incorrect data to be clocked or the flip flop to enter the metastable 'half' state. In both cases, changes at this time indicate that the traceback has not converged on to the correct path. The data output under these circumstances from any Viterbi decoder is usually random depending on the internal logic. It therefore does not matter whether a '0' or '1' is output as there is a 50% chance that it will be correct; this is confirmed by simulations which show that the error rate using the new SMU is no different from those of conventional Viterbi decoders. However, a metastable state still needs to be avoided in the output flip flop. This is achieved by using two flip flops in series having the same clock but with the output of the first connected to the input of the second. This allows one clock period for any metastable output from the first to settle. The mean time between failure (MTBF) is given by [8], [9], [10], [11]

$$MTBF = \frac{e^{t/\tau}}{T_w \times f_d \times f_c} \tag{2}$$

where f_d and f_c are the data and clock frequencies, T_w is the time between the clock and data giving non-zero resolving time, t is the clock period and τ is the time constant for leaving the metastable state. For the $0.18\mu m$ 1.8V standard cell CMOS library used, measurements of τ and T_w yielded maximum values of 43ps and 21ps respectively giving a MTBF of over 1000 years for a 100MHz clock with far larger MTBF for smaller clock frequencies. With this MTBF, the output data can be considered to be sufficiently reliable.

3) Positive timing skew: All paths trace back simultaneously and because of element tolerance and differences in wire length, there will be a variation in time between the arrival of traceback decisions at the oldest timeslot. If large enough, this could cause incorrect data to be determined for clocking into the output flip flop. There are two cases to be considered. In this section, the case of a succeeding global winner sent at time t, indicated by a propagating '1', moves through the trace back logic faster than the previous global winner on another path sent at time $t-1$; this is referred to as positive timing skew.

Both global winners will eventually merge on the same traceback path so if the winner sent at t catches up with the one sent at $t-1$ then the trace back logic is still selecting the correct trace back path and the output will be accurately decoded. Hence, it can be concluded that positive timing skew causes no problem and can therefore be ignored.

4) Negative timing skew: However, negative timing skew where the global winner at time t travels slower than the winner at time $t-1$, may cause problems. This is illustrated in Figure 6. The solid

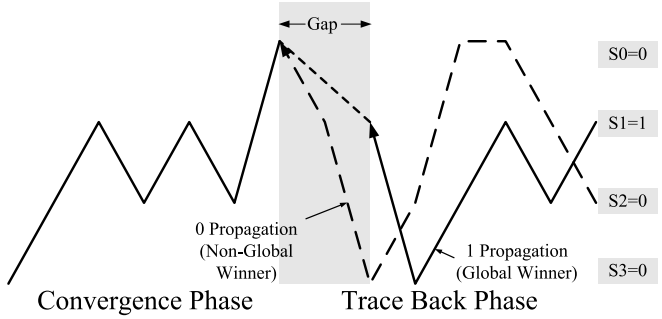


Fig. 6. Trace back gap caused by timing skew.

line from state S_1 represents the global winner propagation with time and the dotted line represents all the other 'loser' states. The slower propagation of the winner means that the loser '0' states can combine with a zero on the winner path from the previous timeslot to indicate a period where no winner is indicated; this is indicated by the shaded region in Figure 6. Were this to happen at the time the decoded data is clocked into the output flip flop then incorrect data would be decoded and output. After loading a timeslot, the load pointer moves on forward and the trace back moves in the reverse direction. The data will be read out just before the trace back reaches the load pointer. If there is a total of L stages, when the load and trace back meet, the back trace has travelled n stages and the load pointer has passed $(L-2-n)$ stages; $L-2$ arises from the timing of the switching of the multiplexers in the trace back stages. If the trace back delay per stage is d and the clock

time is T

$$n \times d = (L - 2 - n) \times T \quad (3)$$

It is the range of delay in a trace back unit that gives rise to the 'zero' gap that may arise. Post layout measurements for the $0.18\mu m$ 1.8V process targeted reveal a variation between d_{min} and d_{max} of 0.55ns to 0.615ns. Taking a SMU path length L as 64 and using equation 3, the variation in n at different frequencies can be computed and are as shown in Table I. It can be seen that despite the delay variations to be expected in the trace back units, the output data is always within the same stage and so will be correctly output over the range of frequencies shown. This has been confirmed from

TABLE I

MINIMUM AND MAXIMUM TRACE BACK STAGES AT A $0.18\mu m$ GEOMETRY.

Frequency (MHz)	**Min Stages (n_{min})**	**Max Stages (n_{max})**
10	61.62	61.66
50	60.15	60.34
100	58.41	58.77
200	55.21	55.86

45MHz to 100Mhz by post layout simulation of a Viterbi decoder using the new SMU. The bit error rate obtained over a wide range of additive Gaussian white noise levels indicates no discernible difference between the conventional decoder and the decoder using the new SMU. Furthermore, comparing the SMU power dissipation at 50 MHz with the previous self-timed design scaled to $0.18\mu m$ and 1.8V shows that the new design consumes only 7mW against 10mW of the other. While the throughput of the previous design was limited by the speed of the handshakes and arbitration, making this a bottleneck to the decoder throughput, the current SMU has no such limitations making operation at far higher frequencies viable. The only possible drawback of this design appears to be the large number of wires required which will probably limit the constraint length to a maximum of seven.

B. Scaling the new trace back design

As geometries scale down, the stage delay d will alter and the negative timing skew may cause the trace back unit to fail. According to the first-order 'constant field' MOS scaling theory [12], scaling a process down by a factor α reduces the gate delay by the same factor α while the wire delay remains the same. Based on this, the variation for d_{min} and d_{max} is shown in Table II. Using

TABLE II

MINIMUM AND MAXIMUM DELAYS OF EACH TRACE BACK STAGE FOR DIFFERENT GEOMETRIES.

Geometry (nm)	**Min Delay (ns)**	**Max Delay (ns)**
180	0.550	0.615
130	0.400	0.465
90	0.275	0.340
60	0.183	0.248

these figures, equation 3 can be used to find the variation in n at these geometries and this is shown for a 50MHz clock and 64 stages $(= L)$ in Table III. Although negative timing skew results in a potential gap equivalent to up to 0.2 of the time through a trace back stage at this frequency, the results for 90nm indicate that the output could fall in different stages and could therefore cause an output error. This can be avoided by either shifting the output clock edge or by altering the number of trace back stages. It does indicate that a design without handshakes is not without problems and that careful timing analysis and simulation is required at the working frequencies and on the targeted process.

TABLE III

MINIMUM AND MAXIMUM TRACE BACK STAGES AT 50MHZ AND L=64.

Geometry (nm)	**Min Stages (n_{min})**	**Max Stages (n_{max})**
130	60.59	60.78
90	***60.96***	***61.16***
60	61.24	61.44

V. CONCLUSION

A new asynchronous SMU which does not require handshakes or timing within the data to operate correctly has been described. The timing problems arising from its use have been discussed and it can be concluded that the only significant difficulty arises from negative timing skew. The amount of skew observed is a function of the maximum variation in time through a trace back stage, the number of stages and the clock period. These enable the timing skew to be computed and any potential problem in decoding data to be identified. Examination of reducing the geometry shows that the design is scalable although the output clock edge or number of stages may require adjustment. Simulations of the new SMU reveal that it has the same bit error rate as the conventional decoder so it can be concluded that outputs are not being erroneously decoded. Since the use of only combinatorial logic has led to very low power levels and high potential throughput, this may be a useful approach in other suitable asynchronous applications.

REFERENCES

[1] G. C. Clark and J. B. Cain, *Error-Corection Coding for Digital Communications*. New York: Plenum Press, 1981.

[2] L. C. Lee, *Convolutional Coding: Fundamentals and Applications*. Norwood: Artech House,Inc., 1997.

[3] A. J. Viterbi, "Error bounds for convolutional codes and an asymptotically optimum decoding algorithm," *IEEE Trans. Inform. Theory*, vol. 13, pp. 260–269, Apr. 1967.

[4] C. M. Radar, "Memory management in a Viterbi decoder," *IEEE Trans. Commun.*, vol. 29, pp. 1399–1401, Sept. 1981.

[5] P. J. Black and T. H. Y. Meng, "Hybrid survivor path architectures for Viterbi decoders," in *Proc. ICASSP*, Minneapolis, MN, USA, Apr. 1993, pp. 433–436.

[6] R. Cypher and C. B. Shung, "Generalized trace back techniques for survivor memory management in the Viterbi algorithm," in *Proc. ICASSP*, Minneapolis, MN, USA, Apr. 1993, pp. 1318–1322.

[7] P. Riocreux, L. Brackenbury, M. Cumpstey, and S. Furber, "A low-power self-timed Viterbi decoder," in *Proc. Async*, Salt Lake City, Utah, USA, Mar. 2001, pp. 15–24.

[8] J. Sparsø and S. Furber, *Principles of Asynchronous Circuit Design: A Systems Perspective*. Dordrecht, The Netherlands: Kluwer Academic Publishers, 2001.

[9] C. L. Portmann and T. H. Y. Meng, "Metastability in CMOS library elements in reduced supply and technology scaled applications," *IEEE J. Solid-State Circuits*, vol. 30, pp. 942–951, Jan. 1995.

[10] D. J. Kinniment, A. Bystrov, and A. V. Yakovlev, "Synchronization circuit performance," *IEEE J. Solid-State Circuits*, vol. 37, pp. 202–209, Feb. 2002.

[11] C. Dike and E. Burton, "Miller and noise effects in a synchronizing flip-flop," *IEEE J. Solid-State Circuits*, vol. 34, pp. 849–855, June 1999.

[12] N. Weste and K. Eshraghian, *Principles of CMOS VLSI design: a Systems perspective*. Reading: Addison-Wesley, 1993.