Automated Timing Constraint Generation for Pulse Gate Circuits

David Mc Carthy
ECE Department, UCSB
Santa Barbara, CA
davidmc@ece.ucsb.edu

Merritt Miller
ECE Department, UCSB
Santa Barbara, CA

Forrest Brewer
ECE Department, UCSB
Santa Barbara, CA

ABSTRACT

In this paper, we address the problem of verification of pulse-gate circuits. These circuits enable the design of very high performance logic functions such as data-recovery, pipeline and FIFO control logic. We adopt an approach of using an abstraction of the structure of the circuit as the specification. From this we can first obtain the nominal case behaviour of the circuit using conventional NFA exploration techniques adapted to distributed activity systems. Following the identification of possible nominal states, we identify the critical path inequalities that must be maintained to ensure this behaviour in implementation. This strategy mimics the abstract designer behavioral view of pulse gate activity and leads to a practical set of timing constraints for composite self-resetting and astable asynchronous logic circuits.

ACM Reference format:
David Mc Carthy, Merritt Miller, and Forrest Brewer. 2019. Automated Timing Constraint Generation for Pulse Gate Circuits. In *Proceedings of 28th International Workshop on Logic and Synthesis, Lausanne, Switzerland, June 21–23, 2019 (IWLS '19),* 8 pages.
https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Pulse gates[4] have signal propagation times that are 30-50% slower than conventional CMOS gates. Nonetheless, pulse gates represent a methodology to create circuits that operate at much higher rates than conventional CMOS logic in practice. This is due to execution with localized timing signalling that is electrically co-incident with the data signal. Due to relatively high power density, high performance pulse circuits are not appropriate for generic logic functions, but are admirably suited for smaller, critical logic such as SERDES, link subcircuits, FIFO/cache control and arbitration logic and selective clock technologies such as elastic pipelines. We choose to create a construction paradigm that allows exploitation of the high performance, but limits the timing complexity to burst-mode asynchronous at the gate level, and consensus at larger scales. Thus, pulses can be gated or indeed subsumed by earlier pulses, but, pulse arrival arbitration is handled by a separate circuit out of the timing model. These constraints act to limit the complexity of timing verification, avoiding factorial complexity growth in more general asynchronous circuits.

IWLS '19, June 21–23, 2019, Lausanne, Switzerland
© 2019 Association for Computing Machinery.
ACM ISBN 978-x-xxxx-xxxx-x/YY/MM...$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

1.1 Related work

The notion of self-resetting gates can be traced to work by Sites and others at the dawn of NMOS technology. The first systematic work was that of Martin and Nyström[6]. In this work, pulse gates were used as part of Quasi-Delay Insensitive design paradigm, so were used in circuits designed not to exhibit external timing dependencies by structure and thus the problem of static timing analysis or verification did not arise. Greenstreet[1] created Pulse gate based micropipelines. These circuits exhibited the relatively high performance potential of pulse-gate designs, but beyond the pipeline stage setup and hold issues, timing models were not developed.

SRCMOS is another pulse gate style aimed at data-path computation. Computation is performed with overlapping wide pulse, implying many timing constraints to ensure pulse overlap[5]. However, such circuits treated these gates as dynamic gates, locked in a governing synchronous paradigm. This use enabled high stage performance of clocked designs, notably Intel P4 arithmetic pipelines[3] (Intel used the term self-resetting domino for this work). Again, the governing synchronous clock (at 1/2 the state rate for Intel) cast the timing problem back into the synchronous model.

Relative timing[7] is a method of identifying timing inequalities in extended burst-mode circuits, to allow the optimisation of slower hazard free structures into faster ones with hazards that can be checked. The overall approach is somewhat similar to this paper, in that intended behaviour can be read from circuit structure. The local composition

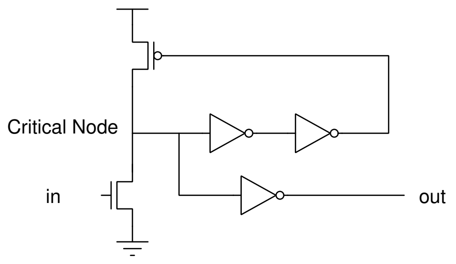

Figure 1: Pulse gate implementation

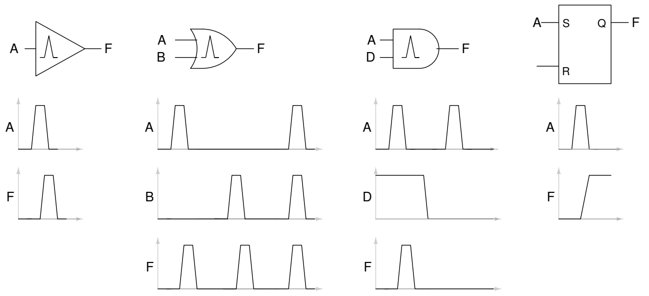

Figure 2: Basic Pulse gates

rules used in that work mean that within the scope of analysis, each signal is only used once and so inequalities can be written in terms of the occurrence times of signals. In pulse gate circuits, designers frequently include looping behaviour even on a local level so it is necessary to analyse circuits where gates are used many times per analysis scope. Thus inequalities must be on paths rather than individual signals.

Yet another pulse gate timing algorithm appeared in [4]. This algorithm requires identification of frames where each gate only used once, this technique did allow for looping behaviour (i.e. self-circuit re-triggering), but the manual enforcement of timing frames open the potential for missing actual behavior to verify and the model was incapable of modeling pulse absorption, a behavior used in stabilizing high performance clock phase generators.

2 PULSE GATE BEHAVIOR

Pulse gate circuits are composed of two types of signals. "Pulse" signals carry timing information, and their presence or absence can also convey state information. Pulses are treated as atomic, that is, the pulse as a whole is considered as conveying timing information not the rising and falling edges separately. "Data" signals are levels coming out of a latch, and convey state information only, and it is assumed that circuits receiving data signals use only their instantaneous value (at the arrival time of some pulse) and not the edge timing.

A pulse gate circuit consists of two types of gates, the pulse gates themselves and pulse latches. A pulse gate is a self resetting CMOS gate, as shown in 1. An arriving pulse pulls down the critical node. The output then starts rising. After the propagation delay of the reset loop, the critical node is pulled back up by the reset transistor, and the output goes low to end the pulse. The gate is sized so as the pulse is "round topped", ensuring that rising and falling times are independent of input excitation. Thus the pulse shape is largely dependent only on actuation arrival time, and not slope or energy.

This simplest pulse gate is a pulse buffer and repeater, taking an input pulse and generating a new pulse after the propagation delay. Buffers are used to restore signal integrity as both amplitude and pulse width are restored. Pulse gates exhibit mild intra-pulse timing sensitivity in that pulses effectively repel each other. (The pulse propagation delay of gate tends to increase if the gate was recently triggered). This leads to long-term stability in that pulses will not approach each other in arbitrarily long gate chains[2]. Pulse AND gates function as guarded pulse buffers and only produce a pulse if the logic level input is high when the pulse arrives. Pulse OR gates take two pulse inputs and produce a pulse output if either one of the input pulses arrive.

Coalesence is a behaviour of pulse OR gates where two inputs can arrive near simultaneously, and result in the gate firing once correctly in response the first pulse and the second slightly slower pulse being ignored.

More complex, gates are possible by combining pulse and data signals in relatively arbitrary pull-down networks. In general, the allowed pull down networks are those that are sum-of-product terms where each product term contains at least one pulse signal and possibly additional data signals.

Finally, data signals are produced by set-reset pulse-latches, where both the set and reset actions are sponsored by pulse arrival. After the propagation time, (similar to pulse gate propagation times), the output is a latched level.

3 SYSTEM CONSTRUCTION

A simple pulse binary counter is shown in Figure 3. It takes a input clock pulse train, and using Pulse And and Or gates to filter the incoming pulse into pulses which encoding both timing and state information and then these are used to set or reset state latches creating the effective next state data signals. There are obviously many possible ways to organize such a counter, and a trade-off between timing and state propagation complexity. At one extreme, the circuit consists of a pulse based clock network and state latches, on the other extreme, multiple paths are selectively executed on arrival of selected pulse paths or even race/consensus mode. The ability to readily produce data-laden signalling events which are locally timed accounts for the potential high performance.

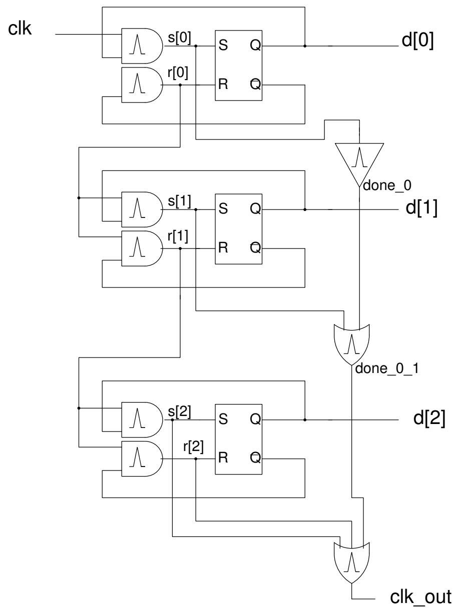

Figure 3: Pulse Gate Binary Counter

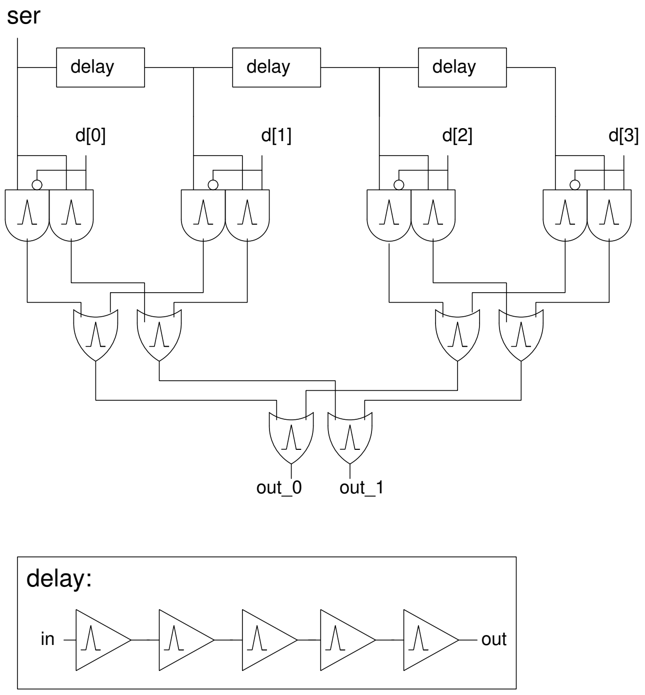

Figure 4: Pulse Gate 4-bit Serialiser

The main restrictions on pulse gate circuits are the typing rules of pulse gate circuits are observed, that timing information is derived only from pulses and not changes in data signals, that two pulses arriving at a pulse gate are either near enough to coalesce or far enough apart to facilitate two complete output firings. Two pulses arriving at an Pulse SR latch do not arrive so close as to cause meta stability, and data signals are held steady when being sampled by a pulse.

Two strategies are commonly used for local coherence in pulse gate circuits, one is depositing data in latches for another pulse to read (the pulse path going into the latch thus terminating and timing then being defined by the other pulse path). The other is using coalescene to pick the first of two pulses arriving at an OR gate. While this can only accomodate a small amount of dispersion (before gate electrical rules are violated) it can prevent these small amounts of dispersion from accumulating over time.

Many of the standard asynchronous design styles can be constructed with pulse gates within these rules. However, pulse gates circuits are at present typically designed by human designers and optimised for high performance, Thus these designs typically violate the assumptions made by most formal design styles. For example, many formal design styles of asynchronous logic do not admit combinational looping behaviours. That is, within the scope of one "evaluation" of a local circuit, no signal changes more than once. An example of a pulse gate circuit that admits looping behaviour is the serialiser in figure 4, in this case, finite looping behaviour.

Given the wide variety of behaviour, designers don't want to commit to a formal specification model that would exclude behaviours leading to better performing designs. In particular, if a circuit has the desired simulated behavior and that behavior is stable across a reasonable range of design parameter variations, one would like to verify the layout timing behavior and validate the design's timing robustness in an automated way. This is approximated by symbolic execution of an abstraction of the circuit to determine all potential "states" and then attempts to build timing constraints that meet the above operating hazard conditions. Such a paradigm operates well for sufficiently small designs but must be augmented by consensus or handshake coherence mechanisms for large designs. The requirements to meet global timing consensus are beyond the scope of this paper as quite substantial circuits fit the pseudo-static assumption, detailed below.

4 PSEUDO-STATIC ABSTRACTION

A key property of these gate circuits is that all pulse gates tend to have a similar nominal-case propagation time, due to the time being dominated by the common reset loop rather than the input PDN. Thus designers working with the circuits assume that if they have a set of nearly simultaneous pulses in the system at one time in a local region of the circuit, then one nominal gate delay later they will have the resultant pulses one gate delay later, and that those resultant pulses will also be almost simultaneous. Whether this unit-time approximation is good or not depends on how large a region of behaviour is being analysed. On a local scale, pulses will only travel separately for a short number of gates before interacting again. Thus only a small amount of attenuation and dispersion can occur. On system scales, different blocks of the system will have pulse behaviour that may be derived from separate local timing loops and thus having no relation to each other. System level timing solutions such as C-gates or arbiters are required, and it would also make little sense to apply the unit time approximation to this.

One way of quantifying whether the approximation is appropriate is to look at how long the pulses in the region of a system travel separately before all pulses remaining in that region can attribute their timing back to one common ancestor. This is called the "coherence depth". If the coherence depth is small then only a small amount of variance can occur.

This paper considers regions of circuits over which the unit time approximation is reasonable, and attempts to generate timing constraints over the region. While this still remains a local strategy that needs to be used within a global system coherence model, the variety of behaviour that is considered local behaviour is larger than other strategies, in particular finite looping behaviour and coalescence are available within a local region while many local timing analyses assume each gate it used once within a local timing region.

4.1 Unit-time Model

The unit time model of a pulse gate circuit is a formalisation of the pseudo-static abstraction. It is a model defined by the abstract gate-level model based on the circuit connections. The behaviour of a circuit is defined as a sequence of discrete 'states'. This nominal behavioural model can then be used as a specification to derive timing constraints, rather than having to provide a formal specification in addition to the structure.

A state in this model is defined as the set of pulses present simultaneously in the circuit, as well as the value of the data signals at that time. The successor of a state is defined as the pulses present in the circuit simultaneously one nominal gate delay later, with latches updated if they have accepted pulses.

That is, the model assumes that all the gates in the circuit have the same nominal delay and thus the progression of pulses can be modelled as bunches of temporally coincident pulses, which each bunch sponsoring the next bunch such that the next bunch is assumed to also be temporally coincident,

The state update function of this state model is defined by evaluating all the gates exactly once on the current state (i.e. set of pulses and data values), producing new data values and a set of successor pulses. For a pulse gate the update function is defined as:

$$pulse[n+1] = f\_pdn[n]$$

where f_pdn is the sum-of-products function defined by the pull down network of the gate input. Data gates can be defined similarly. While not contributing directly to the state, it is also useful to track whether the data signal has changed values or not, since it is changes in data value that leads to data hazards, not the immediate value.

$$data[n+1] = f\_pdn\_set[n] \vee (\neg f\_pdn\_reset[n] \wedge data[n])$$

$$data.changes[n+1] = data[n] \oplus data[n+1]$$

Thus we effectively have a Non-deterministic Finite Automata model of a the nominal behaviour of the circuit. This can explored using any NFA reachability analysis to enumerate all the possible behaviours of the system.

5 HAZARDS

There are two possible ways in which a pulse-gate circuit can function other than intended. Either an electrical error in the pulse gates can occur, or the pulse gates can function electrically correctly but logically incorrectly. Both these types of failures are produced when two signals are combined at a gate, and whether the failure is electrical or logical depends on the degree to which the assumption is violated. For signals that are meant to be separated in time, them moving closer together results in electrical errors. If the error is sufficiently great that the signals move further apart again, now in the wrong order, the circuit will function electrically correctly again but will be logically incorrect. Similarly for signals that are meant to be coincident, slight separations will result in electrical errors but larger separations will give electrically correct but logically incorrect behaviour.

Specifically, 5 categories of hazard exist in pulse gate circuit:

- A **Hold** hazard exists when a pulse is ANDed with a data signal, and the pulse arrives and is expected to be combined with the old value of the data signal before the data signal changes. If the pulse arrives later than nominal, or the data changes earlier, a hold violation occurs.
- A **Setup** hazard exists when a pulse is meant to be ANed with the new value of a data signal, after that data signal changes.
- A **Retrigger** hazard exists when a pulse gate fires for the second time too soon after the first, such that the pulse loop is still resetting.
- A **Set-Reset Order** hazard exists when a SR latch receives a set activation after a reset activation (or

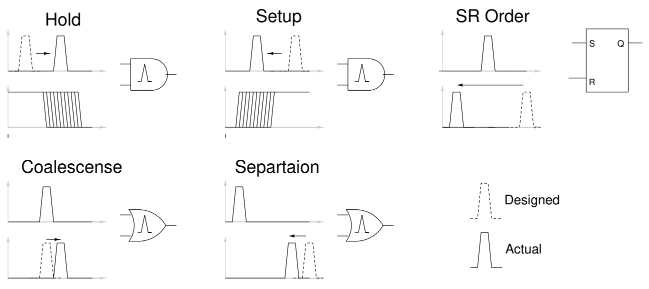

Figure 5: Possible hazards in a pulse circuit

- A **Coalesce** hazard exists when two pulses are meant to arrive at the same time at a gate and coalesce. If they do not arrive at the same time, a near-miss Coalesce violation will results in electrical errors, and a larger miss Coalesce violation will result in two distinct pulses and thus logically incorrect behaviour.

These hazards are illustrated in figure 5

Timing constrains are inequalities in time that enforce that these hazards do not occur. The first 4 of the are hazards form one sided constraints involving 2 paths, one nominally longer path running fast and nominally shorter one slow. Thus these can readily be expressed as inequalities. Coalescence is a 2 sided constraint, but can be split into two one sided constraints which similarly have a fast and slow path.

To ensure that the timing constraints are unambiguous over all the desired circuit behaviours such as combinational looping, the timing inequalities are expressed as inequalities between paths rather than inequalities between signals. The timing paths are built off a common ancestor. Inside the circuit this is a single pulse that fans out, and later combines with itself. The primary input stimulus as a whole is considered a common ancestor for this purpose, so paths can be expressed from two different input pulses plus the time difference from those two input pulses.

5.1 Example timing constraints

On the binary counter above (figure 3) the reset toggle of the LSB produces the following hold timing constraint:

$$t(clk \rightarrow r[0] \rightarrow d[0]) > t(clk) + t_{hold}(d[0]) \; at \; r[0], s[0]$$

That is the longer path must in fact take longer than the shorter path plus any hold margin the gate requires. While the signals the timing constraint is between is $d[0]$ and clk, the constraint must be enforced not at their outputs but at the inputs of the gates where those signals arise, since otherwise we will omit the wire delay associated with that last wire.

For the next bit in the counter the same constraint becomes:

$$t(r[0] \rightarrow r[1] \rightarrow d[1]) > t(clk) + t_{hold}(d[1]) \; at \; r[1], s[1]$$

Here $r[0]$ is the starting pulse for the paths since that is where they diverge.

The setup constraints on these latches involves the time between one clk pulse and the next, for example for bit 0:

$$t(clk \ldots clk) + t(clk) > t(clk \rightarrow s[0] \rightarrow d[0] \; at \; s[0], r[0]$$

6 HAZARD IDENTIFICATION

6.1 Attributed states

To allow for hazards to be found and paths leading them to be traced, we wish to trace all paths that start from a suitable starting point and recombine at some gate. To achieve this, we attribute the pulses in the nominal state space with speed information.

In addition to the non-attributed definition of a state above, an attributed state contains for each event (pulse or data signal change) that occurs in that state, two bits of attribution information, "fast" and "slow". These bits represent whether the pulse is being considered as part of the fast path or slow path leading to the hazard respectively. Each pulse or data change can be labelled as either fast, slow, both ("colourful") or neither ("colourless"). A colourful event change is one being considered as the starting point of a hazard path in the attributed state to which is belongs. A colourless event is one not being considered as part of any paths in the attributed state to which it belongs.

Given an already attributed state, the attributes for the next state can be computed as follows:

- The existence of pulses and the value of data (and changing status) is computed as for the non-attributed case.
- Attributes are only defined on events that actually occur in the present state.
- An event is labelled fast if it would have occurred if and only if fast input pulses are considered
- An event is considered slow if it has any fast or slow inputs pulses, but considering only the fast ones in the update function does not lead to the event occurring.
- For colorful pulses, each receiving gate chooses to interpret the each colourful pulse as fast or slow independently, and then applies above rules as if the gate.

In addition to the current speed attributes, which are defined in each state only for currently occurring events, each signal is attributed with history bits which are defined for all signals in all states as the previous speed attributes the signal had the last time an event occurred on that signal. Note that on states where the events actually occur the history bits are still defined as having the previous value and are updated in the next state only.

To get an attributed base case from an unattributed "nominal" state, all currently occurring events in that nominal state as colorful, and the history attribute of all signals are set to colourless.

Given these definition of how to take an the nominal states of the system and add starting attributes to them, and also a attributed state update function as defined here, a symbolic state space exploration can be performed as was done to identify the nominal states from the starting states.

6.2 Hazards from attributes

Given this attribution, we can identify which hazards can possibly occur in the circuit by querying the set of possible attributed states for patterns. For each hazard the patterns are:

- A **Hold** hazard exists where a set-reset latch is currently changing fast and a pulse which is ANDed with that data signal at some gate input previously fired slowly.
- A **Setup** hazard exists when a pulse is currently firing fast and is to be ANDed with a data signal at some gate input that currently fired slowly, or previously fired slowly if isn't currently firing.
- A **Retrigger** hazard exists when a pulse is currently firing fast and previously the same pulse fired slowly.
- A **Set-Reset Order** hazard exists when a data signal is currently changing fast and previously changed slowly.
- **Coalesce** hazard exists when one pulse is currently firing fast and another is currently firing slow and these are to be ORed together at a pulse gate.

Having identified which of these hazards potentially exist given the circuit behaviour being considered, the next step is to trace the paths that create these hazards. This will be discussed in the following section.

6.3 Coherence depth determination

The other thing that can be determined from the searching attributed states is the coherence depth of the system. Consider traces in the set of possible traces where on the second step (where colorful pulses sponsor fast or slow ones) one fast pulse exists and other steps slow. If the system is to become coherent again, then either this pulse terminates in a latch and the pulses become monochromatic slow, or eventually all remaining slow pulses will coalesce with this fast pulse and since coalescence becomes fast then the pulses become monochromatic fast.

Assignments on the second step with more than one fast pulse become monochromatic fast if and only if at least one of the pulses they contain becomes monochromatic fast. Second steps that are monochromatic slow remain monochromatic slow. Thus while the important information comes from the one-fast second steps, it is not necessary to filter the search space to just those cases.

Thus the coherence depth can be determined from the number of iterations it takes the pulses in the system to become monochromatic.

7 PATH TRACING

After we identify a hazard we can trace back to find the path that lead to that hazard. Given an attributed state in which we know a hazard occurs, first a series of states can be found leading from a state with starting attributes to the hazard state, and then within this series of states picking a valid succession of pulses to form the path.

Given a current state with a chosen signal, and a previous state we can determine the next pulses back in the path from that signal by looking at all the pulses that fan into the gate producing the signal, and test:

- If the previous pulse exists in the previous state.
- If the data values in the previous state are consistent with the current gates pull-down network admitting that pulse.
- If the previous pulse has the same colouring as the previous state.

Any previous pulse that meets these 3 conditions is a valid candidate in the previous step for having cause the current pulse. When operating these procedures over a set of states rather than just a state, each choice of previous pulse may only apply to some of the previous states in the previous set, so this set must be reduced appropriately.

For the fast path, and the slow path of coalescence and some setup hazards where the slow signal is also presently active in the final state and so this procedure can be applied starting from the final state of the trace and followed all the way back. For the slow paths of the other hazards it is just known from the history attributes that the signal must have been present in the past but it is not present in the current state. Thus for these signals it is necessary to step a few iterations back from the end of the trace before the path of signals to be traced is found. Similarly for paths involving two different primary inputs interacting, the path starting from the later arriving primary input will have a signal trace that starts later than the first state.

Having produced possible paths in this manner, it is necessary to filter them for only paths that have well-conditioned starting points. That is where both the fast and slow signal paths start from the same signal, or both start from (possibly different) primary inputs. This definition serves to remove subpaths of the primary paths of interest that would

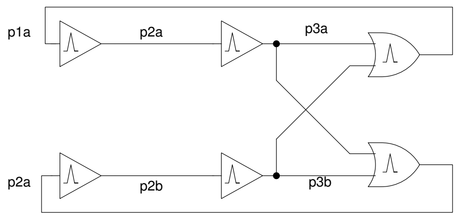

Figure 6: Cross-coupled ring oscillators

over-constrain the timing analysis without contributing to behavioural soundness.

8 IMPLEMENTATION DETAILS

A tool implementing the analysis described here has been implemented, searching the state-space symbolically using Binary Decision Diagrams representations of the automata model described above. First the space of nominal states is established using a standard reachability analysis, then this set of nominal states is attributed with the starting attributes and the attributed state space is then established similarly. To find which hazards exist, the gate database is iterated over to find pairs of signals that are combined and masks are generated which can be used to test the attributed state set.

For tracing back paths, first signal traces of different depths are generated then each signal is iteratively traced back. For each path a breadth first search is performed using a work queue.

To apply inputs to the circuit a simple regular input language generating pulses was used. This allows sequential pulse inputs to be applied. However at present it does not include primary input data changes, hence the results below on the serialiser include the pulse hazards between one clocking of the serialiser and the next but not the setup/hold conditions that would be implied if the inputs changed.

This program was implemented in Python, using the CUDD BDD package through the PyCUDD wrapper.

9 RESULTS

The constraint generation analysis as described was applied to some test circuits. The binary counter and serialiser have already been discussed. A deserialiser (figure 7 based on a tree of Pulse AND gates and toggle latches was also analysed, as was a simple cross-coupled ring oscillator (figure 6) to test coalescence modelling. The results are presented in table 1.

10 CONCLUSIONS

In this paper we have shown a methodology for analysing pulse gate circuits to identify both their nominal behaviour and timing constraints to ensure that behaviour. This was achieved based on taking the structure of the circuit and

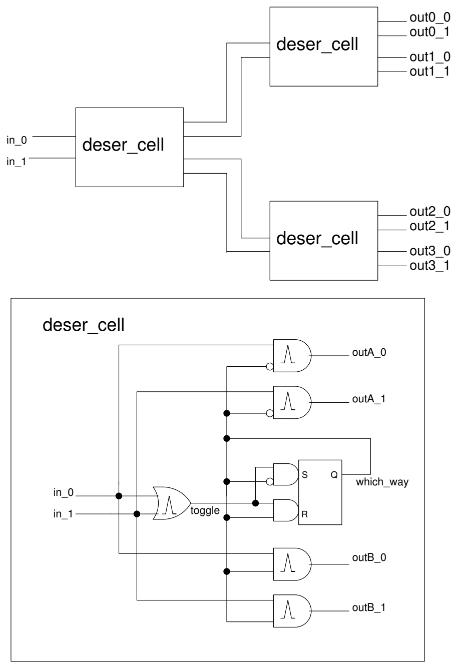

Figure 7: Deserialiser

assuming unit time behaviour. We have identified the possible timing constraints that apply, and detailed a strategy to search for them. Lastly we have implemented this strategy and applied it to some example circuits.

10.1 Future Work

The first area of future work to be undertaken here is to improve the performance of the unit time analysis so that larger local circuits can be analysed. No particular effort has been made to choose a good BDD order, which should lead to immediate improvements. Also presently the entire attributed state space is searched (albeit symbolically), efficiency could be gained by searching only a sufficient subset of possible attributions.

Going forward, it is desired to augment the current analysis so that multiple different pseudo-synchronous islands in a system can be analysed, and the synchronisation between them. This in turn could be used to facilitate certain optimisations or possibly even full synthesis of pulse gate systems.

The timing information generated by either this local or a future system analysis could be used to guide automated layout of pulse gate circuits.

REFERENCES

[1] M. R. Greenstreet and. 2006. Surfing interconnect. In *12th IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC'06)*. 9 pp.–106. https://doi.org/10.1109/ASYNC.2006.28

Circuit	Gates	Constraints	of which					Coherence	Run Time
			Hold	Setup	Retrigger	SR Order	Coalesce	depth	
Counter	12	34	6	6	16	6	0	1	7.8s
Serialiser	29	81	0	0	81	0	0	5	1037.0s
Deserialiser	18	63	15	24	12	12	0	2	11.9s
Ring Osc.	6	28	0	0	22	0	6	4	0.2s

Table 1: Constraints generated from different circuits

[2] A. Dalakoti, M. Miller, and F. Brewer. 2017. Pulse Ring Oscillator Tuning via Pulse Dynamics. In *2017 IEEE International Conference on Computer Design (ICCD)*. 469–472. https://doi.org/10.1109/ICCD.2017.82

[3] G. Hinton, M. Upton, D. J. Sager, D. Boggs, D. M. Carmean, P. Roussel, T. I. Chappell, T. D. Fletcher, M. S. Milshtein, M. Sprague, S. Samaan, and R. Murray. 2001. A 0.18-/spl mu/m CMOS IA-32 processor with a 4-GHz integer execution unit. *IEEE Journal of Solid-State Circuits* 36, 11 (Nov 2001), 1617–1627. https://doi.org/10.1109/4.962281

[4] Merritt Miller, Carrie Segal, David Mc Carthy, Aditya Dalakoti, Prashansa Mukim, and Forrest Brewer. 2018. Impolite High Speed Interfaces with Asynchronous Pulse Logic. In *Proceedings of the 2018 on Great Lakes Symposium on VLSI (GLSVLSI '18)*. ACM, New York, NY, USA, 99–104. https://doi.org/10.1145/3194554.3194592

[5] V. Narayanan, B. A. Chappell, and B. M. Fleischer. 1996. Static timing analysis for self resetting circuits. In *Proceedings of International Conference on Computer Aided Design*. 119–126. https://doi.org/10.1109/ICCAD.1996.569415

[6] Mika Nyström. 2002. *Asynchronous pulse logic.* Kluwer Academic Publishers, Boston.

[7] K. S. Stevens, R. Ginosar, and S. Rotem. 2003. Relative timing [asynchronous design]. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 11, 1 (Feb 2003), 129–140. https://doi.org/10.1109/TVLSI.2002.801606