Unit Time Modelling of Asynchronous and Pulse-Gate Circuits

David Mc Carthy*, Forrest Brewer †
* ECE Department, University of California, Santa Barbara *davidmc@ece.ucsb.edu*
† ECE Department, University of California, Santa Barbara *forrest@ece.ucsb.edu*

***Abstract*—Pulse gates have shown promise as a structured manual methodology for the design of high performance systems. In this paper we present a "unit time" model, identifying the behaviour the circuit from its topology. This model is applicable to asynchronous circuits in general but particularly suitable to pulse gate circuits where it agrees well circuit designer assumptions about high performance circuits. We show that for small-to-medium circuits that have a topological "coherence" property, timing constraints can be produced from the unit time model to ensure behavioural correctness of the circuit. To allow complete systems to be verified, we divide the circuit into regions each of which can be modelled by unit time model. We introduce a notion of communication by "phrases" of events between region that allow coherent communication between separate regions. In this paper, we present a proof that the timing constraints provided by the unit time model are sufficient to ensure behavioural correctness of the circuit in a general bi-bounded sense. Lastly also present a tool that applies this model to pulse gate circuits.**

I. Introduction

Pulse gates[1] 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 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 predictable local timing arcs 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 of more general asynchronous circuits.

The unit time model of pulse gate circuits was originally presented in [2]. In this paper we refine that model generally, and improve it in two specific ways.

- We introduce "phrases" as communication semantics between different regions each modelled by the unit time model independently.
- We prove that timing constraints and properties derived from the unit time model are sufficient to ensure the timing soundness of the system.

In addition to these we present an updated computer tool which implements unit time analysis with phrases, performing timing verification by generating timing path constraints.

A. 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[3]. 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. Further several of the notations used in those paper are based on those of Martin and Nystrom.

Greenstreet[4] 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 pulses, 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[6] (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.

Proteus[7] is a logic synthesis methodology for dynamic domino pipelines, which intentionally sizes the domino gates to have a unit delay. This allows for timing optimisation at a few-gate level but larger timing synchronisation is still achieved in a Quasi-delay-insensitive manner

Relative timing[8] 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 rules used in that work imply that each signal is only used once per analysis scope, while in our pulse gate circuits, designers frequently include looping behaviour even on a local level.

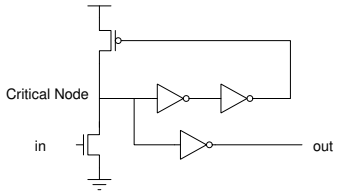

Fig. 1. Pulse gate implementation

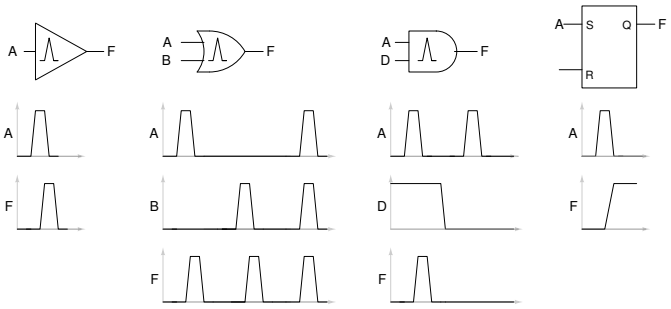

Fig. 2. Pulse gate logic elements

A different pulse gate timing algorithm for design style discussed in this paper appeared in [1]. This algorithm requires identification of frames where each gate only fires once, this technique did allow for looping behaviour (i.e. self-circuit re-triggering), however, the manual placement of timing frames allowed for potentially missing possible behaviors. Further, the model is incapable of modeling pulse absorption, a behavior used in stabilizing high performance clock phase generators.

II. Background

A. Pulse gate Circuits

1) Pulse Gates: A pulse gate circuit consists of two types of gates, the pulse gates themselves and pulse-actuated SR 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[9]. Thus the pulse shape is largely dependent only on actuation arrival time, and not slope or amplitude.

In addition to the basic pulse buffer shown, logic in pulse gates can be obtained in two basic ways. One is by ANDing a pulses with a static level from a SR latch, or a more involved guard combining multiple latch values, producing an output pulse only if the latches have a required value when at the time the input pulses arrive. The other is by ORing two different arriving pulses together, producing an output pulse when either input pulse arrives. Both of these are achieved in the NMOS pull down network. Latches are typically CVSL-style SR latches sized to be correctly actuated by a pulse. Earle latches with logic in the latch pull down network can also be built. Figure 2 shows the basic pulse gate logical elements.

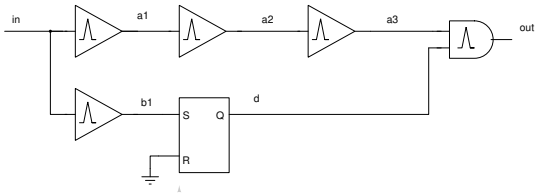

Fig. 3. Example fragment of a pulse-gate circuit

Several more involved pulse gate behaviours can be constructed for synchronisation. "Coalescence" is where an pulse-OR gate is activated by two incoming pulses nearby in time and produces one output pulse instead of two. A pulse consensus gate has two input pulses, and it only produces an output pulse after both inputs have been activated. A pulse conjunction gate only produces an output pulse when it receives two simultaneous input pulses.

2) Pulse gate circuit construction: Pulse gate circuits are strictly typed so that timing information is strictly derived from pulse gates, and not from latch outputs which are only used for guards.

Pulse gates have low variance, both in terms of the spread of different possible nominal delays of different pulse gates, and also the possible PVT variation of gates in implementation compared to that nominal. Thus pulse gate circuits can be designed with timing arcs that are assumed to remain in the order they occur in the nominal case, and further that this nominal-case order is that the would be suggested topologically, i.e. the event that has more gates leading to it occurs later.

Consider the circuit fragment shown in figure 3. It is reasonable to assume that because the topological path to $a3$ is longer than that to d, the new value of the latch output d will be setup in time for the new value to be considered when pulse $a3$ arrives at the gate producing out.

This contrasts with, for example, burst-mode design where the output depending on the order of evaluation of the gates would be disallowed, or QDI design where a handshake join would be added to ensure both d and $a3$ are computed before out is.

3) Timing Constraints: To ensure that the assumed behaviour in the nominal case is what actually occurs, we must impose timing constraints on the above circuit. For the above example setup constraint we might have:

$$t(d) + t_{setup} < t(a3)\ at\ out$$

For recurrent networks, it is useful however to express this as a path constraint. This can be achieved by tracing back causality of these events to a common ancestor.

$$t(in \to b1 \to d) + t_{setup} < t(in \to a1 \to a2 \to a3)\ at\ out$$

4) Timing Hazards: There are two possible ways in which a pulse-gate circuit can escape its intended function. The first is that an electrical error in the pulse gates can occur, where the assumptions about its input types are violated and cause the pulse output to malfunction, e.g. by producing runt or overly wide pulses, or two output pulses for one input. Logical errors

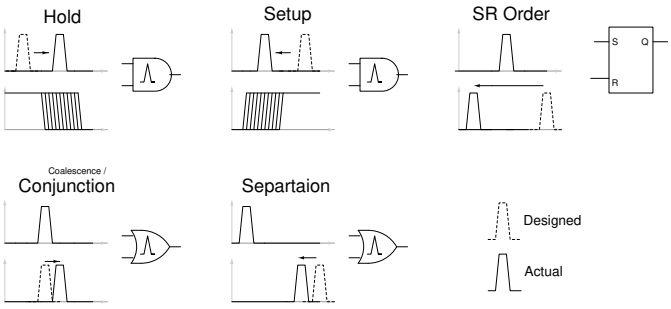

Fig. 4. Pulse gate Hazards

occur when pulse gates function electrically correctly but the system behaviour is different from what was designed. Both these types of failures are produced from the same hazards. Whether the failure is electrical or logical depends on the degree to which the assumption is violated. There are 6 types of pulse gate hazards, shown in figure 4

- 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 ANDed 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 is meant to receive a set and reset activation in a certain order, but that order changes so the latch ends up having the wrong value.
- A **Coalesce** hazard exists when two pulses are meant to arrive at the same time at a gate and coalesce.
- A **Conjunction** hazards is similar to a Coalesence hazard, except for gates where the arrival of both pulses is required to produce the output.

B. Unit time Model

Taking the low-variance topological assumption further, 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 they will have the resultant pulses and those will also be almost simultaneous. More specifically, the designer assumes that while the circuit will actually have some timing dispersion, that local behaviour will be the same as if the gates were moving in lockstep.

Formalising this, the unit time model of a pulse gate circuit (or a region of a pulse gate circuit) is a model defined by simulating gate-level abstract timing behavior on the circuit connections. The behaviour of a circuit is described as sequences of discrete "states". A *state* is defined as the set of pulses present simultaneously in the circuit, as well as the value of the data signals at that time. States are separated by the local sequencing of individual gates. The successors of a state are defined as the sets pulses present in the circuit simultaneously one nominal gate delay later, with latches updated if they have accepted pulses.

C. Coherence Depth

The coherence depth of a circuit, or a region of a circuit is, from any given state in the unit time model, how many steps forward before all the resultant events in the new state have a common ancestor event in the first step, or have timings as if they did.

If we inject a timing dispersion into the pulses in the same unit time state, then after the coherence depth states later the circuit will have absorbed that dispersion and the relative timing between the pulses in that later state will be the same as if no dispersion had been injected.

For small coherence depth only limited variance buildup is possible. For larger coherence depth more timing spread between events considered in the same state is possible, but still bounded.

If the a circuit is such that the behaviour can fork into separate regions which do not communicate with each other for a long time, then the timing dispersion between these two regions will be large, much larger than the timing dispersion within either region. It is possibly unbounded if the regions do not communicate for an indefinite amount of time.

Note that this definition of coherence depth is consistently 1 smaller than the defintion in [2] since we no longer count the starting state.

D. Production Rules

One gate level description Asynchronous circuits, including our pulse gate circuits, is as the production rules of the gates. These are a set of pairs of input preconditions and the output event they cause. For level-based asynchronous circuits the output events are rising and falling transitions, but for pulse circuits the production of an atomic pulse is also a possible result of a production rule. Similarly the guard for level-based circuits is a boolean function of input signals, where for pulse circuits the guard may also include pulse signals.

When a production rule becomes satisfied and produces an event, let the event that caused the production rule to become satisfied be called the "causal event". The design rules for our pulse circuits can be defined as that each gate produces either a pulse event but no static state (and thus its event can be causal), or a data output (static state) with transition events that are never casual.

For multiple events arriving near-simultaneously at a gate, causality can be determined by applying them one-by-one and seeing which one results in the output event being produced. The result of this procedure should result in the same output regardless of the order. Simultaneous arrival of events that would result in different outputs depending on order should be avoided (e.g. data change arriving at the same time as a pulse in a pulse AND gate).

III. Phrases

The applicability of the unit time model is dependant on the coherence depth being small. Trying to build a unit time model of a large system will have an excessively large coherence and it would instead be better to build unit time models of different regions of the circuit and then model the communicating between those regions. Thus it is necessary to extend our unit time model to include communication. To describe our circuits of interest adequately, it is necessary that coherence also be available across communication.

A "phrase" is a unit timed series of pulses and data-changes that represents one communication interaction between two regions of a circuit. A phrase is a regular language whose alphabet is the sets of signals, which shows the patterns of unit timed signal events that consisting the communication being described.

Phrase languages can be expressed using a variation of the Martin's Handshaking Expansion[10] syntax, where the ";" then operator means exactly one unit time step later, and $\varnothing$ describes a unit time step with no events crossing the region boundary.

Each phrase also has a coherence depth. This is the largest distance back between common ancestors of any pair of nominally simultaneous signals in the phrase, or between a current event and the unseen progenitor of a future event. This is at most the coherence depth of the region generating the phrase, or it may be smaller depending on the structure of the region.

For example a simple serial link system (serialiser and deserialiser) can be described with 3 phrases. The first is the "go" phrase to start the serialiser. This has a language of just $go;$. The coherence depth of this is 0, since the single event has no dispersion relative to itself. The "done" phrase of the deserialiser is similar. In practice the "go" and "done" phrase may also include the setup of the input and output data latches respectively. They would remain one state long but that would have additional signals.

The more interesting phrase is the one describing a word on the pulse-based serial link. We here consider a 4-bit word with 4 unit gate delays between each symbol. This described by the language:

$$(ser0|ser1);\varnothing;\varnothing;\varnothing;(ser0|ser1);\varnothing;\varnothing;\varnothing;$$
$$(ser0|ser1);\varnothing;\varnothing;\varnothing;(ser0|ser1);$$

A. Regions by cut

Given a set of phrases for a system we can identify the regions associated with each phrase automatically. We start with the signals of that phrase and flood fill up to the boundaries defined by the signals of other phrases. Thus we obtain a region that obtains receives that phrase, computes on it, and emits zero or more output phrases (of the one or more topologically possible output phrases found by flood fill). This flood fill is shown for a SerDes system in figure 5

In general leads to overlapping regions, where two distinct phrases fan out into the same gates. Further this overlap may mean that multiple technically distinct regions may produce

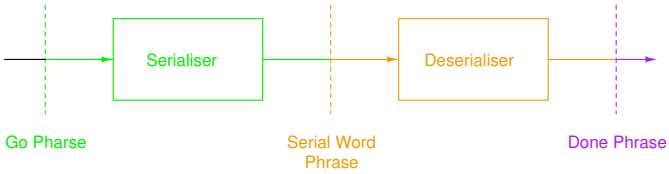

Fig. 5. Phrases dividing up a SerDes circuit and regions derived from them

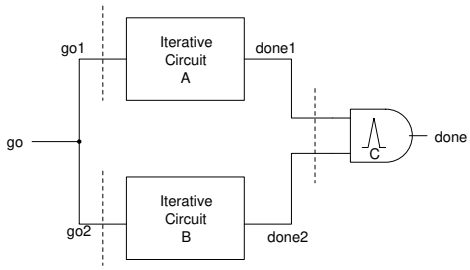

Fig. 6. System with non-trivial control flow

the same output phrase or phrases. Thus at any time we assume that while a particular phrase is being processed in a region, that the behaviour sponsored by that phrase is the only activity in that region. All phrase activity from other phrases whose fanout region overlaps must occur definitely before or definitely after the phrase under consideration. System-level constructions for enforcing this are beyond the scope of this paper.

B. Completeness

To see that a phrase-based model can reasonably model general systems, we assert the following:

- Control forks can be achieved with a region that has one input phrase and two output phrases.
- Control joins can be achieved with two single-pulse phrases whose regions overall and start with a consensus gate. The first pulse to arrive arms that leg of the consensus gate, while not propagating to the rest of the region and thus producing no output phrase. The second then fires the consensus gate and produces further action and an output phrase. Figure 6 illustrates these two possibilities.
- Data can be passed in several ways. 1-hot encoding of data is useful for long physical distance communication. Pipeline like data passing can be achieved by having the data as part of the phrase cut. Also phrases can read "global" data from registers assuming that it can be guaranteed that no other phrase that changes that data is co-executed.

C. Timing Constraints across regions

In tracing back the paths of timing constraints in one region, we may find that the paths trace back to a pair of distinct pulses in the phrase. If this occurs, we can rearrange the inequality into the form of $t2 - t1 > ....$ Then we can in all possible source regions of that phrase trace all paths and enforce that requirement on $t2 - t1$.

D. Unit time model of production rule systems

The unit time model of a gate circuit specified by productions rules is a finite automata model based on the topology of the circuit. A state in the unit time model is the product of the following factors:

- For all signals, whether an event is occurring on that signal for that state.
- For signals that have a level, the value of that level (after the transitions have been taken into account).
- Long term internal state of any gates that have such (e.g. pulse consensus gates).
- The generating machine of the input.
- The recognising machines of the output phrases.

The next state of the unit time model is defined by evaluating the productions rules of each gate over the current state of the system to determine which events should occur, and then apply the transitions to determine the new value of levels.

E. Unit time model of phrases

In the unit time model, a phrase is represented as Moore machine whose outputs are the events and levels of those signals. For the generating machine, the machine is deterministic but is a generator (so can transitions non-deterministically). The initial state is (arbitrarily one of) the first state of the language. The final state is a trap state which loops forever producing no events.

The output recognising machine or machines are non-deterministic recognisers. They can start in either a waiting state to wait for output, or in the first state of the language for cases where the output begins immediately. An accepting state which loops as long as no unexpected trailing output events are produced is appended.

IV. Some Definitions

A. Bi-bounded delay model

For considering the actual rather than unit time behaviour of a region of a circuit, we use a bi-bounded delay model. In this model each gate g, upon its production rule becoming satisfied by an event at time t_{in} it produces its output event (pulse or data change) at time t_{out} such that $t_{in} + t_{min,g} < t_{out} < t_{in} + t_{max,g}$. Let $T_{min} = \min_g t_{min,g}$ and $T_{max} = \max_g t_{max,g}$

When consider the bi-bounded model in conjunction with the unit time mode, for an event on signal g at timestep k in the unit model, let $t(k, g)$ be the time that the corresponding event occurs in the real time model.

For gates with multiple near-simultaneous input events, the output timing is based on the one which is causal by the above definition. For a pulse OR gates performing subsumption with multiple input events $t_{in,1}, t_{in,2}, \ldots$, the gate is activated by the first event. Thus $t_{in,eff} = min(t_{in,1}, t_{in,2})$. We refer to this as a delay-minimising gate. For gates performing conjunction, the gate is activated by the second pulse. Thus $t_{in,eff} = max(t_{in,1}, t_{in,2})$. We refer to this as a delay-maximising gate.

For the events in the input phrase, the phrase is generated by a timed extension of the usual generating automata. It takes each state transition internally for some time in between T_{min} and T_{max}. Let this state time represent the earliest time of the events in the generating machine, noting this event may not be visible at the region boundary. For a input phrase being generated with coherence depth D_{IN} then the visible events of the phrase are generated in the bi-bounded model with times $t_{state} < t_{event} < t_{state} + D_{in} * (T_{max} - T_{min})$. We will see later where this bound comes from. The input timing model may provide additional timing guarantees between specific pairs of signals, either measured by another instance of this theory or specified by another means.

B. Coherence Depth Definition

To formally define coherence depth, we will work from the injected perturbation view of coherence discussed above. We define coherence depth this way since it can be measured from the circuit. Later, theorems 4 and 3 will then show that the "common ancestor" definition is equivalent to this.

The ranked-order labeling is an augmentation of the unit time model to carry information about how variance propagates. A state in this rank-order labeled model consists of a unit time state plus a rank label $l_r(g)$ added to those signals that have an event occurring in that timestep. Rank labels are small positive integers, with lower values representing variance earlier in time and larger values later. Labels are not applied to signals where events are not occurring in a given time step.

The ranked-order labeling is added to trace under the unit time model, starting from some state index k_0 in that trace. For the first state of the ranked-order, unique labels are added to each event that occurs in that state. When considering models with an input phrase, the input events that occurs in the start-of-labelling state and the next $D_{IN} - 1$ states are assigned unique labels, and one label is assigned to all events thereafter. Here D_{IN} is the coherence depth of the input phrase.

If the unit time trace is being considered together with a real-time trace then $l_r(k_0, g_1) < l_r(k_0, g_2)$ iff $t(k_0, g_1) < t(k_0, g_2)$. If a unit time trace is being considered in isolation, all possible orderings are considered.

For the next state of the rank order labeling, the unit time state component is determined as for the regular time model. If gate g_n produces an event:

- For a single causal event g_1 activating the gate, $l_r(k, g_n) = l_r(k-1, g_1)$
- For a delay-minimising gate g activated by two potentially causal input events g_1 and g_2, $l_r(k, g_n) = min(l_r(k-1, g_1), l_r(k-1, g_2))$
- For a delay-maximising gate with input g_1 and g_2, $l_r(k, g_n) = max(l_r(k-1, g_1), l_r(k-1, g_2))$

Definition 1. *Suppose k_0 is a time index into a trace T_U of the unit time model of a circuit region C. Suppose $T_{U,k0}$ be a trace with a ranked order labeling L_R applied starting from step k_0. If there exists a k_d such that all the labels labels are equal to each other in $T_{U,k0}[k]$ for all $k >= k_d$, then $d = k_d - k_0$, otherwise let $d \to \infty$.*

The coherence depth is defined as

$D = \max_{Tu} \max_T \max_{k_0} \max_{L_R} d$

C. Single-ancestor timing

For a given region of the circuit we say that it satisfies single ancestor timing if for each gate in the circuit, whenever that gate fires, the fanout of that gate does not create timing issues in and of itself. One way in which this might be enforced is by a set of path constraints as described in section II. We will show later (theorem 3) that we need only consider those paths where the shorter length is at most the coherence depth D of the circuit. This also bounds the length of the longer leg to be considered until the inequality is trivially true.

When considering input signals into a region, the 'single ancestor' may be a pair of signals possibly even on different timesteps. These have an unknown common ancestor in the previous region. The definition of the input timings into the bi-bounded model is generalised to allow for this possible common ancestor.

V. Timing Containment

In this section, we prove that the unit time model accurately models the behaviour of a circuit or region of a circuit assuming that the circuit has a well-defined coherence depth and an appropriate set of single ancestor timing constraints. We do this by establishing correspondence between any given real time trace and a unit time trace.

Definition 2. *A real-time trace T_B and unit time trace T_U are in correspondence if:*

1) *A one-to-one mapping can be made between the events that happen in the bi-bounded trace and the events that happen in unit time trace.*
2) *For two events $t1$ and $t2$ from the unit time model that are in correspondence with the same time-step of the unit time model, $|t2 - t1| < D * (T_{max} - T_{min})$.*

A pair of traces can be said to be in correspondence for all time, or up to a given time-step n.

Theorem 1. *Given a circuit C with finite coherence depth D that satisfies single-ancestor timing constraints, any pair of unit time trace T_U and bi-bounded trace T_B from the same initial conditions are in correspondence for all time.*

This will be established by induction. First, we will build our base case. Then, we will build our inductive step in three parts. We will show that if we know that the traces correspond up to $n-1$ and the one-to-one mapping exists for step n, then step n also satisfies the timing bound and thus is also in correspondence. We then show a tighter timing bound. Finally we show that given these results the one-to-one mapping exists for step $n+1$.

Theorem 2. *For a circuit C, any pair of T_B and T_U are in correspondence for $n = 0$ and corresponding events exist for timestep $n = 1$.*

Proof. The circuit can have no internal events at $n = 0$, and thus the first part of the statement follows directly from the definition of the how inputs are applied to the bi-bounded model. The second part of the statement follows from the single ancestor timing model applied to these input pulses. □

Theorem 3. *Sps. that we know that T_B and T_U have coherence depth D, corresponding events for timesteps $k = 0..n$ and the delay bound from theorem 1 applies for timesteps $k = 0..n-1$.*

Then for two events $t1$ and $t2$ in timestep n, $|t2 - t1| < D * (T_{max} - T_{min})$.

Proof. Let $k_0 = n - D$. Let T_R be a rank-order trace derived from T_U with the rank orders applied at timestep k_0 in conformity with T_B. We will show by induction for $k >= k_0$ that if $l_r(k, g) = r$ and $t_r = t(k0, g_2)$ such that $l_r(k, g_2) = r$ then

$$t_r + T_{min} * (k - k_0) <= t(k, g) <= t_r + T_{max} * (k - k_0)$$

For $k = k_0$, all signals have distinct labels and this statement collapses to $t(k, g) <= t(k, g) <= t(k, g)$ which is trivially true.

Assuming the bound is true for all gates up to timestep $k-1$, consider a gate g_n in timestep k, we have 3 cases:

Case 1: If g_n has a single causal event g_1 sponsoring its behavior, then $l_r(k, g_n) = l_r(k-1, g_1)$ and $t(k-1, g_1) + T_{min} <= t(k, g_n) <= t(k-1, g_1) + T_{max}$. Thus

$$t_r + T_{min} * (k - k_0 - 1) + T_{min} <= t(k, g_n) <= t_r + T_{max} * (k - k_0 - 1) + T_{max}$$

Case 2: If g_n is a delay-minimising element for two input causal events g_1 and g_2. Let $t(k-1, g_1) < t(k-1, g_2)$. Thus $t(k-1, g_1) + T_{min} <= t(k, g_n) <= t(k-1, g_1) + T_{max}$ and thus:

$$t_{r1} + T_{min} * (k - k_0) <= t(k, g_n) <= t_{r1} + T_{max} * (k - k_0)$$

If $l_r(k-1, g_1) <= l_r(k-1, g_2)$ then $l_r(k, g_n) = l_r(k-1, g_1)$ and this is directly what is required. If instead $l_r(k-1, g_2) < l_r(k-1, g_1)$ then $l_r(k, g_n) = l_r(k-1, g_2)$. The case assumption implies $t_{r2} <= t_{r1}$. Thus $t_{r2} + T_{min} * (k - k_0) <= t_{r1} + T_{min} * (k - k_0) <= t(k, g_n)$

For the other side of the inequality, consider $(k-1, g_1) < t(k-1, g_2)) <= t_{r2} + T_{max} * (k - k_0 - 1)$ and thus $(k, g_n) < t_(r_2) + T_{max} * (k - k_0 - 1) + T_{max}$ giving the desired inequality in the second case.

Case 3:If g_n is operating as a delay-maximiser of two input events g_1 and g_2 then a similar argument applies in the opposite direction.

By the definition of coherence depth, we know that all $l_r(k, g)$ are equal for $k >= k_0 + D = n$ then for $k = n$ we have that

$$\exists t_r \forall_g \Big(t_r + T_{min} * D <= t(k, g) <= t_r + T_{max} * D \Big)$$

Thus any two events in the state n have real times that are at most $D * (T_{max} - T_{min})$. □

Theorem 4. *Assuming T_U and T_B are in correspondence up to time step n, we have that for two events $g1$ and $g2$ in timestep n at times $t1$ and $t2$, over all possible ancestors $g3$ in timestep k_0 then:*

$$t2 - t1 <= max_{g3} \Big(t_{max}(g3...g2) - t_{min}(g3...g1) \Big)$$

Remark. *From theorem 3 we have established the weaker result that* $t_2 - t_1 <= D * (T_{max} - T_{min})$.

Proof. From the definition of coherence depth, $g1$ and $g2$ have a common labelling ancestor $g3$ in timestep $k0$. Let $g4$ be another event in the starting timestep. Supposing the label of $g4$ is later than that of $g3$. The fact that the label of $g4$ is eliminated means one of:

- There exists two delay minimising elements on the paths from $g4$ to $g1$ and $g2$ that chose the label of $g3$ over that of $g3$,
- $g4$ does not have a path to influence the labelling of either $g1$ or $g2$.
- A mixture of delay minimising and maximising elements exist that always choose the label of $g3$ over that of $g4$

For the delay minimising case, let $g5$ and $g6$ be the delay minimising elements on the paths to $g1$ and $g2$ respectively, from both $g3$ and $g4$. If for the times in the bi-bounded trace $g5$ and $g6$ both choose $g3$ then we have our result with $g3$ as the chosen ancestor gate. If $t4$ decreasing causes $g6$ to choose $g4$ as its ancestor instead, the inequality remains true since $t2$ is reduced. If $t4$ decreasing causes both $g5$ and $g6$ to choose it instead of $t3$ the inequality still holds but now with $g4$ as the ancestor gate. If $t4$ decreasing causes $g5$ to choose $t4$ but $g6$ to choose $t3$ then again we have our inequality still holding with $g4$ at the common ancestor and $t2$ for that case reduced by the fact that $t3$ pulled $g6$ earlier.

In the case where the label of $g4$ does not affect the labels of $g1$ and $g2$, the same lack of path prevents the actual timing from affecting $t1$ and $t2$, so we have our result directly. For the mix of minimising and maximising cases, if these elements are arranged to always choose the label of $g3$ then they will also always choose its timing.

For cases when the label of $g4$ is earlier than $g3$ a similar argument holds with delay maximising elements instead of delay minimising elements.

□

Theorem 5. *Assuming traces* T_B *and* T_U *are in correspondence up to timestep* n*, then a one-to-one mapping exists for events in timestep* $n + 1$

Proof. For each gate g_n in timestep $n + 1$, the unit time model makes a prediction that that gate is about to fire in the next step, and any current step causal events that could cause the next gate to fire (under some non-causal preconditions). For each of those causal, for each data precondition g_2 in those guards, to ensure the bi-bounded model to have the same behaviour we require $t_{g_1,n} >= t_{g_2,n} + t_{su,g_n}$ and $g_1, n <= t_{g_2,next} + t_{hold,g_n}$

For the setup constraint, we know from our assumed single ancestor timing model that the constraint can be met if $t_1 - t_2$ is constrained to the maximum of its single ancestor value. From theorem 4 this is the case.

For the hold constraint, there are two possibilities. One is that the next event to change g_2 is the same next event as predicted by the unit time model through a path predicted by the unit time model. Thus, like the setup case, this is covered by timing constraints. The other is the hypothetical creation of an unexpected earlier event on g_2 but this requires a timing failure at the inputs of g_2 which would be covered by timing constraints.

Thus for each g_1 and g_2 driving gate g_n the single ancestor timing model is sufficient to ensure that the gate performs the same in the bi-bounded model as predicted by the unit time model. □

Now we can prove **Theorem 1**.

Proof. By induction. Theorem 2 gives us conformance for $n = 0$. Applying theorems 5 and 3 in turn gives us that if conformance exists for timestep n then it exists for timestep $n + 1$ □

A. Output phrase conformance

For each output phrase the region produces, the region may have a coherence depth D_{out} that is smaller than that of the whole region, for example if the region includes extra latches to buffer its outputs. This can be determined by a similar labelling process to the whole region but looking only at the output phrase's signals. Then a similar argument to theorem 1 will give that the events have the necessary timing variance since the agreement of labels gives a common ancestor.

VI. Tool Implementation

A tool leveraging this analysis was developed. The algorithms implemented by this tool are largely similarly to those in the previous version of the tool [2], so we will only describe the differences. Firstly phrases are implemented, so given a description of the phrases in a system the circuit will be divided up into regions and each analysed separately.

A. Fast-Slow labelling

The fast-slow labeling is an augmentation of the unit time model, similar like the ranked order label discussed above. Only two labels are used of $Fast$ or $Slow$, representing relative timing deviation. The fast-slow labelling is used instead of the ranked order labelling for computational efficiency. The application and update rules are similar to those for the ranked order model.

Multiple passes of the fast-slow labeling can be applied to model a rank-order labeling L_r. Let L_f, i be a fast-slow labeling where $l_f, i(k_0, g) = Fast$ iff $l_r(k_0, g) <= i$. Then for $k > k_0$ we can determine the ranked order label. If $l_f, i(k, g) = Slow$ and $l_f, i+1(k, g) = Fast$ then $l_r(k, g) = i$.

B. Coherence depth measurement

Coherence depth measurement is done by trial of different coherence depths, and applying a BDD-based CTL model check to see if all labels are the same. This is performed using the NuSMV tool. Use of bounded model checking was also investigated but found to be less efficient. The input coherence depth of each region is initially assumed to be 1, or can be user specified to be higher. Output coherence depths are measured similarly. If those output coherence depths feed into a region, and tell us the coherence depth of that phrase is higher than

Circuit	Setup	Hold	SR Ord.	Retrig	Thru
3-bit counter	2	8	2	8	0
4-bit Serialiser (tree)	3	0	0	16	24
4-bit Serialiser (linear)	3	0	0	20	24
4-bit Deserialiser	80	89	40	48	0
4-bit Ripple Carry Adder	240	0	0	0	0

TABLE I
DETAILS OF THE CONSTRAINTS FOUND IN SOME REGIONS OF THE TEST CIRCUITS

previously known we must retest that region to see if the increased input coherence depth increases the region or its outputs coherence depths. This is iterated until upper fixed point.

C. Hazard tracing

The single ancestor timing constraints are found as in previous work[2] by identifying possible pairs of signals that might be a hazard from the circuit topology, and searching backwards in unit time for possible path pairs that lead to them. BDDs are used to ensure that states exist that lead to this trace. This uses internal BDD structures not NuSMV.

Having identified possible backward paths, they are further checked by a forward search that includes speed labelling to see if the path is made redundant by synchronisation. This test here is applied with CTL in NuSMV.

D. Hazards between regions

Where a hazard in a region is traced back to the input boundary, this timing constraint on the inputs is back propagated into the region that produced that phrased and traced there as a new hazard. These are called 'through' hazards. That the correct signals occur is tested in the backtrace, that the output state agrees with the input state in the next is checked during the CTL recheck of the path.

E. Tool Results

A tool implementing the analysis discussed here was implemented, and a set of test examples constructed. A 3-bit counter is a very basic example. Two different varieties of 4-bit pulse ser-des system were designed, one with a linear serliasiser and one with a tree serialiser. Both use tree deserialisers. These are from [11]. A 4 bit ripple carry adder is also tested, implemented using a one-hot design style. A version of the the circuit from figure 6 above was also tested, with the iterative circuits being 3-bit counters looped on themselves until they overflow to achieve a delay of approximately 40 cycles. Table I shows the number of hazards for some key regions of interest in these circuits. Runtimes ranged from 3 sec for the binary counter to 32 sec for the SerDes system with the linear serialiser.

VII. CONCLUSIONS

To the established unit time model of pulse gate circuits we have presented two extension.

First we have introduced "phrases" as a mean of specifying the communication between different unit timed parts of the circuit without enforcing the unit time throughout. These allow much larger circuits to be analysed where it would be inappropriate to analyse the whole thing as a pulse gate circuit.

Secondly, we have proved that a simply-generated set of timing constraints (those arising obviously from a single ancestor gate) and the coherence property of pulse gates circuits, together are enough to ensure the entire system is sound from a timing perspective and performs as predicted by the unit time model.

A proof of concept tool has also been shown, using a path tracing approach to perform timing verification inside and across region boundaries.

While the analysis here was dedicated to pulse gates, the unit time model and proof have been stated in sufficiently general terms that they could be applied to other asynchronous systems readily.

A. Future Work

The local analysis presented in this paper assumes that at a system level overlapping phrases do not co-execute The most immediate future work will be applying some other analysis at the high level to prove the required system level behaviour.

The theoretical framework developed in this paper applies to any "single-ancestor" timing verification. Other such verification will be investigated to see if they can address larger circuits than possible with the path tracing verification.

REFERENCES

[1] M. Miller, C. Segal, D. Mc Carthy, A. Dalakoti, P. Mukim, and F. Brewer, "Impolite high speed interfaces with asynchronous pulse logic," in *Proceedings of the 2018 on Great Lakes Symposium on VLSI*, ser. GLSVLSI '18. New York, NY, USA: ACM, 2018, pp. 99–104. [Online]. Available: http://doi.acm.org/10.1145/3194554.3194592

[2] D. McCarthy, M. Miller, and F. Brewer, "Automated Timing Constraint Generation for Pulse Gate Circuits," *IWLS Workshop, Lausanne, Switzerland*, 2019.

[3] M. Nyström, *Asynchronous pulse logic*. Boston: Kluwer Academic Publishers, 2002.

[4] M. R. G. and, "Surfing interconnect," in *12th IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC'06)*, March 2006, pp. 9 pp.–106.

[5] V. Narayanan, B. A. Chappell, and B. M. Fleischer, "Static timing analysis for self resetting circuits," in *Proceedings of International Conference on Computer Aided Design*, Nov 1996, pp. 119–126.

[6] 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, "A 0.18-/spl mu/m cmos ia-32 processor with a 4-ghz integer execution unit," *IEEE Journal of Solid-State Circuits*, vol. 36, no. 11, pp. 1617–1627, Nov 2001.

[7] P. A. Beerel, G. D. Dimou, and A. M. Lines, "Proteus: An asic flow for ghz asynchronous designs," *IEEE Design Test of Computers*, vol. 28, no. 5, pp. 36–51, 2011.

[8] K. S. Stevens, R. Ginosar, and S. Rotem, "Relative timing [asynchronous design]," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 11, no. 1, pp. 129–140, Feb 2003.

[9] M. Miller, G. Hoover, and F. Brewer, "Pulse-mode link for robust, high speed communications," in *2008 IEEE International Symposium on Circuits and Systems*, May 2008, pp. 3073–3077.

[10] A. J. Martin and M. Nystrom, "Asynchronous techniques for system-on-chip design," *Proceedings of the IEEE*, vol. 94, no. 6, pp. 1089–1120, 2006.

[11] M. Miller, "Realization and formal analysis of asynchronous pulse communication circuits," Ph.D. dissertation, Dept. Elec. and Computer Engineering, Univ. California Santa Barbara, 2015.