Behavioral Simulation of Relative Timed Asynchronous circuits

Sumanth Kolluru, Kenneth S. Stevens
Electrical and Computer Engineering, University of Utah

***Abstract*—Relative Timed design represents timing constraints in an integrated circuit as mathematical equations. This differs from current state of the art integrated circuit design methodologies and electronic design automation tools which employ a finite state machine methodology by using a clock to implement circuit timing. This paper presents a behavioral model for relative timing constraints that enables simulation and evaluation of relative timed circuits written in behavioral Verilog. The model enables the source behavioral Verilog to be directly synthesized into relative timed circuits. The model provides support for rapid design exploration, architectural timing experiments, design validation, and multi-mode behavioral and structural simulations. Experiments show a 50× speedup over current Verilog designs that require a structural implementation.**

I. Introduction

The correct behavior of an integrated circuit (IC) depends on two related but independent domains: logic and timing. Failures in either of these domains will result in an inoperative chip.

Traditional IC design flows implement chip timing as a finite state machine methodology where the next state updates on the edge of a clock. This work focuses on an alternative timing methodology called relative timing (RT) that implements timing as the mathematical equation shown in Eqn. 1 [1]. This equation states that from a common timing reference, called a point of divergence or pod, the maximum delay to one point of convergence (poc_0) is always less than the minimum delay to another point of convergence (poc_1). This ensures that the poc_0 event always occurs before the poc_1event with minimum separation specified by the margin m. This effectively reduces the reachability graph that defines possible circuit behavior.

$$\mathsf{pod} \mapsto \mathsf{poc}_0 + m \prec \mathsf{poc}_1 \tag{1}$$

If relative timing is coupled with handshaking, modular design methods can be implemented. For example, a single stage pipeline is shown in Figure 1. The handshake employs request and acknowledge signaling between `LC` modules. The `LC` modules implement a sequential handshake protocol that controls data transfer between register banks. Relative timing ensures that the delay from pod to poc_0 (the red line) is always less than the delay from pod to poc_1(the blue line). RT constraints guarantee that these pipelined circuits operate correctly in the timing domain, while the handshake protocol implemented in the `LC` blocks guarantee that the circuit operates correctly in the logic domain. By combining sequential handshake protocols with relative timing constraints, arbitrary pipelines, including fan-in and fan-out can be implemented and verified to operate correctly in the logic and timing domains.

Verification of a relative timed handshake design proceeds as follows. Handshake protocols are specified with a formal language (such as a Petri Net) and synthesized to a standard cell library [2], [3]. This gate level sequential circuit is formally verified using model checking applying an unbounded delay model against the specification. This verification generates a set of RT constraints that must hold that allow the circuit to conform to the specification correctly [4]. Modular handshake designs are implemented using these `LC` modules which are formally proven correct in both the logic and timing domains. These designs are now combined with combinational functions and composed in pipelines to implement complex functions. The circuit is fully verified and operational when the combinational functions are verified as well ans the handshake control and timing [5], [6]. For example, the pipeline in Figure 1 implements the x^3 function.

Synthesis flows have been developed to create relative timed circuits whose internal delays meet their timing constraints [7]. Handshake RT designs are synthesized by placing a maximum delay constraint between the pod and poc_0 that is less than a minimum delay constraint between the pod and poc_1 [8], [9]. These behavioral circuits can be simulated and validated to ensure correct behavior and that the relative timing constraints have been met. Timing verification tools such as PrimeTime are employed to ensure these constraints hold in the final implementation. The RT modules are characterized to be compatible with clocked CAD flows, using Synopsys Design Constraints (SDC) commands like `set_size_only`, `set_dont_touch`, `set_data_check`, `set_max_delay`, `set_min_delay` and `set_disable_timing`. These constraints help preserve the functional correctness of the asynchronous design during the synthesis step.

This paper provides behavioral models for relative timing constraints (RTCs), allowing designers to simulate the designs with full functional correctness pre-synthesis (without the SDC). This approach streamlines development and supports mixed-mode simulation, which is particularly beneficial for unit testing and comprehensive design verification. By integrating RT models into a simulator testbench, this method allows for the automatic generation of timing constraints from RTCs, relieving designers from the intricate details of timing implementation. This process not only enhances efficiency but also offers a versatile and efficient tool for

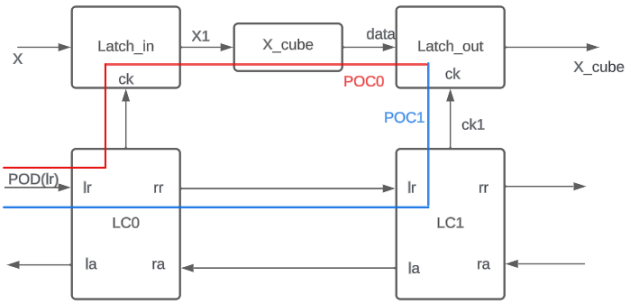

Figure 1: Pipelined RT implementation of x^3

mixed-mode simulations. Behavioral simulations are markedly quicker than structural simulations, facilitating a more efficient design exploration and behavioral validation process [10], [11], [12], [13]. Further, simulation times are prohibitive for large structural designs [14], [15].

To evaluate benefits and demonstrate feasibility of the model, the equations are applied to routers in a network-on-chip. Simulations are performed on an 8×8 array of 64 routers, and comparisons are made between a behavioral, structural, and mixed mode instance of the array.

II. Background and Related Work

Commercial and open source EDA tools are used to implement relative timed design, including synthesis, place and route (PnR), and verification flows. However, additional models, and at times custom tools, are required. (Sequential synthesis and automatic verification and generation of RT constraints use custom tools). Behavior and timing verification has been fully supported for some time for synthesized and PnR designs. This work creates models that enable behavioral simulation.

Numerous designs have been implemented using RT design flows showing advantages in power, performance, and area compared to clocked flows [16], [17]. However, these designs have only been able to use post synthesis or PnR designs for simulation. The Verilog models introduced in this work enable simulation and validation of behavioral relative timed designs for the first time.

RT equations have been proven complete to implement any timed circuit application [18]. This proof of generality of RT constraints requires the specification of multi-cycle instances. However, designs normally only require single cycle constraints on the pod and pocs of Eqn. 1. Verilog models for RT constraints presented in Sec. IV are only shown for single cycle relationships for clarity. However, these models can be extended to the arbitrary indexed transitions that are required by the proof of generality.

A pipelined design flow based on formal timing and logic verification was introduced in Sec. I. This flow partitions the design into data path logic that consists of combinational logic and registers, and a control path that consists of formally verified sequential controllers. The general RT design flow is not restricted to this approach. Arbitrary design constructions and user generated RT constraints are also supported by this flow.

The Verilog models in this paper can also be used for performance verification (PV) as well as functional validation of an RT design. Performance verification of a RT handshake design is more complex than that of a clocked finite state machine [19]. Each request acknowledge handshake pair implements a silicon oscillator, that serves as a "local clock". Some designs have millions of these silicon oscillators. Most designs have architectural feedback that span numerous oscillators. These dependencies create complex timing relationships [20]. If no margin (m from Eqn. 1) is specified in each RT constraint, a fast "zero delay" simulation can be implemented. To perform behavioral performance verification, a minimum requirement is to specify a margin m for each RTC that results in a handshake frequency that matches the performance targets for each oscillator associated with every pipeline stage. This is not shown in this paper for to help maintain clarity.

III. Approach

The Verilog for Figure 1 is shown in Figure 2. This Verilog will be used as an example for building the RT simulation models. This example assumes four cycle return-to-zero handshake protocols where data is valid on the rising edge of request. Latches are assumed to be pulse clocked.

```
module Xcube (lr, la, rr, ra, x, X_cube);
  input lr, ra; output la, rr;
  input [31:0] x; output [31:0] X_cube;
  latch Latch_in (.D(x), .clk(ck0), .q(xl));
  ctrl LC0 (.lr(lr), .la(la), .rr(rr0), .ra(la0), .ck(ck0));
  assign data = x**3;
  latch Latch_out (.D(data), .clk(ck1), .q(X_cube));
  ctrl LC1 (.lr(rr0), .la(la0), .rr(rr), .ra(ra), .ck(ck1));
endmodule // Xcube
```

Figure 2: Verilog for circuit in Figure 1.

In this example the output latch is pulse clocked after the process `X_cube` produces its result. For correct functionality, `data` must arrive at `Latch_out` before clock `ck1` stores the value, with a margin of s. This is expressed by the RT constraint of Eqn. 2. These delay paths are highlighted in red and blue in Figure 1. The path highlighted in red from pod to poc_0 is the maximum delay path and the path highlighted in blue from pod to poc_1 is the minimum delay path. The common causal event or point of divergence of this design is `LC0/lr`, and the two signals racing in time are `data` and `ck1`.

$$\text{LC0/lr}\uparrow \mapsto \text{data} + s \prec \text{ck1}\downarrow \tag{2}$$

Correct signal ordering is enforced in simulation using RT equations implemented with an event based model that detects the arrival of `ck1` relative to `data` for every causal transition on `LC0/lr`. The proposed model utilizes Verilog's `force`, `wait`, and `release` statements to ensure the sequential occurrence of poc_0 (`data`) before poc_1 (`ck1`) as shown in Figure 3. The `force` statement plays a crucial role by fixing the net at a specified logic level, maintaining this value irrespective of

any other logic-driven changes to the net until the `release` statement is executed.

```
always @(posedge LC0.lr) begin
    force ck1 = ck1;
    wait(data);
    release ck1;
end
```

Figure 3: Verilog behavioral RT constraint for Eqn. 2

The model in Figure 3 first checks for transitions on pod. In scenarios involving concurrent system operations, the model can freeze the value of poc_1 upon the occurrence of pod. The state of poc_1 is now maintained until the poc_0 event takes place. This model narrows the simulation's reachability, thereby ensuring the precedence of poc_0 over poc_1. The poc_0 event is then detected using the Verilog `wait` statement. At this point poc_1 is allowed to proceed based on the logic and timing of the design. This technique is instrumental in establishing the event order as dictated by a relative timing constraint within a behavioral design context.

This basic *force-wait-release* mechanism is the foundation of the behavioral simulation model for an RT constraint. This method offers a robust implementation of a mathematical RT constraint. In the context of Figure 1, when a rising transition on `LC0/lr` happens, transitions on `ck1` are delayed until `data` arrives at `Latch_out`.

Based on the implementation and behavior of a circuit and its sequential protocols, multiple RT constraints may be required to work in concert to correctly prune the behavioral reachability of a design. These include constraints to enforce minimum clock pulse width and transparency window control for the latches among others. The singular constraint of Eqn. 2 used in this section is not sufficient for Figure 1 to operate correctly in the time domain. For instance, Eqn 2 alone does not guarantee that `ck1` is low when `LC0/lr` rises. A verification flow is therefore recommended to ensure a necessary and sufficient set of constraints.

IV. Key Requirements

The purpose of an RT constraint is to prune the reachability of a circuit based on the ordering of events that is imposed based circuit timing. Multiple constraints often interact to produce the correctly pruned reachability graph (such as both setup and hold constraints). This produces a number of important but subtle requirements when pruning reachability in Verilog with `force` and `wait` statements. Likewise, the amount of concurrency in a design can add complexity to design. This includes complexity in terms of bit timing of a wide function as well as the amount of concurrency between individual signals in a sequential design.

To illustrate these relationships a hypothetical constraint set is introduced in Figure 4 to outline the key requirements and challenges that must be considered in the development of the Verilog RTC models. The hypothetical constraints `rtc0`, `rtc1` and `rtc2`, `rtc3` have a common poc_1 (`y_+` and `lr−` respectively). Constraint `rtc3` has a datapath with a 32-bit endpoint (`din`$[31:0]$).

```
rtc0: lr+ ↦ rr+ ≺ y_+
rtc1: lr+ ↦ la+ ≺ y_+
rtc2: la+ ↦ y_+ ≺ lr-
rtc3: lr+ ↦ din[31:0] ≺ lr-
```

Figure 4: Example Constraint Set

Each RT constraint is mapped to a Verilog code segment such as shown in Figure 3. These are included in the Verilog testbench. In the interest of brevity and clarity, the Verilog code snippets related to the following requirements represent segments of the full mapping codebase. Portions of the code which include various necessary checks and flags have been replaced with comments or omitted entirely to maintain readability and focus on the critical components.

Requirement 1. *Causality:* The pod is assumed to be causal to both poc_0 and poc_1 in rt constraint $\mathsf{pod} \mapsto \mathsf{poc}_0 + m \prec \mathsf{poc}_1$. For example, in RTC a+ ↦ b- ≺ c+, a rising edge on a will cause a falling transition on b and a rising transition on c.

Requirement 2. *Trigger Condition:* A poc signal is *triggered* when the logic level of the poc is in the opposite boolean state than the condition specified by the RT constraint (e.g. a logic low value for poc+ and logic high for poc-). If the poc is not triggered when the pod event occurs, a transition on pod must occur to place it in a triggered condition.

This assumption implies that the causal path from pod to poc is not multicycle, which can be extended in future work.

Requirement 3. *Breaking combinational cycles.* Verilog simulators are not required to preserve signal causality when a design contains combinational cycles. Since causality is specified in Requirement 1, combinational cycles must be broken into different simulation steps.

This is performed by adding delay to the feedback signal in a combinational cycle specified behaviorally. For example, assume a C-element, which is a state-holding circuit that can be implemented with combinational feedback. The following snippet adds one unit of delay that breaks the combinational cycle into different simulation steps.

```
assign #1 c = (a & b) | (a & c) | (b & c);
```

Requirement 4. *Reachability Pruning:* Concurrency is reduced in the design by ensuring the poc events are sequential with Verilog `force` and `release` statements. Signal poc_1 is prevented from occurring until the event on poc_0 has occurred. This forms the basis for the Verilog statements that enforces a timing constraint in the system.

This requirement can be implemented as follows. This form of reachability pruning requires that poc_1 is triggered. Timing endpoint poc_1 is forced to remain in the triggered state until poc_0 fires, at which point it is released and can follow the logical and standard timing behavior of the circuit.

```
always @(pod) begin
```

```
    force poc1 = poc1; // force to current value
    wait(data);
    release ck1;
end
```

Requirement 5. poc *sharing:* The semantics of the Verilog `release` statement will immediately return the specified net to the control of the logic behavior of the circuit regardless to the number of times it has been forced. Thus any set of RT constraints that share the same poc_1 net must work in concert to ensure the net is properly released. Any RTC with a shared poc_1 may not release the poc_1 signal until all of restraining poc_0 signals have fired in active RTCs where the pod event has occurred.

The constraint set example shown in Figure 4 has shared poc_1 signals `y_+` in constraints `rtc0` and `rct1`. Since they also share the same pod signal `lr+`, signal `y_+` may not assert until after both `rr` and `la` rise. A counter is added for each poc_1 that exists in multiple RTCs. It is incremented each time a pod event occurs in one of these RTCs, and decremented when the poc_0 event fires. If the counter reaches zero then the `force` statement can be released. A code snippet for the model for `rtc0` is shown. A similar model will be implemented for `rtc1`.

```
 int poc1_y_UP = 0;
always @(posedge lr) begin
    poc1_y_UP = poc1_y_UP + 1;
    force y_ = y_; // force to current value
    wait(rr);
    poc1_y_UP = poc1_y_UP - 1;
    if (poc1_y_UP == 0) release poc1;
end
```

Requirement 6. *Continuous Assignments:* The semantics of the Verilog `force` statement are limited in scope as it operates only on continuous assignments. This requires that `reg` variables must be assigned to a continuous statement to have their value forced.

One primary goal of this work is to allow the source Verilog to be synthesized as well as simulated. To implement this requirement the source code must be modified. Fortunately this can be done in such a way that still allows the code to be synthesized. Assume that net `y_` from `rtc0` is declared a `reg`. A new wire is created and assigned the `reg` value. This wire can be forced and released. With this change the poc_1 signal `y_` is assigned to `y_RTC_CA`. Net `y_RTC_CA` must replace all instances of net `y_` in the source code, and is the signal that is forced and released to prune reachability of the design.

```
reg y_; wire y_RTC_CA;
assign y_RTC_CA = y_;
```

Requirement 7. *Multibit timing endpoints:* Data validity of a multi-bit signals such as a data bus must be identified with a single boolean value.

Identifying data validity in a behavioral simulation using zero-delay functions is usually readily performed. For example, data will always be stable per Requirement 3 when handshake signals are asserted in a zero delay simulation of a bundled data design style when handshake controllers with combinational loops are employed. However, the bundled data timing requirement including setup and hold times must be correctly modeled in structural and mixed mode designs. This issue can be addressed by modern hardware description languages that implement a framework that determines data validity [21].

The constraint `rtc3` shown in Figure 4 has a multibit data bus at the first point of convergence. A boolean flag must be generated indicating data is valid, shown as `dinvalidp` in the code snippet below. The specific approach for determining when to assert `dinvalidp` depends on the implementation.

```
reg dinvalidp = 0;
...
assign dinvalidp = 1; // when din is stable
...
always @(posedge lr) begin
    force lr = lr;
    wait(dinvalidp);
    dinvalidp = 0;
    release lr;
end
```

Requirement 8. *Margins of separation:* Minimum margins of separation between poc0 and poc1 are implemented by adding a delay between the `wait` and `release` statements.

Minimum separation between events is important for pulse shaping, to ensure correct setup and hold times in a design, and for using behavioral simulations for performance verification. The RT constraint $\mathsf{pod} \mapsto \mathsf{poc}_0 + m \prec \mathsf{poc}_1$ indicates that poc_1 is triggered after poc_0 plus a margin m. The example below shows how this is modeled. Performance verification of a constraint is implemented when m is set to match the desired phase delay of silicon oscillator.

```
always @(pod) begin
    force poc1 = 0;
    wait(poc0);
    #m;           //introduce margin or delay
    release poc1;
end
```

Requirement 9. *Mixed mode simulation:* Delay tracing of relative timed paths is required to ensure RT constraints hold in a mixed mode relative timed design.

This work enables simulations of mixed designs which are part structural and part behavioral. The behavioral design can be either "zero delay" or implement delays for performance verification.

Structural design that have been synthesized with RT constraints will simulate without requiring any models developed in this work. However, handshake designs have signal paths at the interfaces that are outside of the synthesized logic. For example, a pipeline controller at the interface of a synthesis block will have request acknowledgment handshakes with adjacent modules [22]. Relative timing constraints where the timing path goes outside of the synthesized module can be violated, particularly when interfacing with zero delay behavioral models.

The RT constraint path in Figure 5 illustrates this point.

The red max delay path is internal to the circuit, whereas the blue min delay path is outside the circuit. In a zero delay behavioral model, the blue min delay path can be less than the red max delay path, violating an RT constraint. Thus all RT constraints which have paths that pass outside of a synthesized module must be properly modeled with delays appropriate for performance verification and verified for correct RT behavior.

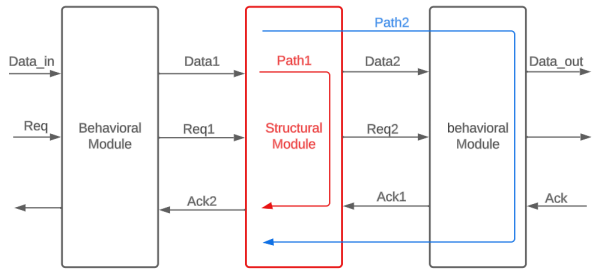

Figure 5: Internal and external paths in a mixed mode design

Requirement 10. *Pulse Logic:* Pulses are generated when the same net is specified for poc_0 and poc_1. To model minimum pulse width the margin of separation m must be specified in the Verilog model.

Pulse based logic is particularly challenging for traditional design flows, but is easily modeled with relative timing. Consider the pulse generator example as shown in Figure 6 which generates a high pulse on the rising transition of `in`. The RT constraint of Eqn. 3 can be used to define the correct timed behavior of this pulse.

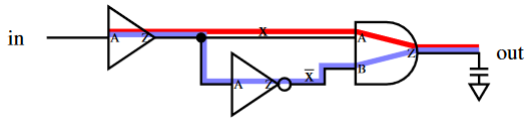

Figure 6: Pulse generator circuit.

$$\text{in+} \mapsto \text{out+} + m \prec \text{out-} \tag{3}$$

This can be modeled as follows.

```
always @(posedge in) begin
    wait(out);
    force out = out;
    #m;
    release(out);
end
```

V. Results

Simulation models for the rules presented in Sec. IV are developed for a network-on-chip (NoC) [23]. The NoC's architecture comprises 64 routers arranged in an 8×8 grid, configured in a wrapped torus topology. It utilizes dimension-ordered source routing with single-flit packets. The sequential controllers implement the communication protocols and control latch clocking similar to the `LC` blocks in Figure 1.

As a considerably extensive design, the NoC incorporates hundreds of timing constraints across its 64 submodules, ensuring precise operation. A complete set of RTCs are generated for the sequential controllers with formal verification. These constraints are largely mapped to behavioral simulation constraints by hand in order to validate this approach. The Verilog models generated for the different RT constraint sets are added to simulation testbenches.

Simulations are performed in QuestaSim using SPICE modeled NoC link delays and sdf back annotated timing for the synthesized designs. The base testbench employs 303 lines of Verilog code. The testbench code interfaces with a C++ NoC simulator that generates various work loads and delivery patterns that are offered to the NoC. Each simulation run verifies correct message delivery, records latency statistics, and measures activity factors for power evaluation.

The comparison between four designs is shown in Table I. All of the four designs passed the testbench with no errors. The second column lists the size of the testbench. All the additional lines of testbench code beyond 303 are comprised of the RT Verilog models developed from the requirements enumerated in Sec. IV. The third column lists the size of the NoC under test. The fourth column lists the maximum memory used during simulation, and the last column lists the runtime of the testbench in seconds.

The structural design was synthesized and timing closed in a 65nm technology node using the RT constraints to enforce the signal ordering specified in the RTCs. The behavioral and behavioral min designs implement the NoC with behavioral Verilog. In the mixed design, one of the routers is structural, and the other 63 are behavioral.

Zero delay behavioral simulation coupled with Requirement 3 (breaking combinational cycles) will prune many of the error states from a Verilog simulation of a handshake design. Therefore the behavioral min design used a subset of the RT constraints for simulation. The behavioral and mixed designs implement the full set of RT constraints. The size of the RT Verilog models for the NoC comprises 19,126 lines of code, or an average of approximately 300 lines of Verilog code per router. The reduced model reduces the size of the Verilog RT models by 21%, and contains an average of approximately 235 lines of Verilog code per router.

The synthesized structural design contains over 11× more lines of Verilog code than the behavioral or mixed mode versions, as synthesis has mapped the design to standard cell library gates. No behavioral timing constraints are required for the behavioral simulation of the structural design.

The process memory for the structural design simulation was approximately 12× larger than the behavioral design and 9× bigger than the mixed design. The simulation time for the structural design is over 50× longer than the behavioral designs.

The mixed mode design allows one to focus on a particular design block, given its full synthesized or PnR delays, while the rest of the design is behavioral. This is advantageous when performing unit testing. Mixed mode simulation can hence

Table I: NoC Results

Design	Lines of code(Testbench)	Lines of code(Design)	Simulation process memory (KB)	Simulation time (s)
Structural Design	303	95,207	3,100,000	1800
Behavioral Design	19,429	8,563	239,788	34
Mixed Design	19,361	19,469	325,952	180
Behavioral Min	15,332	8,563	230,487	32

help in reducing the time for the overall design debugging and provide a way for easier integration in the RT design space. The proposed work also paves the way for design modularity in asynchronous circuit design. In this mode, the runtime memory is reduced by 89% and simulation time by 90% over the fully structural design.

VI. Conclusion

A methodology is presented for mapping relative timing constraints to Verilog models. This enables the ability to behaviorally simulate designs with arbitrary timing expressed as relative timing constraints. The Verilog models are explicitly designed to be added to a testbench. Only small modifications to the behavioral code are required to implement the RT simulation models. The modifications do not prevent the behavioral code from being directly used for synthesis and place and route. The mapping approach is rule based and can be automated through CAD development. This approach enables rapid design and architectural exploration and evaluation of circuit designs that employ relative timing constraints.

To prove feasibility and correctness, four testbenches with simulation models were developed and tested on three versions of a 64 node toroidal network on chip design. All designs passed the regression tests. This experiment showed the advantages in memory and run time of a behavioral design over a synthesized structural design, with a 92% and 98% reduction in simulation memory and runtime respectively. This demonstrated significant advantages and feasibility of this approach at simulation time. A mixed mode design with one structural block and 63 behavioral blocks showed a 89% and 90% reduction in simulation memory and runtime, while providing detailed gate-level delay data of the structural router.

References

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

[2] J. Cortadella, M. Kishinevsky, A. Kondratyev, L. Lavagno, and A. 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.

[3] K. Y. Yun and D. L. Dill, "Automatic Synthesis of Extended Burst-Mode Circuits: Part I (Specification and Hazard-Free Implementation)," *IEEE Transactions on Computer-Aided Design*, vol. 18, no. 2, pp. 101–117, Feb 1999.

[4] Y. Xu and K. S. Stevens, "Automatic Synthesis of Computation Interference Constraints for Relative Timing," in *26th International Conference on Computer Design*. IEEE, Oct. 2009, pp. 16–22.

[5] M. Bozga, H. Jianmin, O. Maler, and S. Yovine, "Verification of Asynchronous Circuits using Timed Automata," *Electronic Notes in Theoretical Computer Science*, vol. 65, no. 6, pp. 47–59, 2002.

[6] K. Desai, K. S. Stevens, and J. O'Leary, "Symbolic Verification of Timed Asynchronous Hardware Protocols," in *Annual Symposium on VLSI (ISVLSI)*. IEEE Computer Society, Aug 2013, pp. 147–152.

[7] K. S. Stevens, Y. Xu, and V. Vij, "Characterization of Asynchronous Templates for Integration into Clocked CAD Flows," in *15th International Symposium on Asynchronous Circuits and Systems*. IEEE, May 2009, pp. 151–161.

[8] E. Quist, P. Beerel, and K. S. Stevens, "Enhanced SDC Support for Relative Timing Designs," in *Digital Automation Conference*. IEEE/ACM, July 2009, user Track Poster.

[9] G. Gimenez, A. Cherkaoui, G. Cogniard, and L. Fesquet, "Static Timing Analysis of Asynchronous Bundled-Data Circuits," in *24th International Symposium on Asynchronous Circuits and Systems*. IEEE, May 2018, pp. 110–118.

[10] M. Renaudin and A. Fonkoua, "Tiempo Asynchronous Circuits System Verilog Modeling Language," in *18th International Symposium on Asynchronous Circuits and Systems*. IEEE, Oct 2009, pp. 105–112.

[11] A. Saifhashemi and P. A. Beerel, "High Level Modeling of Channel-Based Asynchronous Circuits Using Verilog," in *Communicating Process Architectures*, J. Broenink, H. Roebbers, J. Sunter, P. Welch, and D. Woods, Eds. IOS Press, 2005, pp. 275–287.

[12] E. Esimai and M. Roncken, "Flexible Compilation and Refinement of Asynchronous Circuits," in *28th International Symposium on Asynchronous Circuits and Systems*. IEEE, July 2023, pp. 109–119.

[13] A. Saifhashemi and P. A. Beerel, "SystemVerilogCSP: Modeling Digital Asynchronous Circuits Using SystemVerilog Interfaces," *Communicating Processor Architectures*, vol. 68, pp. 287–302, 2011.

[14] Y. Zhang, H. Cheng, D. Chen, H. Fu, S. Agarwal, M. Lin, and P. A. Beerel, "Challenges in Building An Open-source Flow from RTL to Bundled-Data Design," in *24th International Symposium on Asynchronous Circuits and Systems*. IEEE, May 2018, pp. 26–27.

[15] C. Chau, W. A. Hunt Jr., M. Roncken, and I. Sutherland, "A Framework for Asynchronous Circuit Modeling and Verification in ACL2," in *Hardware and Software: Verification and Testing*, ser. Lecture Notes in Computer Science, vol. 10629, Nov 2017, pp. 3–18.

[16] K. S. Stevens, S. Rotem, R. Ginosar, P. Beerel, C. J. Myers, K. Y. Yun, R. Kol, C. Dike, and M. Roncken, "An Asynchronous Instruction Length Decoder," *IEEE Journal of Solid State Circuits*, vol. 36, no. 2, pp. 217–228, Feb. 2001.

[17] Y. Chen, X. Zhang, Y. Lian, R. Manohar, and Y. Tsividis, "A Continuous-Time Digital IIR Filter With Signal-Derived Timing and Fully Agile Power Consumption," *IEEE Journal of Solid State Circuits*, vol. 53, no. 2, pp. 418–430, Feb 2018.

[18] R. Manohar and Y. Moses, "Timed Signalling Processes," in *28th International Symposium on Asynchronous Circuits and Systems*. IEEE, July 2023, pp. 10–19.

[19] C. E. Molnar, I. W. Jones, W. S. Coates, J. K. Lexau, S. M. Fairbanks, and I. E. Sutherland, "Two FIFO Ring Performance Experiments," *Proceedings of the IEEE*, vol. 87, no. 2, pp. 297–307, February 1999.

[20] G. Gill and M. Singh, "Bottleneck Analysis and Alleviation in Pipelined Systems: A Fast Hierarchical Approach," in *15th International Symposium on Asynchronous Circuits and Systems*. IEEE, May 2009, pp. 195–205.

[21] T. Bourgeat, C. Pit-Claudel, A. Chlipala, and Arvind, "The Essence of Bluespec: A Core Language for Rule-Based Hardware Design," in *41st International Conference on Programming Language Design and Implementation*. ACM, June 2020, pp. 243–257.

[22] C.-F. Law, B.-H. Gwee, and J. S. Chang, "Modeling and Synthesis of Asynchronous Pipelines," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 19, no. 4, pp. 682–695, April 2011.

[23] V. Nori, B. Chauviere, M. J. Wibbels, and K. S. Stevens, "A Novel Asynchronous Network-On-Chip Based on Source Asynchronous Signaling," in *28th International Symposium on Asynchronous Circuits and Systems*. IEEE, July 2023, pp. 71–77.