Automatic Timing Closure for Relative Timed Designs

Tannu Sharma and Kenneth S. Stevens
Email: kstevens@ece.utah.edu
Electrical and Computer Engineering, University of Utah

***Abstract*—Relative timing is a universal representation of the sequencing property of time. Relative timing can be applied to specify the correctness and performance properties of digital circuits in the form of a set of timing constraints. These constraints are passed to commercial synthesis, place and route, and timing verification tools to optimize the robustness, power, and performance of digital integrated circuits. This paper presents an algorithm for the automatic convergence of a set of relative timing constraints in the synthesis and place and route flows to create a timing closed design. The algorithm performs timing closure for a design by starting with zero maximum delay targets. This tool produces improved results compared to a start point with delay values close to final closed values.**

I. Introduction

Time delays are manifested in the components and wires of an integrated circuit (IC). These delays are evaluated based on a timing path between two points in a circuit, which consists of a sequence of components a signal must pass through. They dictate the robustness, performance, and power of a system. Static timing analysis is employed to evaluate and optimize delays and to close timing during synthesis and layout [1].

The traditional techniques employed by commercial EDA tools are insufficient to close timing on a relative timed (RT) design. RT timing paths can be cyclical, and may be controlled by state bits in sequential logic implemented as combinational gates with feedback. To improve quality of relative timed designs, an engine that understands relative timing is required to obtain delay target value and sign-off timing in the current commercial framework.

A. Relative Timing

Relative timing (RT) is a universal representation of the sequencing property of time [2]. Sequencing in the time domain is a common correctness requirement used in integrated circuits. An example of a critical well known correctness requirement in the time domain is the storing of data in a flip-flop. The data must arrive at a flop earlier than the clock.

The simplified universal specification of a relative timing constraint is shown in Eqn. 1 [2]. There must be a common timing start point called a point of divergence (pod) that has causal paths to two timing endpoints called points of convergence (poc_0, poc_1). To ensure the delay on the early path is always less than the delay of the late path requires taking the maximum delay from pod to poc_0 (plus margin m) and the minimum delay from pod to poc_1. RT expressions employ unbounded delays, and thus are a property of the circuit structure. Therefore, RT constraints are agnostic to specific implementation details of a design that affect circuit delays such as technology node, device sizes, or standard cell layout. Specific path delays are not known until design instantiation and timing closure.

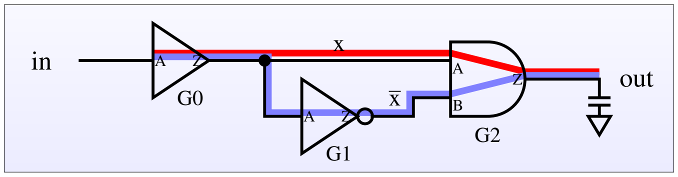

Fig. 1: Pulse generator circuit. The maximum delay path from the rising edge of `in` (pod) to the rising edge of `out` (poc_0) of Eqn. 2 is highlighted in red, and the minimum delay path from the rising edge of `in` (pod) to the falling edge of `out` (poc_1) is highlighted in blue. The capacitor models wire and gate capacitance of a latch array.

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

Assume one needs to synthesize and verify a circuit that generates a pulse. Such a circuit could be used to pulse clock a latch bank. The circuit in Fig. 1 generates a pulse on the `out` net upon a rising edge of the `in` net when proper circuit delays are employed. The delay path through the inverter can be designed to generate the required minimum pulse width. The relative timing constraint (RTC) to correctly realize the pulse generator of Fig. 1 is shown in Eqn. 2. This produces a pulse on net `out` with a minimum width m. The '+' or '-' appended to a net name indicates a rising or falling transition respectively on the net. The causal path through the circuit to create a rising edge on `out` is through the `A` pin of gate `G2`; the causal path for the falling edge is through pin `B`.

$$\text{in+} \mapsto \text{out+} + m \prec \text{out-} \tag{2}$$

A relative timed design can contain as many as several million RT constraints. Many of these paths conflict as maximum and minimum delay path segments can partially or completely overlap [3]. Manually converging timing by modifying the timing constraints of individual paths is not a feasible option for such designs, and it may not be possible to resolve the large number of violating paths through mere post-layout ECO [4]. As such, an automated aid to produce timing closed designs is required.

The timing on a relative timed design is complex, where the optimization of one path may affect timing of other paths (associated or non-associated). The entire problem is an interaction of non-convex optimization algorithms across often competing timing path constraints. To model the delay target value for each path, the algorithms consider device sizes, drive strength, transition capacitance, fanout, derating factors, EDA uncertainty along with the affects of other paths on a path.

The key contribution of this paper describes a heuristic based timing closure (HBTC) CAD tool that automatically generates functional delay targets for a complete RT constraint set. The algorithm updates relative timing targets based on the results of synthesis or place and route EDA tools to produce a design with high quality of results in terms of circuit power, performance, and area (PPA). The goal of the HBTC algorithm it to produce high quality PPA in as few a number of EDA tool iterations as possible because EDA tools such as Design Compiler (DC) and Integrated Circuit Compiler (ICC) are in the inner loop. The iteration terminates when there is no negative slack on any relative timing constraint.

Iterations start with a set of RTCs where delay targets have been assigned to timing constraints (such as y in Eqn. 3). The HBTC engine presented here is capable of generating a timing closed design with arbitrary initial delay targets, including zero delay (0 ps) initial maximum delay targets (e.g. $y = 0$ is the starting condition). The tool is usually able to produce a completely closed set of constraints with no negative slack violations.

The heuristic based timing closure tool presented here is evaluated on a few designs for run time, power, performance, and design robustness at different technology nodes. Various types of RT constraints are employed in these designs. Pipelined designs employ the bundle data design style with handshake controllers [5]. In general fully automated timing closure often shows better results than designs which have been manually brought to near convergence.

II. Background and Related Work

A. Specifying RT Constraints

Relative timing constraints can be mapped to Synopsys design constraints (sdc) to drive timing verification and timing driven synthesis and place and route [6], [7]. An example of a relative timing constraint expressed in sdc format that specifies timing of the pulse circuit of Fig. 1 with the RTC of Eqn. 2 is shown in Eqn. 3 and 4. The early arriving path – with a maximum constrained delay of y – is highlighted in red in Fig. 1. The late arriving path – with a minimum delay target of z – is highlighted in blue. RT constraints assume causal paths, but static timing only employs net connectivity. Therefore, when mapping the early and late RTC paths to sdc format the causal paths through this circuit must explicitly specified with the -through commands.

```
set_max_delay y -rise_from in
    -rise_through G2/A -rise_to out          (3)
set_min_delay z -rise_from in
    -fall_through G2/B -fall_to out          (4)
```

Specific delay values must be assigned to RTC paths for synthesis and evaluation with tools such as DC, ICC, and PrimeTime. These are the y and z values in Eqn. 3 and 4. Given a specific timing delay target for y and z, the synthesis and place and route (PnR) tools will optimize the design, attempting to create path delays that meet the path constraints.

Assume $y = 0.100$ ns and $z = 0.200$ ns in Eqn. 3 and 4. If the synthesis and PnR tools are able to create a design without negative slack for these sdc constraints, then the margin m, and thus the pulse width, will be at least 100 ps.

We assume that an sdc representation has been created for the design for which timing closure is to be applied. The relative timed design flow consists of an independent but related flow to characterize a set of sequential or timed design macros and also to implement these macros in a system design. The characterized macros contain sdc representations of relative timing constraints to direct the timing driven optimization of commercial EDA tools. These constraints are used by HBTC that is integrated with the commercial EDA tools to verify and/or update the target values.

B. Relative Timed Circuits

Relative timed circuits are an interesting class of circuits because of their ability to leverage the advantages of both clocked and asynchronous design styles. They can also provide a significant improvement over power, performance, and area than a clocked design.

An implementation of a C-Element with combinational feedback is shown in Fig. 2. The output `c` asserts high when both inputs are high, and goes low when both inputs are low [5]. The `[c (ac)']` cycle remembers that `a` has risen, and the `[c (bc)']` cycle remembers that `b` has risen. The set of RT constraints that make the design conform to its specification is given in Eqn. 5–8. Eqn. 5 and 7 ensure that the circuit remembers that `a+` occurred by asserting the `[c+ (ac)'-]` cycle before the environment observes `c+` and lowers `a-` (path `[c+ a-]`), and before `b-` unasserts cycle `[c+ (bc)'-]` through timing path `[c+ b- (bc)'+]`. The second symmetrical pair of RTCs enforces the same constraints on cycle `[c (bc)']`.

$$\text{c+} \mapsto \text{(ac)'-} + m \prec \text{a-} \quad (5) \qquad \text{c+} \mapsto \text{(ac)'-} + m \prec \text{(bc)'+} \quad (7)$$

$$\text{c+} \mapsto \text{(bc)'-} + m \prec \text{b-} \quad (6) \qquad \text{c+} \mapsto \text{(bc)'-} + m \prec \text{(ac)'+} \quad (8)$$

In order to employ commercial EDA tools, combinational loops in the C-element circuit are cut using sdc commands `set_disable_timing` to create a DAG. This DAG must preserve RT paths specified in the sdc constraint file.

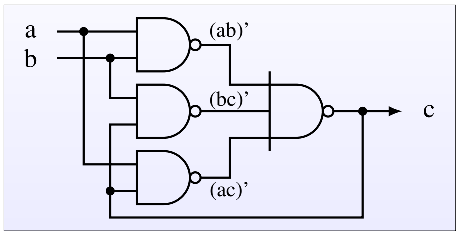

Fig. 2: C-Element NAND Implementation.

C. Related Work

Much work has been invested to develop CAD to support asynchronous design as a viable design choice [8], [9], [10], [11], [12], [13]. Also, there has been work to advance timing flows and apply clocked CAD to timed asynchronous methodologies [14], [15], [16], [17], [18], [19]. The absolute timing analysis tool for asynchronous designs like ATACS does well on small designs. Its extension was limited for complex designs. The ACDC synthesis flow for RT designs works with commercial EDA framework, however, the flow is also suitable for small subsystems [17]. This flow doesn't ensure convergence or optimization for better PPA. To overcome the gaps of conventional modeling, the timing loops are systematically removed without disabling timing arcs. The flow works well for WCHB QDI asynchronous circuits within traditional EDA flow [19]. An iterative approach to converge timing on a RT design by gradually increasing delay on failing paths has proven effective to sign-off timing on complex designs [18]. However, it also suggests incremental ECO runs to overcome non-convergence on designs with large number of failing RT constraints.

The iterative approach presented in this paper is applicable for both small and large designs with millions of RT constraints. It employs cycle cutting to create a DAG to perform timing analysis using conventional EDA. To converge timing on violating paths, the delay targets are increased. Min-delay path targets in an RTC are also increased when necessary to ensure that they remain greater than the max-delay path plus the assigned margin when the associated max-path delay has been increased. In addition, optimizations exist to reduce power consumption by relaxing timing to reduce the drive strength on some paths.

III. Language

This work leverages commercial tools and languages whenever available. However there are some properties of a RT constraint which are not supported by modern circuit design specification languages such as sdc. Thus we have created a simple language to represent relative timing constraint relationships that can be embedded into an sdc constraint set.

While we can specify and constrain independent paths in a circuit as done in Eqn. 3 and 4, there are no semantics in the sdc specification language to express relationships between constraints. Thus we have added a `#margin` pragma to the language. This identifies the two timing paths in an RTC, as well as their margin of separation. The semantics are:

```
#margin  << margin >>  << maxpath >>,  << minpath >>;    (9)
```

An example of a complete RTC constraint for the pulse circuit of Fig. 1 expressed in our extended sdc notation is shown in Eqn. 10.

```
set_max_delay y -from in -through NAND/A -to out
set_min_delay z -from in -through NAND/B -to out
#margin m -from in -through NAND/A -to out , \        (10)
          -from in -through NAND/B -to out ;
```

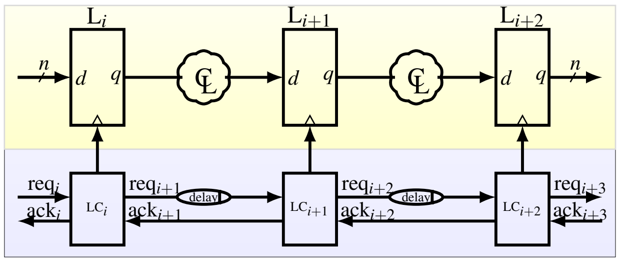

Fig. 3: Timed bundled data handshake design.

Note that EDA tools optimize against specific delay values, rather than relative values. When mapping to a specific technology node or design implementation, the path delays y and z in Eqn. 10 must be replaced with specific values. For the circuit to be properly specified, $y+m<z$. Timing closure consists of adjusting the values of y and z such that each path has no negative slack, and $y+m<z$. In the current implementation of HBTC, the margin m is specified by the designer and remains constant.

When constraints such as those expressed in Eqn. 10 are passed to a commercial tool, a specific delay value is required. Specific timing values are therefore provided for each sdc constraint for timing closure. HBTC will iterate across the design until the maximum delay plus a margin in an RTC constraint is less than the minimum delay. This is achieved by modifying the delay variables passed to each tool until the synthesis or place and route tool are able to create a design that meets all of the margins. Because of this, nearly any initial maximum delay value is okay, including zero delay targets.

Many designs have the best PPA quality of results when multiple delay targets are equivalent or optimized in concert with each other. For example, a linear pipeline may be best optimized when each pipeline stage has the same cycle time targets. Identical delay targets can be achieved by using a tcl variable to identify a delay or wildcards to specify timing path end points. For example, delay z in Eqn. 10 can be expressed as a tcl variable, such as `$pd0`, rather than a specific scalar value. All RT constraints using a similar delay target are updated concurrently. The constraint in Eqn. 11 is an example that employs both a tcl variable and wild cards to create uniform delays between pipeline controllers.

```
set pd0 0.200                                              (11)
set_min_delay $pd0 -from pipe/*/rr -to pipe/*/lr
```

RT constraints are associated to each module master in an object oriented fashion. For example, each linear controller (LC) in the design of Fig. 3 will contain a set of RT constraints that are used to direct EDA tools in performing timing driven synthesis and place and route. However, some timing endpoints may lie outside of the LC controller itself – such as as the `d` and `clk` ports of a flip-flop. We have adopted a language extension to express endpoints outside of the module that contains the relative timed point of divergence (pod) [20]. Endpoints outside of the current module are referenced relative to adjacent pipeline stages upstream or downstream from the standard direction of the data flow.

IV. Algorithm

The EDA timing synthesis and PnR optimization algorithms do not result in a convex search space when provided a set of relative timing constraints. A change in any RT constraint in the system can trigger a change in the delay of associated or non-associated RT constraint path, be they min or max delays.

A number of different algorithmic approaches have been tried to converge timing closure on a set of RT constraints. Gradient descent approaches were explored during the implementation stage. The Newton Raphson method and Brent algorithm were implemented to calculate delay target values for each path [21]. The cost function evaluates delay target value for each path while optimizing the performance of the design. Newton Raphson results are too dependent on the initial value selected, whereas with Brent algorithm the range has to be known. Defining the values between zero and a large number wasn't sufficient to obtain a value for which that design would achieve timing convergence.

We were not able to converge timing on a relative timed circuits using these convex optimization techniques. We collaborated with mathematicians to find an algorithm to model this problem. However, none was obtained before we expanded to implement the greedy heuristic algorithm described here.

A. Heuristic Based Timing Closure

A greedy search algorithm, presented as Algorithm 1, has been employed to quickly converge timing on a design with good PPA. EDA tools must support the synthesis and optimization of minimum delay constraints because the delay of the min-delay path must be larger than the delay of the max-delay path in an RT constraint.

The heuristic algorithm is based on two assumptions. (a) The delay target of a maximum delay path can be increased until the tools can implement a design where the path meets timing. (b) We can maintain fidelity of a relative timing constraint by adding delay to the late path (which is the minimum delay path) in order to ensure that the margin of the relative timing constraint continues to hold.

The heuristic based search engine modifies a subset of the design's timing constraints by monotonically increasing max and min delay constraints such that the new constraint set has a higher probability of reaching a timing converged design. This iteration continues until the circuit meets timing or no progress is made. It is possible to create designs which have no consistent solution, such as designs where a min-delay path with a large delay is entirely covered by a max-delay path with a small delay [3]. In such circumstances, there is no solution.

Based on this approach, one could very quickly create a timing closed design by directly applying path delay results of synthesis or place and route to each delay target. However, the PPA of the design will be poor. Instead, the heuristic algorithm slowly "cools" the design until no outliers remain. HBTC is also capable of performing a second iteration of the algorithm that evaluates and relaxes timing on paths with large device sizes as a power saving optimization.

The commercial EDA tools, in general, do not directly support minimum delay constraints as a part of their central timing driven optimizations; the tools generally provide "min-delay fixing". Path delays are padded to fix min delay violations after the design has been optimized for maximum delays. Likewise, this HBTC tool does not repair minimum delay violations until after the maximum delay violations have all been met. At this point both minimum and maximum timing violations will be concurrently repaired. However, when a max-delay target is increased the associated min-delay path in the RTC will be updated to ensure that the RTC margin holds.

This algorithmic approach provides RT designs that better meets performance targets given the current EDA optimization algorithms. However, designer intervention may still be required to fully close a design as timing and routeability are significantly influenced by area and design density, which is not adjusted as part of this algorithm.

Algorithm 1 Heuristics Based Algorithm for each path.

```
1:  $checkMin \leftarrow false, failures = \{0,0,0\}, W_p \leftarrow 0.8, D_m \leftarrow 100ps$
2:  while 1 do
3:      $failures[2] \leftarrow failures[1], failures[1] \leftarrow failures[0], failures[0] \leftarrow 0$
4:      Run Synthesis or PnR
5:      begin forall $n \in$ max $\cup$ min timing constraints        ▷ Check all paths
6:      $s_n \leftarrow$ slack($n$)
7:      $t_n \leftarrow$ getTargetDelay($n$)        ▷ Get path slack
8:      if (isMaxPath($n$) $\vee$ $checkMin$) $\wedge$ $s_n < 0$ then
9:        ▷ Update target delay on max paths or all paths with negative slack
10:         $t_n \leftarrow t_n +$ min($(0 - s_n) \times W_p, D_m$)        ▷ Increase path target delay
11:         setDelayTarget($n$, $t_n$)
12:         $failures[0] \leftarrow failures[0] + 1$        ▷ Increment failed path count
13:         $t_m \leftarrow$ getTargetDelay(minPath($n$))
14:         if (isMaxPath($n$) $\wedge$ $t_m - t_n <$ margin($n$)) then
15:                 ▷ RTC margin doesn't hold, fix to retain RTC margin
16:             $t_m \leftarrow t_n +$ margin($n$)
17:             setDelayTarget(minPath($n$), $t_m$)        ▷ increase min path target
18:         end if
19:     end if
20:     end forall
21:     if $failures[0] = 0$ $\wedge$ $checkMin = false$ then
22:                 ▷ No max path violations, now check min paths too
23:         $checkMin \leftarrow true$
24:         goto 5
25:     end if
26:     if $failures[0] = 0$ $\vee$ ($failures[0] = failures[1] = failures[2]$) then
27:         Break        ▷ Exit on no negative slack or non-convergence
28:     end if
29: end while
```

B. Implementation

Algorithm 1 summarizes HBTC where n is the index number of a timing path, $failures$ is an integer array identifying the number of paths with negative slack in the last three runs, $checkMin$ controls whether minimum delay paths are evaluated, and W_p and D_m are user defined values used to control convergence time and quality of results of the algorithm. Their values vary based on design type and technology node. Function isMaxPath(n) returns $true$ if path number n is a maximum delay path, delay(n) returns the maximum delay for early RTC path n and minimum delay if n is a late path, minPath(n) returns the path index number of the minimum delay path associated with maximum delay path n in a relative timing constraint, and margin(n) returns the margin between the two paths of a RTC for which n is one of the two paths. Slack(n) returns the slack of path n, and the functions getTargetDelay(n) and setDelayTarget(n, t) retrieves the delay target for the RTC associated with delay path n, or sets it to value t.

Each path n has delay target t_n which is initially assigned by the user (e.g. y in Eqn. 3). In the case of negative slack, delays are updated and the margin between the minimum and maximum delay paths of an RTC are maintained. To fix a path with negative slack, the target delay is updated based on the size of the negative slack and the convergence values W_p and D_m. The delays for a single path between synthesis runs can vary greatly, particularly when some paths are poorly placed. Converging too quickly allows the algorithm to settle to a poor local minimum. By slowly converging to a closed solution, poor placement is significantly reduced and PPA improves. In Algorithm 1 the target delay will be updated by either 80% of the negative slack, or 100ps, whichever is smaller. The cost is increased iterations and run time.

The results of the last three iterations are compared to detect non-convergence (normally related to the min delay constraints). If they are equal, no progress has been made toward convergence and the program terminates with paths that have negative slack.

Several extensions to the algorithm have been implemented. To bypass non-convergence, margin relaxation is performed, where margin for critical paths is raised to accommodate variations and timing. Timing relaxation can be performed to improve power based on cell size on critical paths. Timing targets for multiple paths can be shared with a single target variable. This can be used to mitigate cycle time variation between pipeline stages and to reduce tool run times.

A design can have thousands of relative timing constraints. Thus, employing an SQL database to search for and update RT constraints and delay targets substantially improves the run time performance of HBTC. The constraint data for each design is managed under delay and margin tables, with separate tables for both synthesis and physical design runs. The delay table contains rows for both minimum and maximum delay constraints. HBTC verifies constraint data for missing maximum or minimum delay paths entries and duplicate constraint entries.

C. Timing Violations After Timing Closure

HBTC assumes that synthesis and PnR is performed by the commercial EDA tools in a single pass. This requires representing the circuit as a directed acyclic graph (DAG) [7]. Many relative timed designs will have sequential circuits built from combinational gates similar to the circuit in Fig. 2. In such designs, full timing paths of an RTC will often contain cycles. The process of creating a DAG representation of cyclic timing paths will result in an incomplete timing path specification for these RTCs. Therefore, when the HBTC tool closes timing, it is doing so on a segment of the full RTC timing path. In such cases a closed design, with no negative slack on the path segments, can still fail full path timing because the full cyclic path cannot be provided to the commercial EDA tools. Also in these cases, simulation through ModelSim may identify the violation, as reflected by sim-errors column in Table I. The means of formally identifying full-path cyclical timing verification are outside the scope of this work, but will be included in future implementations.

V. Evaluation

This paper reports on applying the HBTC tool to converge timing on designs with varying complexity at the 40 nm cmos technology node. These include a watchdog timer, wakeup timer, timer, FFT_64, and peripheral system design which contains the timers, a network-on-chip, and scratchpad register file. The three timers are all “clocked” designs that have been optimized using relative timed techniques to significantly reduce power. The FFT-64 and network-on-chip used in the peripherals design are bundled data asynchronous handshake designs. The small designs contain approximately 100 RT constraints, with the large designs containing approximately 50,000 RT constraints and over 30,000 pipeline latches.

Synopsys Design Compiler is used for synthesis, ModelSim for simulation, and PrimeTime for path delays and power analysis. The results are compared for power, performance, run time, and simulation errors that occur after the timing closure run. Two starting points are employed, one with zero maximum delay targets and a second set using user-defined delay targets that are close to a timing closed solution.

VI. Results

Table I shows results obtained in two scenarios. The first set is where a designer specified critical path delays and manually drove the timing targets to get close to a closed solution. The second set allows HBTC to fully close the design by assigning zero delay targets to all max delay paths. These two approaches are compared with manual design effort used as the baseline. This shows that HBTC tool can produce faster and lower power designs when no targets are provided. However, this comes at a considerable run time cost for larger designs. CPU run time for the HBTC algorithm is less than 2 seconds for all examples.

The number of iterations is critical for run time, because synthesis or place and route is in the inner loop. When starting with timing constraints dictating desired performance, the small watchdog timer converged in two cycles. The more complicated peripheral and FFT-64 designs required 8 or 9 cycles to converge. Many more iterations are required when HBTC closes a design when no timing targets are provided. The timer designs took between 10 and 25 iterations to converge. The iterations for the peripherals more than tripled. The FFT-64 design employed tcl variables for the zero delay start point to help identify frequency domains, so that design converged in 18 iterations. Most of the designs simulated without timing violations. To completely close a design, a full path timing validation must also be applied which supports cyclic timing paths.

TABLE I: HBTC results for two configurations and their comparison with zero-max delay target design as baseline.

Designs	Close to Convergence Target						Zero Max Delay Target						Compare – Zero / Close			
	child process run time (s)	No. of itera-tions	Power (uW)	Energy (pJ)	sim. errors	$e\tau^2$	child process run time (s)	No. of itera-tions	Power (uW)	Energy (pJ)	sim. errors	$e\tau^2$	Run time (x)	Power	Energy	$e\tau^2$
Watchdog	1,646.48	2	56.88	71.19	1	11.15	1,735.86	20	52.37	64.87	0	9.95	1.05	0.92	0.91	0.89
Wakeup	634.84	7	54.56	30.58	0	0.96	1,972.70	25	52.53	28.80	0	0.87	3.11	0.96	0.94	0.91
Timer	842.65	3	41.38	87.52	2	40.64	2,073.17	10	39.70	85.55	2	38.27	2.46	0.96	0.98	0.94
Peripherals	2,811.87	9	135.80	147.90	1	175.40	25,230.97	33	132.30	143.10	3	167.41	8.97	0.97	0.97	0.95
FFT-64	1,972.70	8	12,100	20.43	0	58.21	32,758.92	18	12,400	20.93	0	59.64	16.61	1.02	1.02	1.02

Most of the designs demonstrated improved energy efficiency and performance when the HBTC tool was allowed to perform timing closure from zero delay targets. In general, the smaller designs provided greater improvements than the larger designs. The FFT-64 design is a multirate design, and used a different design target approach as the zero max delay target design employed tcl variables for common frequency targets which reduced the iterations. The user specified frequencies helped the design PPA. The number of iterations required to timing close a system design like peripherals is reduced by employing HBTC on individual hierarchical blocks.

The HBTC tool provided a rapid approach to converging timing for designs which otherwise could only be closed manually at great time and effort. It is well integrated in commercial EDA flow and scalable to different process technologies that could be represented in relative timed methodology.

VII. Conclusion

Relative timing is a method for specifying and optimizing digital integrated circuits. A heuristic algorithm was developed to automate timing closure of a complex set relative timing constraints. The HBTC tool provides a fully automated approach to converging timing for designs which otherwise could only be closed by hand.

The heuristic based timing closure tool is implemented and applied to designs reported in this paper. The tool provides reasonable run times with a synthesis tool, such as Design Compiler, in its inner loop. The tool can begin the closure process agnostic of technology node delay information by using zero delay starting point targets. Starting from this point, this tool produced results for several designs showing 5% power reduction and about 8% better performance in our design suite compared to designs that started with a near-converged values provided by designers. This relative timing closure tool is designed to work well with the commercial EDA tools and is well integrated with the design flow.

References

[1] R. Nair, C. L. Berman, P. S. Hauge, and E. J. Yoffa, "Generation of Performance Constraints for Layout," *IEEE Transactions on Computer-Aided Design*, vol. 8, no. 8, pp. 860–874, Aug 1989.

[2] 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.

[3] J. V. Manoranjan and K. S. Stevens, "Qualifying Relative Timing Constraints for Asynchronous Circuits," in *International Symposium on Asynchronous Circuits and Systems*, May 2016, pp. 91–98.

[4] T. Sharma and K. S. Stevens, "Physical Design Variation in Relative Timed Asynchronous Circuits," in *IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, July 2017, pp. 278–283.

[5] I. E. Sutherland, "Micropipelines," *Communications of the ACM*, vol. 32, no. 6, pp. 720–738, June 1989.

[6] 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.

[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] F. Burns, D. Shang, A. Koelmans, and A. Yakovlev, "An Asynchronous Synthesis Toolset using Verilog," in *Design, Automation and Test in Europe (DATE)*, vol. 1, Feb 2004, pp. 724–725.

[9] X. Jia and R. Vemuri, "CAD Tools for a Globally Asynchronous Locally Synchronous FPGA Architecture," in *19th International Conference on VLSI Design*, Jan 2006, p. 6 pp.

[10] C. J. Myers, T. G. Rokicki, and T. H.-Y. Ming, "Automatic Synthesis of Gate-Level Timed Circuits with Choice," in *16th Conference on Advanced Research in VLSI*, March 1995, pp. 42–58.

[11] R. Kol, R. Ginosar, and G. Samuel, "Statechart Methodology for the Design, Validation, and Synthesis of Large Scale Asynchronous Systems," in *2nd International Symposium on Advanced Research in Asynchronous Circuits and Systems*, March 1996, pp. 164–174.

[12] B. A. Brady, A. K. Jones, and I. S. Kourtev, "Efficient CAD Development for Emerging Technologies using Objective-C and Cocoa," in *11th International Conference on Electronics, Circuits and Systems*. IEEE, Dec 2004, pp. 369–372.

[13] M.-D. Shieh, J.-M. Horng, M.-H. Sheu, and Y.-C. Hsu, "A CAD System for Automatic Synthesis of Generalized Asynchronous Circuits," in *International Symposium on Circuits and Systems (ISCAS)*, vol. 4. IEEE, May 1996, pp. 818–821.

[14] A. Kondratyev and K. Lwin, "Design of Asynchronous Circuits Using Synchronous CAD Tools," *IEEE Design & Test of Computers*, vol. 19, no. 4, pp. 107–117, July-Aug. 2002.

[15] K. S. Stevens, S. Rotem, S. M. Burns, J. Cortadella, R. Ginosar, M. Kishinevsky, and M. Roncken, "CAD Directions for High Performance Asynchronous Circuits," in *Proceedings of the Digital Automation Conference (DAC99)*. IEEE, June 1999, pp. 116–121.

[16] W. Belluomini and C. J. Myers, "Efficient Timing Analysis Algorithms for Timed State Space Exploration," in *3rd International Symposium on Advanced Research in Asynchronous Circuits and Systems*, April 1997, pp. 88–100.

[17] M. Bibiluka, M. T. Moreira, and N. L. V. Calzans, "A Bundled-Data Asynchronous Circuit Synthesis Flow Using a Commercial EDA Framework," in *Euromicro Conference on Digital System Design*, Aug 2015, pp. 79–86.

[18] G. Miorandi, M. Balboni, S. M. Nowick, and D. Bertozzi, "Accurate Assessment of Bundled-Data Asynchronous NoCs Enabled by a Predictable and Efficient Hierarchical Synthesis Flow," in *23rd International Symposium on Asynchronous Circuits and Systems*. IEEE, May 2017, pp. 10–17.

[19] Y. Thonnart, E. Beigné, and P. Vivet, "A Pseudo-Synchronous Implementation Flow for WCHB QDI Asynchronous Circuits," in *International Symposium on Asynchronous Circuits and Systems*. IEEE, May 2012, pp. 73–80.

[20] 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.

[21] M. C. Seiler and F. A. Seiler, "Numerical Recipes in C: The Art of Scientific Computing," *Risk Analysis*, vol. 9, no. 3, pp. 415–416, Sept 1989.