Timing Driven Placement for Quasi Delay-Insensitive Circuits

Robert Karmazin, Stephen Longfield Jr., Carlos Tadeo Ortega Otero, Rajit Manohar
Computer Systems Laboratory, Cornell University
Ithaca, New York 14853, U.S.A.
{rob, slongfield, cto3, rajit}@csl.cornell.edu

***Abstract*—Asynchronous circuits offer promise in handling current and future technology scaling challenges. Unfortunately, their impact has been limited by the lack of design automation. We present A-NTUPLACE, a timing-driven placer uniquely suited to handling quasi delay-insensitive circuits. Our tool uses a generalization of repetitive event rule systems to identify critical signal transitions. The cell placement engine, based on a leading academic placer, NTUPlace3, incorporates net weights to minimize critical wirelengths as well as a novel balancing scheme to ensure isochronic fork constraints are met. We show that our placer is effective at both prioritizing selected nets and balancing forks, demonstrating improvements in 3 of our 4 benchmarks.**

***Index Terms*—Design Automation, Asynchronous Circuits, Optimization.**

I. INTRODUCTION

As device dimensions decrease, variability increases, as does device susceptibility to environmental variations, such as temperature and voltage. The resulting gap between designers' expectations and the measured performance has resulted in ever higher margins being included in synchronous designs just to ensure correct operation.

Quasi delay-insensitive (QDI) circuits offer an elegant solution to this problem. Their self-timed nature allows them to tolerate arbitrary gate delay, freeing designers from worrying about incorrect operation due to timing violations. Furthermore, QDI circuits can be optimized for average case performance, since the critical paths through asynchronous systems are defined only by their active pipeline stages [18]. Contrast this with synchronous systems, where the global clock period is defined by the slowest pipeline stage, regardless of how often that stage is executed. QDI circuits are also capable of achieving an ultra low power envelope; stages that are not being executed are completely devoid of switching activity, consuming only leakage power when idle.

Unfortunately, despite of these advantages, QDI circuits have remained on the fringe of both academic and industrial design. The oft-stated, and widely accepted, reason for this is the lack of design automation suited to correctly and efficiently handle QDI circuits and their particular timing constraints. Some work has been done to address this shortcoming on all stages of the design process: from high-level compilation and synthesis of QDI descriptions to low-level gate mapping and physical implementation [1,4,9,36]. However, most of these solutions make some sort of compromise, whether in the generality of circuit family or design methodology supported, the efficiency of the implementation, or the overall performance of the produced circuitry.

In this paper, we focus on the problem of timing driven placement. Most of the previously published solutions attempt to fit circuit topologies and timing assumptions unique to asynchronous designs into pre-existing, synchronous tool flows. While this is useful in that it lets asynchronous designers take advantage of the decades of development in these tools, it has many limitations. Most significantly, traditional synchronous timing analysis requires an acyclic netlist. While this may be a good match for some asynchronous families, QDI circuits have many interacting logic loops, which need to be cut to fit into this paradigm. Picking how to cut these loops can be challenging, as incorrect cuts may dramatically affect the accuracy of the timing solution. This challenge is the reason why many tools limit their application space to template-driven design, where the cut locations can be statically determined. However, while template-driven design has allowed for great productivity increases, it limits designer flexibility, and may remove some of the advantages of QDI design.

Furthermore, synchronous tools are not aware of certain assumptions unique to QDI designs, namely the isochronic fork timing assumption. This assumption relates with signal transitions that are not acknowledged by all its endpoints. If such a transition can occur, we must assume this transition is visible to all forks of the wire within some bounded time [16]. This type of timing assumption is not present in synchronous design, and can be easily violated by synchronous tools unaware of its existence.

In this work, we address these issues by presenting A-NTUPLACE, a timing aware placer ideally suited to handling QDI circuit topologies and their associated timing constraints. Our placer is based on a leading academic placer NTUPlace3 [3], which generates placement solutions comparable to commercial CAD software. Our timing analysis method, based on repetitive event rule systems, is general and can be applied to any stable and non-interfering Production Rule Set (PRS). The timing information is then passed to the placer in the form of net weights, prioritizing certain nets over others. To ensure no timing assumptions are violated, A-NTUPLACE adds explicit constraints to balance isochronic forks, a unique feature of our placer.

In Section II we discuss previous work related to both timing analysis for synchronous and asynchronous systems as well as placers implementing various placement techniques, Section III details the implementation of A-NTUPLACE, describing both the timing formulation as well as placement strategies. In Section IV we evaluate A-NTUPLACE against

several asynchronous circuit benchmarks for which previously placed and routed solutions exist.

II. Related Work

A. Timing Models and Analysis

1) Synchronous: In clocked circuits, timing analysis has long been a pressing issue, due to the need to guarantee that critical paths can meet target cycle time. For the purposes of timing-driven placement, most of these analyses can be grouped into two classes: *net-based* and *path-based*, depending on if their results are given as net weights and constraints, or as path weights and constraints [26]. Though path-based methods more accurately represent the underlying problem, due to the possible exponential number of paths, they are typically either too computationally expensive, or only applied to a subset of the paths [5].

The timing analysis method with the lowest computational overhead is doing net-based static timing analysis with the critical path method [15]. This method uses a topological sorting of the gates to compute the set of *arrival times* for each net from the timing start points, and a set of *required times* from the timing end points. The *slack* of each net is the difference between the required time and the arrival time, and if it is positive, indicates that the signal can be delayed without affecting the overall delay. Due to the use of the topological sort, the time complexity of this method is linear w.r.t. the number of gates and wires in a netlist.

2) Asynchronous: Much of the work in characterizing the timing behavior of asynchronous circuits has either been based around the framework of a Timed Marked Graphs (TMG), a class of Petri nets [21], or in Event-Rule (ER) systems, which have the same expressive power [2]. Using these frameworks, there exist efficient algorithms to find the period of the system for high-level performance characterization [2,7]. However, neither TMGs nor ER systems allow for the expression of conditionality or data-driven behavior, limiting their applicability. There has been recent effort into extending Petri-net based analyses to cover conditional circuit behavior [22], but these methods reason at a higher layer of abstraction than the gate level. Therefore, placement tools for asynchronous circuits typically limit their analysis to template based systems, and use the known properties of these templates to simplify the analysis process, at the cost of generality [31].

B. Cell Placement

Cell placement can be formulated as a constrained optimization problem whose objective is to assign physical positions to modules within a fixed area such that some metric is optimized while guaranteeing placement legality (i.e., non-overlapping modules). The challenge of solving this problem results from its size; cell placers often must handle designs with up to millions of cells or more, requiring techniques that can scale with the design.

Placement strategies that have been developed over the last 30+ years can be broadly categorized into three categories:

1. Stochastic: This method usually involves using the simulated annealing heuristic: an initial placement is perturbed with some probability, which is decreasing based on some annealing schedule. For example, TimberWolf [28] employs three possible cell perturbations, each with an independent probability. This technique produces good quality results for smaller designs, and is flexible enough to easily handle multi-objective optimization, but tends to be slower and less scalable than other techniques.

2. Partitioning: This method divides the circuit into smaller subcircuits, minimizing the number of net connections across partitions. Typically a min-cut heuristic is recursively applied to the design until some minimum sized partition threshold is reached. Module placement is performed by simultaneously dividing the die area into bins, to which partitions are assigned. For example, Capo [27] uses a multilevel implementation of the Fiduccia-Mattheyses algorithm [6] to perform recursive bi-partitioning. Dragon [34] uses a hybrid technique: The design and placement area are recursively 4-way partitioned, and simulated annealing is used to assign design partitions to bins. Overall, partitioning techniques are quite scalable, and handle mixed-sized designs well. However, they are limited by the types of objective functions they can optimize for.

3. Analytical: This method uses a mathematical programming approach, which defines an objective function subject to constraints, which can then be optimized using analytical approaches. Placers using this technique can be further categorized as either quadratic or non-linear, based on the nature of the objective function. Quadratic placers, such as Kaftwerk2 [30], Gordian [10], FastPlace [32] and RQL [33], use a quadratic cost function to model wirelength. Non-linear placers, such as NTUPlace3 [3] and APlace [8], use non-linear cost functions to better model net wirelengths. Overall, analytical placers tend to be more scalable and can handle multiple objectives and constraints, depending on the problem formulation. However, certain aspects, such as whitespace management and module rotation and orientation, are hard to capture using mathematical programming.

C. Asynchronous CAD

Many placement solutions have also been proposed in the asynchronous community. Weaver [29] synthesizes a synchronous netlist from a VHDL specification, and then replaces combinational gates with their QDI equivalents. The design is mapped to a small library before placement. Proteus [1] uses a logic clustering approach and a more diverse cell library to mitigate the area costs of Weaver. However, both approaches require the final design be implemented with the Pre-Charge Half Buffer (PCHB) template [13], and both rely on the circuit library to guarantee timing constraints, such as bounds on isochronic forks. Recently, Wu et al. [35] used a Lagrangian relaxation approach to enforce timing constraints on critical cycles within the Proteus flow.

Approaches using other template styles have also been explored, such as a psuedo-synchronous approach [31] which generates Weak-Conditioned Half Buffer (WCHB) template circuits and employs standard synchronous techniques to characterize cells and generate timing constraints. Loops are cut at the inputs to certain C-elements to create directed acyclic graphs (DAGs), allowing for synchronous timing driven placement.

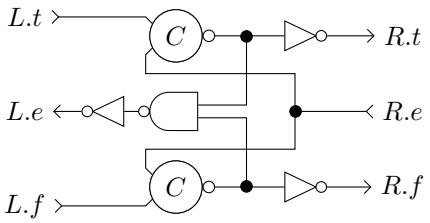


Fig. 1: Weak Conditioned Half Buffer

CPlace [11] takes a different approach, unlike all the placers mentioned above, which are iterative, fixed die placers, CPlace is a constructive placer, meaning that cells are placed sequentially. This allows for direct control of wire lengths and bounds on isochronic forks, but tends to require a much larger placement area.

III. Timing Driven Placement

A. Timing Model

We use Repetitive Event Rule (RER) to reason about performance in asynchronous circuits. RER systems are defined by:

- E, a set of transitions
- R, a set of rule templates, written as $\langle u, i-\epsilon\rangle \overset{\alpha}{\mapsto} \langle v, i\rangle$:
  - u, $v \in E$ are the transitions
  - $\alpha \in [0, +\infty)$ is the delay
  - $\epsilon \in \{0, 1\}$ is the occurrence-index offset and
  - i is the instantiation index

and commonly represented using a collapsed constraint graph [2]. For the sake of circuit analysis, these graphs are restricted to only finite, strongly connected graphs, where cutting the edges with $\epsilon = 1$ (also known as marked edges, or back edges) cuts all cycles in the graph.

Intuitively, each transition represents either a low-to-high transition (e.g., $a\uparrow$), or a high-to-low transition (e.g. $a\downarrow$), and each of the rules represents causality between these transitions. When a transition has more than one incoming rule, it waits for all of the rules to be ready before firing. If a rule has $\epsilon = 1$, this corresponds to an connection between one "iteration" and the next, allowing a finite RER to represent infinitely-executing sequences of transitions.

RER systems can be constructed from circuits using an algorithm known as index-priority simulation, which typically runs in linear time, though may exhibit worst-case exponential time in aberrant systems [12]. To reason about sub-systems separately, and thereby cut down on the run time, RER can be constructed as Partitioned RER (PRER) [14], which distinguish the circuit and its *environment*, shown in the collapsed constraint graph as boxes around these *external* transitions.

As an example, consider the WCHB shown in Figure 1 [13]. If we augment this with an environment that is always sending a **true** token on the L channel, and always receiving on the R channel, we can generate the PRER shown in Figure 2a, where delays have been elided for the sake of compactness, and edges with $\epsilon = 1$ are decorated with black dots. Though this RER is useful for performance analysis, it is insufficient for doing placement, as the resulting graph does not cover every transition, only those that result from the **true** token. While it is possible to repeatedly run this analysis for all possible inputs, this would quickly become intractable for larger systems.

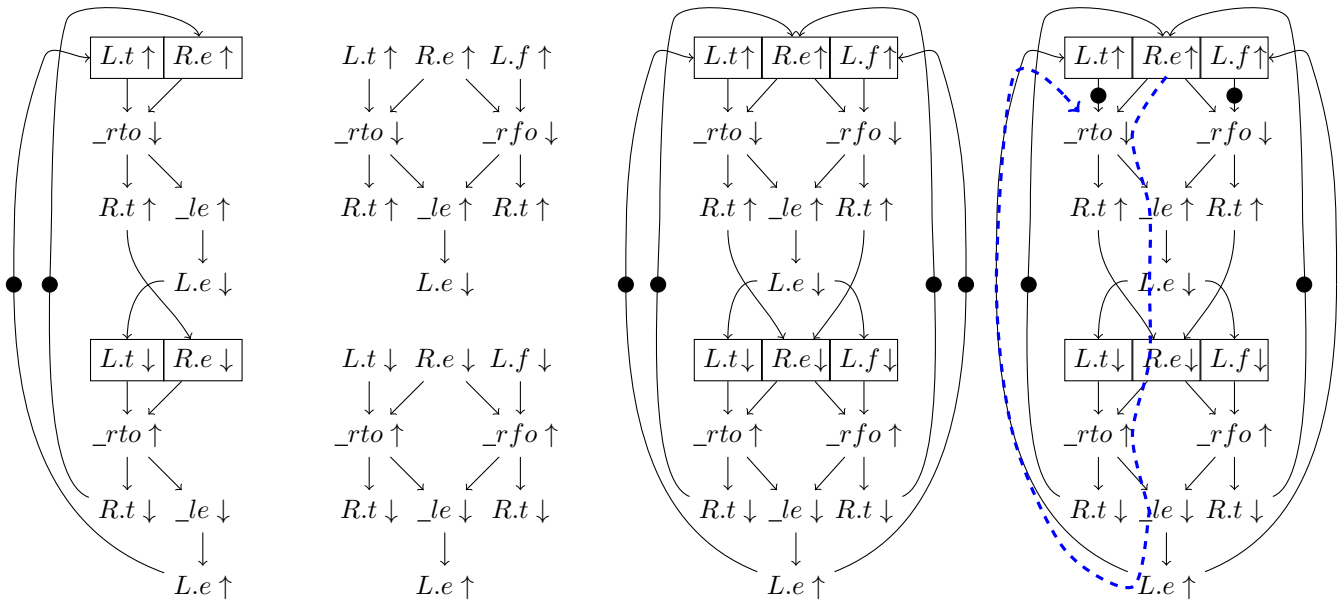
As an alternate method to fill in these gaps, consider the gate connectivity graph of the WCHB, as shown in Figure 2b. The RER, which represents causality, is necessarily a subgraph of this graph, as in QDI circuits there is no causality except that which comes from direct dependencies. However, this connectivity graph lacks two features of RERs which are needed for analysis: first, it lacks the environment that would complete the graph, and second, it lacks the marked edges which connect one iteration to the next. Using what we know about the dual-rail datatype used in the WCHB, we can generalize the PRER over the connectivity graph, to create the approximation of the PRER seen in Figure 2c. As the original RER is a subgraph of this graph, any critical cycle that exists in the original graph must exist in the new approximation, and so this will only ever overestimate cycle time.

If this information about data symmetry is not available, or if index-priority simulation is too computationally expensive, marked edges can also be placed on the connectivity graph using depth-first search (DFS) to find back edges. As an example of this, see Figure 2d, where a DFS initiated from $R.e\uparrow$ (indicated as a dashed blue line) discovers $L.t\uparrow \mapsto \_rto\downarrow$ as a back edge and marks it. While this procedure does guarantee that if these edges are cut, all cycles in the graph will be cut, it does not guarantee that the placement of the marked edges has any meaning.

Through testing, we found the best results were obtained by generalizing from the data whenever such symmetry was available, using DFS to detect cycles in the graph that are not cut by the marked rule, and then removing the last rule in this cycle. This allows us to generalize over data values whose treatment is symmetric, but not generalize over data where the treatment of different values was asymmetric (e.g., control values). While this method does require multiple runs to get complete coverage, this coverage is easily measurable, and the number of required runs is typically small.

Once this model is generalized, it can be used for STA, but the methods we use are not quite the same in traditional clocked STA. Similarly, we cut all of the marked edges (analogous to the flip-flop inputs/outputs from the synchronous world) to get a DAG, which we topologically sort. However, unlike in clocked systems, where all of the events at the "top" of this DAG are activated at the same time (the beginning of the clock cycle), and all events at the "bottom" commit at the same time (end of the clock cycle), asynchronous circuits do not have such alignments. Additionally, it is incorrect to approximate the system in this way, as the schedule of marked events can drastically affect the time separation of later events [20].

Instead, we take advantage of the fact that there exist efficient algorithms to identify average cycle time of these graphs and, from that, find the critical cycles of the graph [2]. For each of the critical rules that is also a marked rule, we can run the event-initiated timing simulation [24], defined for



(a) WCHB **true** token PRER (b) WCHB gate connectivity (c) WCHB generalized PRER (d) WCHB DFS PRER

Fig. 2: Approximating the WCHB RER

initiating event g as:

$$t_g(f) = \begin{cases} 0 & \text{if } f = g \text{ or } f \not\prec g \\ \max\{t_g(e) + \tau | g \preceq e \wedge e \overset{\tau}{\mapsto} f\} & \text{otherwise} \end{cases}$$

where $f \prec g$ means that event f precedes event g in the RER unrolling. Intuitively, this is the schedule for an event given that a particular critical rule dominates its execution time. Given the topological sort of the RER after cutting the marked edges, we can determine one "unrolling" of the event-initiated timing simulation in linear time.

For an individual event-initiated timing simulation, we can compute the required time for each event by taking the reverse topological sort, and determining the latest possible time that this event can be scheduled while maintaining the required time of its successors. Then, we compute the *slack* of a particular rule by subtracting the scheduled time from the required time, giving the amount of delay that can be added to this rule without affecting system performance. This procedure is repeated for every critical marked edge, looking for the minimum slack for each rule, to ensure that the slack used for placement is conservative.

We incorporate net criticality information from this timing simulation into our placer by generating *net weights* from the computed slack of each net. Since this type of simulation is data dependent, we perform multiple simulations in order to ensure maximal coverage of transitions. Net slacks are computed by averaging the worst case slacks across multiple timing simulations. Net weights are computed using (1) from [17]

$$w_e = \left(1 - \frac{slack_e}{T}\right)^{\alpha} \tag{1}$$

where $slack_e$ is the computed slack for net e, T is the maximum slack across all nets in the design, and α is the criticality exponent, typically set to 2. A detailed description on how we fit net weights into our placer is found in Section III-B.

B. Analytical Placement

A-NTUPLACE inherits many features from NTUPlace3 [3]. It can be classified as a fixed-die, multi-level analytical placer. Its objective is to minimize the total wirelength of the design under given density constraints. This can be formally written as (2).

$$\begin{aligned} \min \quad & W(\mathbf{x}, \mathbf{y}) \\ \text{s.t.} \quad & D_b(\mathbf{x}, \mathbf{y}) \leq M_b \end{aligned} \tag{2}$$

where $W(\mathbf{x}, \mathbf{y})$ is the wire length function, typically represented using half perimeter wire length (HPWL), as defined in (3), $D_b(\mathbf{x}, \mathbf{y})$ is the density function for a particular bin b, and M_b is the target density for that bin. This formulation assumes the netlist is represented as a hypergraph, with each net represented as a hyperedge e.

$$\text{HPWL}(\mathbf{x}, \mathbf{y}) = \sum_{e \in E} \left(\max_{v_i, v_j \in e} |x_i - x_j| + \max_{v_i, v_j \in e} |y_i - y_j| \right) \tag{3}$$

Like NTUPlace3, our placer solves the above constrained minimization problem by using the penalty method; The original objective function (2) is transformed into the following:

$$\min \quad W(\mathbf{x}, \mathbf{y}) + \lambda \sum_b (D_b(\mathbf{x}, \mathbf{y}) - M_b)^2 \tag{4}$$

where the density constraint is converted into a penalty term scaled by λ. As is, (4) cannot be solved by analytical techniques because neither the wirelength function $W(\mathbf{x}, \mathbf{y})$ nor the density penalty $D_b(\mathbf{x}, \mathbf{y})$ are smooth and differentiable. Instead, an approximation for (4) is used, where the wirelength is represented as the Log-Sum-Exp (LSE) function described by Naylor [23], and the bin density is computed using a

bell-shaped curve as in APlace [8]. This yields an objective function that is smooth and differentiable, lending itself to standard analytical techniques, such as the conjugate gradient method.

We begin by first adding support for our timing model described in section Section III-A. The NTUPlace3 placer treats all nets equally, with the aim of minimizing the total wirelength across all nets. However, nets deemed more critical by our timing model must be given a higher priority, possibly at the expense of other, less critical nets. To achieve this, A-NTUPLACE annotates each net with a weight; a larger weight indicates a more critical net. The net weights are computed using (1), which use the slacks generated by the timing simulation as described in Section III-A. Altering net criticality affects multiple aspects of the placement algorithm:

- The clustering stage is used to reduce the number of movable blocks in the design by grouping modules with high affinity, thus allowing the placer to scale to very large designs. By changing the weight of a given net, we change the affinity between two modules connected by that net, and thus, change the likelihood they would be grouped into a cluster. Modules that are clustered together tend to be placed close to one another in subsequent declustering and placement iterations.
- The analytical solver operates on the LSE wirelength approximation, whether the LSE is for the movable modules specified in the design, or groups of modules formed during clustering. Thus, for our placer to respect net criticality, weights must be included in the LSE formulation:

$$\gamma \sum_{e \in E} w_e \left(\ln \sum_{v_k \in e} \exp(\frac{x_k}{\gamma}) + \ln \sum_{v_k \in e} \exp(\frac{-x_k}{\gamma}) + \ln \sum_{v_k \in e} \exp(\frac{y_k}{\gamma}) + \ln \sum_{v_k \in e} \exp(\frac{-y_k}{\gamma}) \right) \quad (5)$$

  where w_e is the weight assigned to net e, x_k, y_k are the x and y coordinates of module v_k, and γ is a smoothing factor. The LSE approximation for the HPWL is convex, which is not affected by the scaling term w_e. Note that as γ approaches 0, the LSE approaches the HPWL in (3).
- The detailed placer uses a sliding window to locally optimize cell positions within that window. For each window position, a cost matrix representing the change in wirelength resulting from moving a cell to a new position, or swapping two similarly sized cells, is built. A bipartite matching problem is formulated based on moving cells to these new potential positions. Net weights are used to adjust the cost matrix, putting greater emphasis on shortening higher-priority nets.

We further augment A-NTUPLACE with constraints on wire forks to ensure that all isochronic fork assumptions are met. This is accomplished using two methods. First, in the global placement stage, the objective function described in (4) is augmented with an additional penalty term which penalizes nets for having excessively large differences in distances from net driver to destinations. The new objective function becomes:

$$\min \quad \hat{W}(\mathbf{x},\mathbf{y}) + \lambda \sum_b (D_b(\mathbf{x},\mathbf{y}) - M_b)^2 + \mu \sum_{e \in E_I} \max_{n \in FO(e)} (d_n - d_{lim,e}, 0)^2 \quad (6)$$

where d_n is the Euclidean distance from driver module to fanout module n, $d_{lim,e}$ is the maximum allowed distance from net driver to a fanout module for net e, μ is a scaling term similar to λ, and E_I is the subset of nets marked as potentially isochronic. The set E_I is conservatively defined as containing all nets that have exactly one driver and has fanout of greater than one.

Second, in the detail placement stage, the cost matrix is further modified to incorporate penalties for any cell movements that could potentially violate the isochronic fork assumption. For every module in the current window, a penalty is computed for every net connected to that module that has been flagged as potentially isochronic. The value of this penalty depends on whether the current module is the net driver or is in the fanout set of that net. If module n is in the fanout set, then the penalty of violating the isochronic assumption is

$$P_{n,e} = (d_n - d_{lim,e})^2 \quad (7)$$

which is the same as in (6). If the module is driving the net, then it receives a penalty from all the other modules connected to it:

$$P_{m,e} = \frac{HPWL_{FO(e)}}{HPWL} \sum_{n \in FO(e)} (d_n - d_{lim,e})^2 \quad (8)$$

where $P_{m,e}$ is the value of penalty applied to module m from net e, n represents all the modules connected to net e (excluding module m). The factor $\frac{HPWL_{FO(e)}}{HPWL}$, the ratio of the HPWL of only the fanout modules to the total HPWL of the net, is used to scale the penalty based on how tightly the fanout modules are placed; if they are placed close together, then moving the net driver should not make a big difference in terms of the isochronic timing assumption.

IV. EVALUATION

We evaluate our placer on several benchmarks from a fully-implemented and silicon-verified ultra low-power asynchronous sensor network processor [25]. These designs encompass a wide variety of logic and implementation styles. Each benchmark was completely specified using sequential CHP, it was then compiled down to a PRS following Martin's synthesis procedure [19]. The PRS is then converted into a cell library using *cellTK* [9], a cell generation tool capable of creating physical layout for cells with arbitrary, non-static logic topologies, a frequent occurrence in QDI designs. A-NTUPLACE then uses these cells to assemble the benchmarks. The evaluated benchmarks are listed in Table I.

A. Timing Model

As our timing model is based on the circuit structure, instead of the RER itself, and because we are simultaneously reasoning about all possible data values, we are only approximating the timing solution. To determine the accuracy of this

TABLE I: Evaluated Benchmarks

Benchmark	Unique cells	Total cells	Total Nets	Baseline HPWL
Logic	13	103	178	8437
Shift	51	1065	1132	66712
Decode	56	282	329	26048
Fetch	100	571	636	39975

approximation, we can compare it to the brute force method of looking at all possible RER solutions for each benchmark. As the number of possible RER systems is exponential w.r.t. the size of the inputs, we limit our comparison to a random subsampling of 10,000 brute force instances for each benchmark. Table II presents four comparisons for our benchmarks:

- Cycle time error relative to median brute force cycle time
- Fraction of transitions identified by the approximation as critical that are never critical in the brute-force evaluation
- Fraction of transitions identified as non-critical that are ever critical in the brute-force evaluation
- Root-mean-squared-deviation (RMSD) of the slack.

In all cases, there were typically few false positives, only a small number of false negatives, and the approximated slack was very close to the slack computed using the brute force methods. The cycle time was also typically close, however, there was significant deviation in the Shift unit. This stems from the highly variable cycle time of the Shift unit, as the static approximation can only have a single cycle time, but several different cycle times were seen during the brute-force exploration. Nevertheless, this does not substantially affect the accuracy of the slacks, as they are independent of the cycle time, and only dependent on interactions of local delays.

TABLE II: Timing Model Evaluation

Benchmark	Cycle Time Error	False Pos Critical	False Neg. Critical	RMSD Slack
Logic	0.00 %	0.00 %	8.33 %	1.20 %
Shift	67.62 %	0.00 %	1.60 %	0.90 %
Decode	0.07 %	17.76 %	4.63 %	7.30 %
Fetch	8.60 %	3.43 %	3.45 %	3.30 %

B. Placement

We evaluate A-NTUPLACE by first comparing it to a state-of-the-art baseline commercial standard cell placer. As mentioned above, we first convert our PRS into a standard cell library using *cellTK*, where roughly every pair of Production Rules (PRs) driving a node in the PRS corresponds to one cell in the library. The designs are placed by our baseline commercial placer with fairly aggressive density targets, all greater than 90%. Note that no timing information is passed to our baseline placer, as *cellTK* does not characterize the cells it generates. Once placed, the commercial placer's core area and pin locations are used to initialize A-NTUPLACE in order to ensure a fair comparison. The execution times of our placer are not evaluated, but are observed to be comparable to the unmodified NTUPlace3 for our four benchmarks.

We first evaluate how well our placer performs on asynchronous netlists with uniform net weights and no fork balancing (ie, default settings). Specifically, we look at various features of the placer that could introduce unexpected and unpredictable behavior, such as the huersitic techniques employed by look-ahead legalization (LAL) during global placement and local search during detailed placement. The results are shown in Table III. All data is with respect to the baseline HPWL. For every configuration, the final placement is legal with all density constraints met.

TABLE III: HPWL Results

Benchmark	Baseline	No LAL	With LAL	With Detail
Logic	8437	1.30 %	−0.03 %	−10.10 %
Shift	66712	7.80 %	2.40 %	−3.80 %
Decode	26048	−15.80 %	−17.80 %	−21.50 %
Fetch	39975	7.7 %	2.2 %	−5.50 %

As can be seen, all of the benchmarks benefit from performing LAL during the global placement stage, and also from running the detailed placer stage after legalization. This conforms with previously published results for other placers on synchronous benchmarks

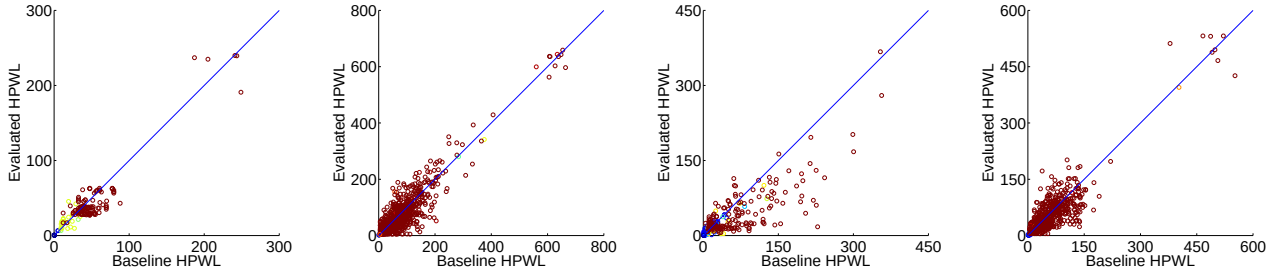
Figure 3 shows how each net is benefited by the net weights generated by our index-priority simulation on the RER. Here, each net is represented by a point in the scatter plot. The darker the shade of red, the more critical the net, and thus, the higher the weight. The X-axis and Y-axis represent the HPWL of the baseline placer and A-NTUPLACE with wire weights, respectively. The coordinates of each point are scaled by the net weight. A reference line $y = x$ is added; points below that line indicate an improvement in HPWL for that net.

As can be seen in Figure 3, many of the nets in the design are in fact critical. This is not uncommon for balanced, slack matched QDI circuits, since the same phenomenon occurs in well-balanced synchronous circuits as well. This observation is confirmed by our brute force evaluation of the RERs for each benchmark. Benchmarks with many critical nets, such as the shift and fetch units, have many nets with the same weight, and thus, does not differentiate much from the baseline, resulting in points close to the reference line. However, the logic and decode units have greater variety in their net wet weights, allowing for a greater differentiation in resulting wirelengths. Indeed, many of the points representing critical nets are below the reference line, indicating an improvement in HPWL for those nets.

C. Fork Balancing

As discussed in Section III-B, two different methods for balancing isochronic forks are implemented in A-NTUPLACE: one is a modification to the objective function in the global placement stage, and the other is augmenting the cost matrix in the detailed placement stage. We evaluate the effectiveness of both methods separately. Table IV shows the number of nets that were improved (balanced) as a result of solely the global placement stage and the global stage in conjunction with the detailed stage.

As can be seen, our global method improves fork balancing for more than half of nets in three of the four benchmarks compared to the commercial baseline. Running the detailed stage afterwards further improves overall fork balancing in those three benchmarks. However, in the fetch unit, the number of forks improved decreases after detailed placement. This is



(a) Logic (b) Shift (c) Decode (d) Fetch

Fig. 3: Benchmarks evaluated on A-NTUPLACE with net weights derived from the timing model and no fork balancing.

TABLE IV: Balanced Isochronic Forks

Benchmark	Total Forks	Improved Global	Improved Detailed
Logic	71	52 %	62 %
Shift	174	70 %	76 %
Decode	54	65 %	76 %
Fetch	184	46 %	42 %

likely caused by an interleaved placement of modules from two distinct but tightly coupled components within the benchmark, which makes it difficult for the detailed placer to improve on HPWL *and* balance forks. To fully understand the impact of this improvement, we plot the magnitude of the imbalances in wire forks. In Figure 4, each point represents a net e within the potentially-isochoric fork set E_I. The X-axis and Y-axis represent the difference in distances between the modules closest to and furthest from the net driver for the baseline and the evaluated scheme, respectively. Red circles indicate a net with large baseline HPWL, and blue circles indicate a small baseline HPWL. A reference line $y = x$ is added; points below the reference line indicate an improvement, i.e., a more balanced fork.

As can be seen, all benchmarks have many blue points clustered around the origin. This indicates that many of the nets in set E_I are relatively short, and the penalty function does not have a large effect on them. Since they are so short, the imbalances in these nets are unlikely to manifest as timing errors. However, nets represented by dark red points are longer more likely to violate the isochronic fork assumption if not well balanced. Fortunately, the A-NTUPLACE does well at balancing these forks and improving over the baseline, as most of these red points fall below the reference line.

Finally, we evaluate how well our timing driven placement strategy works in conjunction with our fork balancing scheme. Our benchmarks are run through A-NTUPLACE using the net weights derived from the timing model. We also enable fork balancing in both the global and detailed placement stages. The results are shown in Figure 5.

As can be seen, adding fork bounding constraints to the placement does have a small effect on the overall distribution of net HPWL. Compared to Figure 3, the points trend slightly upwards, above the reference line. This indicates that the forces minimizing the wirelength and balancing wire forks are in opposition. This is expected, since, as in (6), fork imbalances add a penalty to the function being minimized. However, we observe that there is no significant difference in terms of fork balancing with net weights compared to without net weight, as in Figure 4. This indicates that fork balancing is less sensitive to varying net weights. This too is expected, since the fork balancing penalty formulation in (6) only takes distances into account, not the net weights.

V. Conclusion

We presented A-NTUPLACE, a timing-driven placement tool designed specifically to handle quasi delay-insensitive asynchronous circuits. Our timing engine, based on repetitive event rule systems, is general and can identify critical timing paths through any stable and non-interfering production rule set. Our cell placer is capable of prioritizing those nets deemed critical by our timing model, and is uniquely formulated to handle the isochronic fork timing assumption. Our main contributions are being able to balance wire forks while still respecting net criticality as derived by our timing model. Our results indicate that our placer can improve fork imbalances in our benchmarks by an average of 64 % compared to a baseline commercial place & route tool.

Acknowledgment

This work has been supported in part by IARPA award N66001-12-C-2009, NSF award CCF-1065307. We'd like to thank Emmett for his impeccable timing.

References

[1] P. A. Beerel, G. D. Dimou, and A. M. Lines. Proteus: An ASIC flow for GHz asynchronous designs. *IEEE Des. Test. Comput.*, 2011.

[2] S. M. Burns. *Performance analysis and optimization of asynchronous circuits*. PhD thesis, California Institute of Technology, 1991.

[3] T.C. Chen, Z.W. Jiang, T.C. Hsu, H.C. Chen, and Y.W. Chang. NTUplace3: An analytical placer for large-scale mixed-size designs with preplaced blocks and density constraints. *TCAD*, 2008.

[4] J. Cortadella, M. Kishinevsky, A. Kondratyev, L. Lavagno, and A. Yakovlev. Petrify: a tool for manipulating concurrent specifications and synthesis of asynchronous controllers. *Transactions on Information and Systems*, 1997.

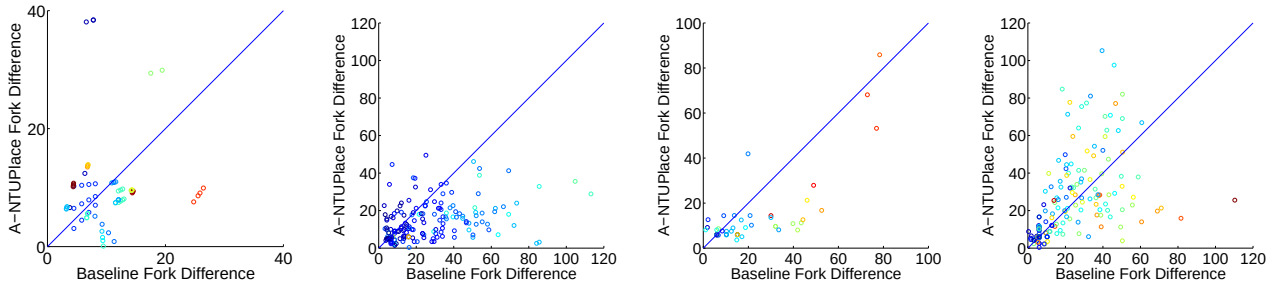
[5] A. E. Dunlop, V. D. Agrawal, D. N. Deutsch, M. F. Jukl, P. Kozak, and M. Wiesel. Chip layout optimization using critical path weighting. In *DAC*, 1984.

[6] C. M. Fiduccia and R. M. Mattheyses. A linear-time heuristic for improving network partitions. In *DAC*. IEEE, 1982.

[7] H. Hulgaard, S. M. Burns, T. Amon, and G. Borriello. An algorithm for exact bounds on the time separation of events in concurrent systems. *IEEE Transactions on Computers*, 1995.

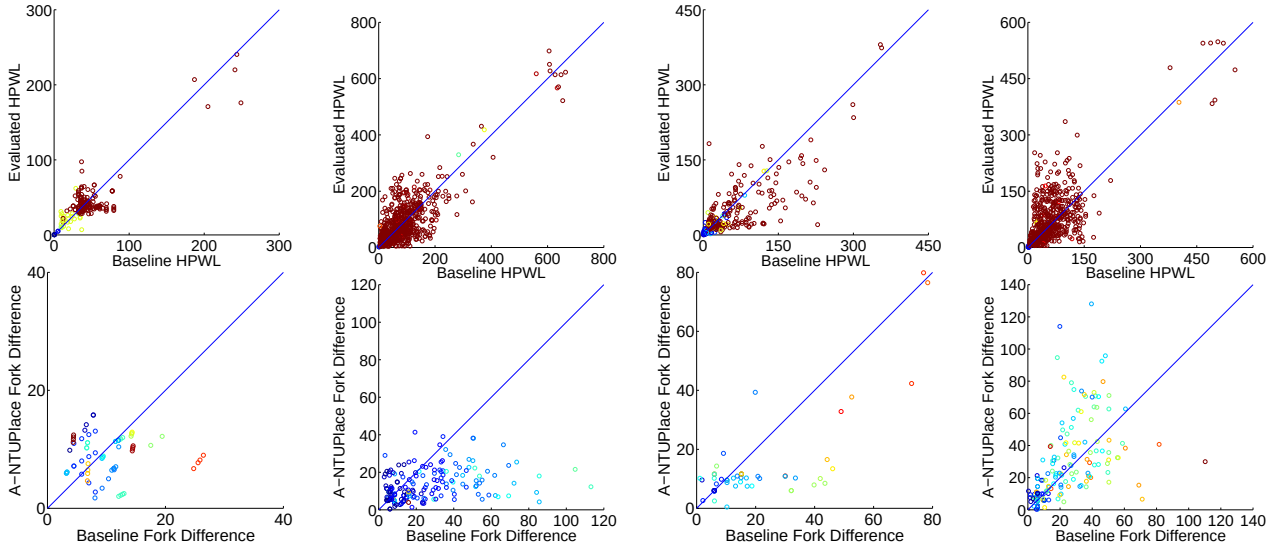
[8] A. B. Kahng and Q. Wang. Implementation and extensibility of an analytic placer. *TCAD*, 2005.

[9] R. Karmazin, C. Ortega-Otero, and R Manohar. celltk: Automated layout for asynchronous circuits with nonstandard cells. In *ASYNC*, 2013.



(a) Logic (b) Shift (c) Decode (d) Fetch

Fig. 4: Benchmarks evaluated on A-NTUPLACE with unit net weights and fork balancing with both global and detail stages.



(a) Logic (b) Shift (c) Decode (d) Fetch

Fig. 5: Benchmarks evaluated on A-NTUPLACE with derived net weights and global and detailed fork balancing techniques.

[10] J. M. Kleinhans, G. Sigl, F. M. Johannes, and K. J. Antreich. GORDIAN: VLSI placement by quadratic programming and slicing optimization. *TCAD*, 1991.

[11] E. Kounalakis and C. P. Sotiriou. CPlace: A Constructive Placer for Synchronous and Asynchronous Circuits. In *ASYNC*, 2011.

[12] T. K. Lee. *A General Approach to Performance Analysis and Optimization of Asynchronous Circuits*. PhD thesis, California Institute of Technology, 1995.

[13] A. M. Lines. Pipelined asynchronous circuits. 1998.

[14] S. J. Longfield and R. Manohar. Inverting martin synthesis for verification. In *ASYNC*, 2013.

[15] N. Maheshwari and S. S. Sapatnekar. *Timing analysis and optimization of sequential circuits*. Springer, 1999.

[16] R. Manohar and Y. Moses. Analyzing isochronic forks with potential causality. In *ASYNC*, 2015.

[17] A. Marquardt, V. Betz, and J. Rose. Timing-driven placement for FPGAs. In *FPGA*, 2000.

[18] A. J. Martin. Compiling Communicating Processes for Delay-Insensitive VLSI Circuits. *Distributed Computing*, 1986.

[19] A. J. Martin. Compiling communicating processes into delay-insensitive VLSI circuits. *Distributed computing*, 1986.

[20] P. B. McGee and S. M. Nowick. An efficient algorithm for time separation of events in concurrent systems. In *ICCAD*, 2007.

[21] T. Murata. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, 1989.

[22] M. Najibi and P. A. Beerel. Deriving performance bounds for conditional asynchronous circuits using linear programing. In *ASYNC*, 2013.

[23] W. C. Naylor, R. Donelly, and L. Sha. Non-linear optimization system and method for wire length and delay optimization for an automatic electric circuit placer, 2001. US Patent 6,301,693.

[24] C. D. Nielsen and M. Kishinevsky. Performance analysis based on timing simulation. In *DAC*, 1994.

[25] C. Ortega-Otero, J. Tse, R. Karmazin, B. Hill, and R Manohar. ULSNAP: An ultra-low power event-driven microcontroller for sensor network nodes. In *ISQED*. IEEE, 2014.

[26] D. Z. Pan, B. Halpin, and H. Ren. Timing-driven placement. *Handbook of Algorithms for VLSI Physical Automation*, 2007.

[27] D. Roy, J.and Papa, S. Adya, H. Chan, A. Ng, J. Lu, and I. Markov. Capo: robust and scalable open-source min-cut floorplacer. In *ISPD*, 2005.

[28] C. Sechen and A. Sangiovanni-Vincentelli. The TimberWolf placement and routing package. *JSSC*, 1985.

[29] A. Smirnov, A. Taubin, M. Su, and M. Karpovsky. An automated fine-grain pipelining using domino style asynchronous library. In *ACSD*, 2005.

[30] P. Spindler, U. Schlichtmann, and F. M. Johannes. Kraftwerk2a fast force-directed quadratic placement approach using an accurate net model. *TCAD*, 2008.

[31] Y. Thonnart, E. Beigne, and P. Vivet. A pseudo-synchronous implementation flow for WCHB QDI asynchronous circuits. In *ASYNC*, 2012.

[32] N. Viswanathan and C. Chu. FastPlace: Efficient analytical placement using cell shifting, iterative local refinement, and a hybrid net model. *TCAD*, 2005.

[33] N. Viswanathan, G.J. Nam, C. J. Alpert, P. Villarrubia, H. Ren, and C. Chu. RQL: global placement via relaxed quadratic spreading and linearization. In *DAC*, 2007.

[34] M. Wang, X. Yang, and M. Sarrafzadeh. Dragon2000: Standard-cell Placement Tool for Large Industry Circuits. In *ICCAD*, 2000.

[35] G. Wu, T. Lin, H-H Huang, C. Chu, and P. A. Beerel. Asynchronous circuit placement by lagrangian relaxation. In *ICCAD*, 2014.

[36] A. Ziesemer Jr, R. Reis, M. Moreira, M. Arendt, and N. Calazans. A design flow for physical synthesis of digital cells with ASTRAN. In *GLSVLSI*. ACM, 2014.