Gate Sizing and Vth Assignment for Asynchronous Circuits Using Lagrangian Relaxation

Gang Wu, Ankur Sharma and Chris Chu
Department of Electrical and Computer Engineering, Iowa State University, IA
Email: {gangwu, ankur, cnchu}@iastate.edu

***Abstract*—Gate sizing and threshold voltage selection is an important step in the VLSI physical design process to help reduce power consumption and improve circuit performance. Recent asynchronous design flows try to directly leverage synchronous EDA tools to select gates, which have a lot of limitations due to the intrinsic difference between asynchronous and synchronous circuits. This paper presents a new simultaneous gate sizing and Vth assignment approach for asynchronous designs. We formulate the asynchronous gate version selection problem considering both leakage power consumption and cycle time. Then, the optimization is performed based on a Lagrangian relaxation framework. A fast and effective slew updating strategy is also proposed to address the timing-loops of asynchronous circuits during static timing analysis. Our approach is evaluated using a set of asynchronous designs based on the pre-charged half buffer (PCHB) template and compared with the Proteus asynchronous design flow which is leveraging synchronous EDA tools. The experiments show our approach can achieve much better quality results in terms of both leakage power and cycle time compared with the other approach.**

I. Introduction

As the feature size of advanced fabrication process is down to nanometer scale, the design of synchronous circuit is facing more and more issues such as process variation and power consumption. Asynchronous design provides a very attractive alternative to synchronous design due to its robustness, lower power consumption and higher operating speed. Its advantages have been demonstrated by many fabricated chips [1] [2] [3]. However, asynchronous design is still not widely adopted in the industry because of its long learning curve and the lack of asynchronous EDA tools.

Gate sizing and V_{th} assignment have been shown to be very effective techniques to optimize the power and performance of synchronous circuits. We expect these techniques to have similar impact on asynchronous circuits. In particular, we want to minimize the leakage power, as in advanced process it contributes to a large part of total power consumption which has become an important design objective nowadays due to the limited battery life of the widely used portable devices. For asynchronous design, leakage power minimization can be even more critical because of its higher gate count than synchronous design. Therefore, in this paper, we are focusing on the problem of leakage power and cycle time minimization for asynchronous design by selecting gates of different sizes and V_{th} from a standard cell library.

Both gate sizing and V_{th} assignment techniques for synchronous circuits have been extensively studied for decades [4] [5] [6] [7]. However, for asynchronous circuits, there are only very few works on it. Most of the automatic synthesis flows for asynchronous circuits try to directly leverage synchronous EDA tools [8] [9]. As the circuit structure, performance metric and timing constraints for asynchronous circuits are quite different from those for synchronous circuits, these approaches require to break the timing-loops and add explicit timing constraints the number of which is exponential to the circuit size. For large scale designs, the complicated timing constraints are beyond the ability of synchronous EDA tools to handle thus inferior results are generated. In [10], a genetic algorithm based simultaneous gate sizing and V_{th} assignment technique specific for asynchronous circuits has been proposed to minimize the leakage power while maintaining the performance requirements. However, genetic algorithms usually have long runtime and are not scalable, which makes it unsuitable for large scale circuits.

A fast and accurate static timing analysis (STA) method is essential to guide the gate selection algorithm to achieve a good solution within a short amount of runtime. For synchronous circuits, this can be done by a simple graph traversal as the corresponding combinational logic network can be represented as a directed acyclic graph (DAG). However, for asynchronous circuits, the way to perform STA is not straightforward due to its more general circuit structure which might contain internal combinational loops. In [11], a STA flow on pre-charged half buffer (PCHB) and Multi-Level Domino (MLD) templates has been proposed, which leverages a commercialized synchronous timing analyzer. However, this approach is limited to template based designs as automatically finding the cut points requires a regular circuit structure. Also, the achieved timing value is not accurate as time borrowing across the broken segments is not allowed.

This paper presents a new simultaneous gate sizing and V_{th} assignment approach for asynchronous circuits. We formulate the gate selection problem to minimize both the leakage power and cycle time while satisfying various type of asynchronous timing constraints. A Lagrangian relaxation framework is applied on the formulated problem to transform it into a sequence of Lagrangian relaxation subproblems (LRS). In particular, the arrival time based linear constraints allow us to simplify LRS using Karush-Kuhn-Tucker (KKT) conditions [12]. This simplified LRS can then be easily solved using an effective greedy algorithm. The proposed gate selection approach considers discrete cell sizes and threshold voltages from the standard cell library, and is implemented based on the accurate non-linear delay model (NLDM). In addition, to

This work is supported in part by NSF award CCF-1219100.

overcome the obstacle of STA for asynchronous circuits, we propose an iterative slew update algorithm which is accurate and guarantees fast convergence with library-based timing models.

We evaluate our flow using a set of asynchronous designs based on the PCHB templates and compare it with a latest asynchronous design flow Proteus [8]. Our flow is shown to be consistently better and we have achieved significant improvements in both the cycle time and leakage power. Our approach is more effective than those which twist and trick synchronous EDA tools to generate a functional circuit as it directly handles timing loops, which means time borrowing along the loop is allowed and the number of constraints is polynomial in circuit size.

The rest of this paper is organized as follows. In Section II, we give an overview about the synchronous gate selection techniques. Several timing issues related to asynchronous circuits are discussed and the gate selection problem is then formulated. In Section III, a Lagrangian relaxation based approach is presented to solve the gate selection problem. In Section IV, we discuss the proposed STA approach for asynchronous circuit. In Section V, we summarize the implemented asynchronous gate selection flow. Finally, the experimental results are shown in Section VI.

II. Preliminaries

A. Gate Selection Techniques for Synchronous Circuits

The discrete gate sizing problem is proved to be NP-hard [13]. Therefore, various heuristic algorithms like convex programming [14], sensitivity based algorithms [15] and so on have been proposed by researchers to assign proper sizes and threshold voltages for gates in synchronous circuits. There are even organized gate sizing contests [16] [17] to help expose the challenges faced in modern industrial designs to the academic field. One powerful heuristic approach adopted by leading synchronous gate selection algorithms [18] [19] is to apply the Lagrangian relaxation (LR) technique. In [6], foundations for LR-based gate sizing approach is first established, which considers continuous sizing and simple delay models. The LR-based approach is then continuously got improved as people are combining it with library-based timing model, discrete gate sizing, V_{th} assignment, dynamic programming and network flow algorithms [18] [19] [20] [21]. The advantage of LR-based approach is that it can be easily modified to handle different objectives and various complex design constraints. Even though convexity cannot be claimed for the discrete gate sizing problem, the LR-based approach is shown to have fast convergence in practice and is practical to large scale problems. Although extensive research has been done for the LR-based gate selection algorithms of synchronous circuits, whether it is applicable and effective to asynchronous circuits is still not being explored.

B. Full Buffer Channel Net Model

Here we use the Full Buffer Channel Net (FBCN) [22] to model our asynchronous circuits. A FBCN is a specific form of timed marked graph. The idea is to model each leaf cell as a *transition* and asynchronous channels between cell ports are modeled with a pair of *places* which are annotated with delay information. Please note that here we treat the asynchronous circuits as unconditional. For conditional asynchronous circuits modeled using FBCN, the circuit performance can be guaranteed conservatively as proved in [23].

As an example, a simple ALU design is shown in Fig. 1 and the corresponding marked graph based on the FBCN model is shown in Fig. 2. Here, t_{mul1} and t_{mul2} represent the two-stage multiplication cells and t_{add} represents the addition cell. All the places are denoted as circles. In particular, places containing tokens are represented by circles marked with a black dot. Two channels on the left are assumed to be in the full state and have tokens assigned on the forward places. The rest of the channels are assumed to be empty and have tokens assigned on the backward places.

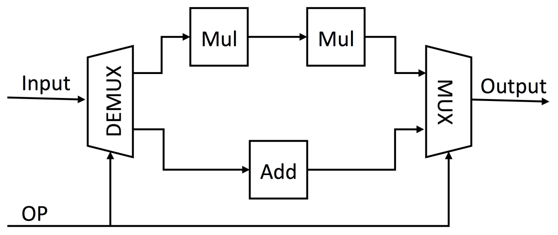

Fig. 1: Asynchronous ALU.

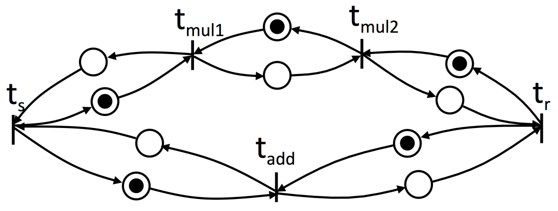

Fig. 2: Marked graph representation for Asynchronous ALU.

C. Asynchronous Performance Analysis

There are many existing algorithms which are able to capture the cycle metric of asynchronous circuits. In this paper, we adopt a linear programming based approach [24], as it can be easily incorporated in to our Lagrangian relaxation framework.

For any asynchronous circuit modeled with FBCN, the cycle time τ can be obtained by solving the following linear program:

$$\begin{aligned} &\text{Minimize} \quad \tau \\ &\text{Subject to} \quad a_i + D_{ij} - m_{ij}\tau \le a_j \quad \forall (i,j) \end{aligned}$$

where a_i and a_j are the arrival time associated with transitions t_i and t_j. D_{ij} is the delay associated with a place p between two neighboring transitions t_i and t_j. $m_{ij} = 1$ if the corresponding place p contains a token and 0 otherwise.

Please note that in practice, there can be multiple delay values such as the rising and falling delay associate with a place p, The different delay values can be considered by simply extending the existing constraints. Here we ignore these details in order to make our presentation more concise.

D. Asynchronous Timing Constraints

Compared with synchronous circuits which only have setup and hold time constraints, asynchronous circuits can have some totally different timing constraints depending on the timing assumptions made by the specific asynchronous logic implementation style. For the bounded-delay asynchronous designs [25], two-sided timing constraints need to be enforced which require both a minimum and maximum allowable delay of a specified gate or wire. Instead, for the quasi-delay-insensitive (QDI) design style such as WCHB, PCHB, MLD template, relative delay constraints referred to as *relative timing* [26] need to be enforced, which dictate the relative delay of two paths that stem from a common point of divergence. These two design categories cover most of the existing timing constraints specific to asynchronous circuits. We will show how to incorporate these constraints into our problem formulation in Sec. II E.

E. Asynchronous Gate Sizing and Vth Assignment

In this subsection, we formulate the asynchronous gate selection problem considering the performance and timing constraints. For an asynchronous circuit modeled with FBCN, let T be the set of transitions and P be the set of places in the timed marked graph. In particular, we use $p(i,j)$ to denote the place between neighboring transitions t_i and t_j. Let $\boldsymbol{a} = \{a_1, a_2, ..., a_{|T|}\}$ be the set of arrival times corresponding to T. Let $\boldsymbol{g} = \{g_1, g_2, ..., g_{|T|}\}$ be the set of gates corresponding to T and we use v_i^j to represent a specific selected version j for gate g_i. Let $\boldsymbol{g_0}$ be the initial set of selected gates before optimization and τ_0 be its corresponding cycle time. In addition, we use P_b to denote the set of places annotated with two-sided delay bounds. We use P_{rt} to denote the set of places annotated with relative timing constraints. Then the problem of minimizing both total leakage power consumption and cycle time subject to timing constraints can be formulated as:

$$\begin{aligned} &\text{Minimize} \quad \text{leakage}(\boldsymbol{g})/\text{leakage}(\boldsymbol{g_0}) + \alpha\tau/\tau_0 \\ &\text{Subject to} \quad a_i + D_{ij} - m_{ij}\tau \le a_j \qquad \forall\, p(i,j) \in P \qquad (1) \\ &\qquad L_{ij} \le a_j - a_i \le U_{ij} \qquad \forall\, p(i,j) \in P_b \qquad (2) \\ &\qquad |(a_i - a_k) - (a_j - a_k)| \le I_{ij}\ \forall\, p(i,j) \in P_{rt} \quad (3) \end{aligned}$$

where the constant α can be chosen to adjust the tradeoff between minimizing the normalized leakage power and the normalized cycle time. L_{ij} and U_{ij} denote the minimum and maximum bounded delay. I_{ij} denotes the relative delay stemming from transition t_k and forking into two transitions t_i and t_j. leakage($\boldsymbol{g}$) captures the summation of leakage power for the set of gates $\boldsymbol{g}$ with selected versions.

We can rewrite the timing constraints in Equations (2) and (3) into the same form with the performance constraints in Equation (1) as follows:

$$(a_i + L_{ij} \le a_j) \wedge (a_j - U_{ij} \le a_i) \qquad (4)$$

$$(a_j - I_{ij} \le a_i) \wedge (a_i - I_{ij} \le a_j) \qquad (5)$$

Then, combining Equation (1) with the reformulated Equations (4) and (5), we can get a more concise representation of our primal problem:

$$\begin{aligned} \mathcal{PP}: \quad &\text{Minimize} \quad \text{leakage}(\boldsymbol{g})/\text{leakage}(\boldsymbol{g_0}) + \alpha\tau/\tau_0 \\ &\text{Subject to} \quad a_i + \hat{D}_{ij} - \hat{m}_{ij}\tau \le a_j \quad \forall(i,j) \end{aligned}$$

where $\hat{D}_{ij}$ represents D_{ij}, L_{ij}, $-U_{ij}$ or $-I_{ij}$ depending on the corresponding places annotated with performance or timing constraints. Also, we have $\hat{m}_{ij} = m_{ij}$ for all performance constraints and $\hat{m}_{ij} = 0$ for all timing constraints. $\forall(i,j)$ represents all the i, j pairs corresponding to $p(i,j) \in P \cup P_b \cup P_{rt}$.

III. Simultaneous Gate Sizing and Vth Assignment by Lagrangian Relaxation

In this section, we propose our LR-based approach to solve the formulated asynchronous gate selection problem $\mathcal{PP}$. In Sec. III A, the Lagrangian relaxation subproblem ($\mathcal{LRS}$) which provides a lower bound to the solution of $\mathcal{PP}$ is obtained by applying LR technique to $\mathcal{PP}$. In Sec. III B, we first simplify $\mathcal{LRS}$ using applying KKT conditions. Then, an effective greedy algorithm is proposed Sec. III C to solve this simplified $\mathcal{LRS}$. In Sec. III D, we solve the Lagrangian dual problem ($\mathcal{LDP}$) to achieve a solution of $\mathcal{PP}$ by iteratively solving a sequence of simplified $\mathcal{LRS}$.

A. Lagrangian Relaxation Subproblem ($\mathcal{LRS}$)

First, we apply Lagrangian relaxation to the primal problem. We attach a set of nonnegative Lagrangian multipliers $\boldsymbol{\lambda} = \{\lambda_{ij} \mid \forall(i,j)\}$ to all the constraints in $\mathcal{PP}$ and relax these constraints into the objective function. Then the Lagrangian relaxation subproblem we get is:

$$\begin{aligned} \mathcal{LRS}: \quad &\text{Mimimize} \quad \text{leakage}(\boldsymbol{g})/\text{leakage}(\boldsymbol{g_0}) + \alpha\tau/\tau_0 \\ &\qquad + \sum_{\forall(i,j)} \lambda_{ij}(a_i + \hat{D}_{ij} - \hat{m}_{ij}\tau - a_j) \end{aligned}$$

The relaxed problem becomes an unconstrained optimization problem. Please note that the variables we have for $\mathcal{LRS}$ are $\boldsymbol{g}$, $\boldsymbol{a}$ and τ while $\boldsymbol{\lambda}$ is a given parameter. For any given set of $\boldsymbol{\lambda} \ge \boldsymbol{0}$, solving $\mathcal{LRS}$ will provide a lower bound to the optimal solution of $\mathcal{PP}$ [12].

B. Simplified Lagrangian Relaxation Subproblem ($\mathcal{LRS}^*$)

Similar to [6], we rearrange terms here and the $\mathcal{LRS}$ can be rewritten as:

$$\begin{aligned} \text{Mimimize} \quad &\text{leakage}(\boldsymbol{g})/\text{leakage}(\boldsymbol{g_0}) + (\alpha - \sum_{\forall(i,j)} \lambda_{ij}\hat{m}_{ij})\tau/\tau_0 \\ &+ \sum_{k \in T}(\sum_{\forall(k,j)} \lambda_{kj} - \sum_{\forall(i,k)} \lambda_{ik})a_k \\ &+ \sum_{\forall(i,j)} \lambda_{ij}\hat{D}_{ij} \end{aligned}$$

The idea behind this rearrangement is to group all the coefficients associated with cycle time variable τ and arrival time variables $\boldsymbol{a}$, which make them easier to be removed as we will show in the next step.

Let $\mathcal{L}(\boldsymbol{g}, \boldsymbol{a}, \tau)$ be the objective function of $\mathcal{LRS}$. The KKT stationarity conditions imply $\partial\mathcal{L}/\partial a_i = 0$ for $1 \leq i \leq |T|$ and $\partial\mathcal{L}/\partial\tau = 0$ at the optimal solution of the primal problem. Then we can get the following optimality conditions:

$$\mathcal{KKT}: \qquad \alpha = \sum_{\forall(i,j)} \lambda_{ij}\hat{m}_{ij}$$
$$\sum_{\forall(k,j)} \lambda_{kj} = \sum_{\forall(i,k)} \lambda_{ik} \quad \forall\, k \in T$$

Apply the optimality conditions into $\mathcal{LRS}$, we can obtain a simplified Lagrangian relaxation subproblem as follows:

$$\mathcal{LRS}^*: \quad \text{Minimize} \quad \text{leakage}(\boldsymbol{g})/\text{leakage}(\boldsymbol{g_0}) + \sum_{\forall(i,j)} \lambda_{ij}\hat{D}_{ij}$$

After the simplification, variables τ and $\boldsymbol{a}$ are removed. The only variables left are the set of selected gates $\boldsymbol{g}$ associated with transitions. It can be seen that $\mathcal{LRS}^*$ is equivalent to $\mathcal{LRS}$ and it is much easier to solve.

C. Solving $\mathcal{LRS}^*$

Algorithm 1 shows our algorithm to solve $\mathcal{LRS}^*$. First, we assign an initial version to each gate and insert all the gates into a set $\mathcal{G}$. Then, we pick any gate g_i from $\mathcal{G}$ and use Algorithm 2 to find a better version for it, i.e., picking a different size or threshold voltage for that gate. If the selected new version v_i^k of g_i is different from its old version v_i^j, we assign v_i^k to the gate and insert all its fanout gates not in $\mathcal{G}$ into $\mathcal{G}$. Otherwise, we do not reevaluate its downstream cells if they are not in $\mathcal{G}$. In particular, if the gate has been visited more than a certain number (n) of times, we also do not reevaluate its downstream cells in order to save runtime. The algorithm terminates when $\mathcal{G}$ is empty.

Algorithm 1 Solve $\mathcal{LRS}^*$

```
Ensure: a proper version for each gate which minimize $\mathcal{LRS}^*$
 1: Initially assign all the gates with a version;
 2: Insert all the gates into a set $\mathcal{G}$;
 3: while $\mathcal{G} \neq \emptyset$ do
 4:     Pick one gate $g_i$ from $\mathcal{G}$. Let its current version be $v_i^j$;
 5:     Select a new version $v_i^k$ for gate $g_i$; /* Algorithm 2 */
 6:     if $v_i^j \neq v_i^k$ then
 7:         Assign $g_i$ with this new version $v_i^k$;
 8:         if $g_i$ is visited less than or equal to $n$ times then
 9:             Insert all gates $\notin \mathcal{G}$ and directly driven by $g_i$ into $\mathcal{G}$;
10:         end if
11:     end if
12:     Remove $g_i$ from set $\mathcal{G}$;
13: end while
```

The algorithm to select a new gate version is shown in Algorithm 2. For each possible version v_i^j of a specific gate g_i, we first update the gate to this new version.

Next, we need to estimate the timing impact made by this gate version change. Instead of doing STA for the entire circuit which can be very time consuming, we perform a local timing update here. In particular, we update the output load of all the fanin gates ($fanin(g_i)$) which is driving g_i and the input slew of all the fanout gates ($fanout(g_i)$) which is driven by g_i. We also update the input slew of all the side gates ($side(g_i)$) which are defined as all the gates driven by $fanin(g_i)$ except g_i. Then we recompute the delay for all the timing arcs associated with g_i and its fanin, fanout and side gates.

Let Arc_i = timingArcs($g_i \cup fanin(g_i) \cup fanout(g_i) \cup side(g_i)$) be the set of updated timing arcs. After the local timing update, we can evaluate the cost to objective value of $\mathcal{LRS}^*$ as follows:

$$\text{Cost}(g_i) = \text{leakage}(g_i) + \sum_{(u,v)\in Arc_i} \lambda_{uv}\hat{D}_{uv}$$

After all the possible options for the current gate have been evaluated, the one providing the minimum cost will be returned as the best choice.

Algorithm 2 Gate Version Selection

```
Ensure: Best version for the gate $g_i$ which minimize $\mathcal{LRS}^*$
 1: for each available option $v_i^j$ for the gate $g_i$ do
 2:     Assign $v_i^j$ to $g_i$;
 3:     Local timing update;
 4:     if Cost($g_i$) < bestCost then
 5:         bestVersion = $v_i^j$;
 6:         bestCost = Cost($g_i$);
 7:     end if
 8: end for
 9: return bestVersion;
```

D. Lagrangian Dual Problem ($\mathcal{LDP}$)

In Sec. III A, we mention that a lower bound to the optimal solution of $\mathcal{PP}$ can be obtained by solving $\mathcal{LRS}$ for any given set of $\boldsymbol{\lambda} \geq \boldsymbol{0}$. Now we discuss how to find the specific $\boldsymbol{\lambda}$ that gives us the maximum (i.e., tightest) lower bound. It is formulated as the Lagrangian dual problem as follows:

$$\mathcal{LDP}: \quad \text{Maximize} \quad \mathcal{LRS}$$
$$\text{Subject to} \quad \boldsymbol{\lambda} \geq \boldsymbol{0}$$

Please note that $\boldsymbol{g}$, $\boldsymbol{a}$, τ along with $\boldsymbol{\lambda}$ are all variables for the Lagrangian dual problem. Solving $\mathcal{LDP}$ will provide the best solution to the primal problem.

Instead of maximizing $\mathcal{LRS}$, we want to incorporate the optimality conditions and maximize the equivalent yet simpler problem $\mathcal{LRS}^*$. Thus, the Lagrangian dual problem can be rewritten as:

$$\text{Maximize} \quad \mathcal{LRS}^*$$
$$\text{Subject to} \quad \boldsymbol{\lambda} \geq \boldsymbol{0},\ \boldsymbol{\lambda} \in \mathcal{KKT}$$

E. Solving $\mathcal{LDP}$

$\mathcal{LDP}$ can be solved by iteratively solving a sequence of $\mathcal{LRS}^*$. A commonly used strategy is the subgradient optimization method [12]. However, this method requires a projection for $\boldsymbol{\lambda}$ after each iteration in order to maintain $\boldsymbol{\lambda}$ within the feasible region of $\mathcal{LDP}$. For synchronous circuits, this can be done by simply traversing the circuit in topological order. For asynchronous circuits, it will not be easy to redistribute $\boldsymbol{\lambda}$ as the corresponding circuit structure contains loops. In addition, achieving a good convergence using this subgradient optimization method is difficult and usually requires a careful choice of initial solution and step size.

To resolve these issues, we apply a direction finding approach inspired by [21] to solve $\mathcal{LDP}$, which is shown to have better convergence and no projection is needed.

Let $q(\boldsymbol{\lambda})$ denotes the optimal objective value of $\mathcal{LRS}^*$ for a given set of $\boldsymbol{\lambda}$. The direction finding approach wants to find an improving feasible direction $\Delta\boldsymbol{\lambda}$ and a step size β such that at each step we have:

$$q(\boldsymbol{\lambda} + \beta\Delta\boldsymbol{\lambda}) > q(\boldsymbol{\lambda})$$

In particular, the improving feasible direction $\Delta\boldsymbol{\lambda}$ can be found by solving the following linear program:

$$\begin{aligned} \mathcal{DF}: \quad & \text{Maximize} && \sum_{\forall(i,j)} \Delta\lambda_{ij}\hat{D}_{ij} \\ & \text{Subject to} && \boldsymbol{\lambda} \geq \mathbf{0},\ \boldsymbol{\lambda} \in \mathcal{KKT} \\ &&& \max(-u, -\lambda_{ij}) \leq \Delta\lambda_{ij} \leq u \end{aligned}$$

where u is used to bound the objective function and avoid it goes to infinity, similar to [21].

After we find the improving feasible direction, the step size β can be obtained by optimizing along this direction using any line search technique.

The detailed algorithm to solve $\mathcal{LDP}$ is presented in Algorithm 3. It starts from an initial dual feasible $\boldsymbol{\lambda}$, which is non-negative and satisfies the optimality conditions. Then the method iteratively improves $q(\boldsymbol{\lambda})$ by finding an improving direction and performing a line search to find the best step size. The algorithm terminates when $q(\boldsymbol{\lambda})$ is not improving or the total number of iterations exceeds the limit.

Algorithm 3 Solve $\mathcal{LDP}$

```
Ensure: $\boldsymbol{\lambda}$ which maximizes $\mathcal{LRS}^*$
 1: $n = 1$; /* loop counter */
 2: $\boldsymbol{\lambda} =$ initial non-negative value satisfy optimality conditions;
 3: while $n < limit$ do
 4:     Solve $\mathcal{DF}$ to obtain improving direction $\Delta\boldsymbol{\lambda}$;
 5:     while line search not terminate do
 6:         Compute $\beta$ based on specific line search technique;
 7:         $\boldsymbol{\lambda}' = \boldsymbol{\lambda} + \beta\Delta\boldsymbol{\lambda}$;
 8:         Solve $\mathcal{LRS}^*$ to obtain $q(\boldsymbol{\lambda}')$;
 9:         if $q(\boldsymbol{\lambda}') > bestObj$ then
10:             $bestStep = \beta$;
11:             $bestObj = q(\boldsymbol{\lambda}')$;
12:         end if
13:     end while
14:     if $bestObj \leq q(\boldsymbol{\lambda})$ then   /* $q(\boldsymbol{\lambda})$ is not improving */
15:         exit loop;
16:     end if
17:     $\boldsymbol{\lambda} = \boldsymbol{\lambda} + bestStep * \Delta\boldsymbol{\lambda}$; /* move one step further */
18:     Solve $\mathcal{LRS}^*$;
19:     $n = n + 1$;
20: end while
```

IV. Static Timing Analysis for Asynchronous Circuits

An efficient asynchronous STA method is necessary for us to compute the delay values and cycle time τ using library-based timing model. In order to do the STA, we first find the output slew values of each gate using an iterative slew rate update approach described in Sec. IV A. Then, $\hat{D}_{ij}$ can be achieved by lookup table interpolation in the same manner as synchronous STA. Finally, the cycle time can be computed using the linear program described in Sec. II C.

A. Iterative Slew Update Approach

The algorithm to implement the iterative slew update approach is presented as Algorithm 4. It is similar to Algorithm 1 which solves $\mathcal{LRS}^*$. Here, we also keep a set $\mathcal{G}$ of gates. A gate is in $\mathcal{G}$ if its current output slew is potentially *inconsistent* with its current input slews. In particular, we define an output slew to be *inconsistent*, if the resulting output slew might be larger than the current one when we evaluate it based on the current input slews. We define it to be "larger than" as we are trying to find an upper bound of all the slews.

In the beginning, we initialize the output slew of all gates to 0. Thus, all gates should be in the set $\mathcal{G}$ because all of them are potentially *inconsistent*. In each step, we pick any gate g_i from the set and update its output slew. If the new output slew s_{new} is larger than the current one s_{old}, we update the output slew of g_i to s_{new} and put all gates driven by this gate while not in $\mathcal{G}$ into the set $\mathcal{G}$. When $\mathcal{G}$ is empty, i.e., the output slews of all gates are *consistent*, the algorithm terminates.

Algorithm 4 Iterative Slew Update

```
Ensure: A tight upper bound of the output slew for all the gates;
 1: Initialize the output slew to 0 for all the gates;
 2: Insert all the gates into a set $\mathcal{G}$;
 3: while $\mathcal{G} \neq \emptyset$ do
 4:     Pick one gate $g_i$ from $\mathcal{G}$. Let its current output slew be $s_{old}$;
 5:     Compute new output slew $s_{new}$ of $g_i$ based on its input slew;
 6:     if $s_{new} > s_{old}$ then
 7:         Update the output slew of $g_i$ to $s_{new}$;
 8:         Insert all gates $\notin \mathcal{G}$ and directly driven by $g_i$ into $\mathcal{G}$;
 9:     end if
10:     Remove $g_i$ from set $\mathcal{G}$;
11: end while
```

B. Convergence of the proposed approach

We can easily use induction technique to prove that the output slew of every gate will be monotonically increasing throughout the execution of Algorithm 4. Since all the slew values are upper-bounded, we know the proposed algorithm always converges. Let $\mathcal{S} = \{s_1, ..., s_{|T|}\}$ be the set of slew values computed by the proposed algorithm. Then, we can have the following theorem:

Theorem 1. For each gate, its corresponding output slew value in $\mathcal{S}$ is a tight (i.e., smallest) upper bound of all its output slew values during the operation of the circuit.

Proof: We prove this using contradiction. Assume the slew values computed by Algorithm 4 are not tight, which means there exists gates whose corresponding slew value in $\mathcal{S}$ is larger than its maximum achievable slew value during the circuit operation. Let g_i be the first gate during the iterative slew evaluation process such that its output slew is set to s_i which is larger than its maximum achievable slew value. Based on the slew evaluation process, we know the output slew of g_i is computed based on the output slew of one of its fanin gates g_j. Since s_i is unachievable, the current output slew s_j of gate g_j must also be unachievable. This makes g_i not the first gate the output slew of which is set to an unachievable value, which contradicts our assumption. Therefore, we can conclude that the computed slew in $\mathcal{S}$ is a tight upper bound of all the slew values. □

C. Extension to a tight lower bound

A tight upper bound for all the slew values will guarantee the design satisfies performance constraints as shown in Equation (1). However, for other type of constraints such as the two-sided timing constraints in Equation (2), we might need a tight low bound in order to conservatively satisfy them. This can be achieved by a simple modification of Algorithm 4. Instead of setting all the slew values to 0 in the beginning, here we initialize all of them to the maximum possible slew value in the cell timing library. Then at line 6, we change the condition to $s_{new} < s_{old}$, which makes the output slew of every gate monotonically decrease throughout the execution. Similar to the proof of Theorem 1, the modified algorithm will give us a tight lower bound of all the slew values.

V. Asynchronous Gate Selection Flow

We summarize our flow for asynchronous gate sizing and V_{th} assignment in Fig. 3.

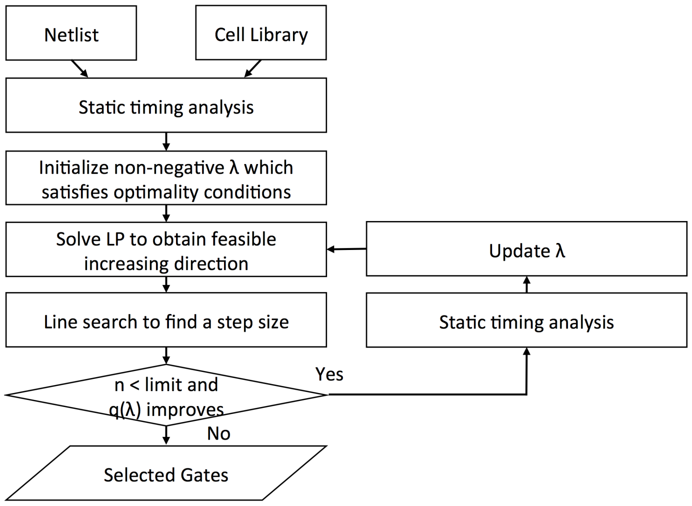
Netlist
Cell Library
Static timing analysis
Initialize non-negative λ which satisfies optimality conditions
Solve LP to obtain feasible increasing direction
Update λ
Line search to find a step size
Static timing analysis
n < limit and q(λ) improves
Yes
No
Selected Gates

Fig. 3: Asynchronous Gate Selection Flow.

The input to our flow is the characterized cell timing library and the netlist. Initially, STA is run for the whole circuit to update the timing and an initial set of $\boldsymbol{\lambda} \geq \mathbf{0}$ satisfying the optimality conditions is found. Next, we enter the loop to solve the $\mathcal{LDP}$ similar to Algorithm 3. The final output of our flow is a gate sizing and V_{th} assignment solution with both cycle time and leakage power being minimized.

VI. Experiments

The proposed gate sizing approach is implemented in C++ and runs on a Linux PC with 8 GB of memory and 2.4 GHz Intel Core i7 CPU.

We are using the Proteus standard cell library [8] which is based on an implementation of the PCHB template. Cell delay and slew values are calculated using the static timing analysis method described in Sec. IV with the accurate non-linear delay model lookup tables from the cell timing library. Cell interconnections are modeled as simple lumped capacitance in our experiment. The lumped capacitance value is obtained from the SPEF file generated by Proteus flow after placement and routing. As the leakage power is not available in Proteus standard cell library, we assign it to be proportional to cell area, which is the same strategy used in ISPD 2013 discrete gate sizing contest benchmarks [17].

In order to show the effectiveness on the power saving after applying the V_{th} assignment techniques, we also extends the Proteus standard cell library into a multi-V_{th} library, as the original library only considers single V_{th}. We scale the leakage power value and look up tables according the ratio between different V_{th} cells in the cell library provided by the ISPD 2013 gate sizing contest. In particular, based on the original cells, we generate a set of low threshold voltage cells with 4X more leakage power but 0.9X smaller delay. Similarly, we generate a set of high threshold voltage cells with 0.25X leakage power but 1.15X larger delay.

We evaluate our approach using two sets of benchmarks. First is a set of asynchronous designs transformed from ISCAS89 benchmarks, which have flip-flops mapped as token buffers and combinational gates mapped as logic cells using Proteus. We also have a set of specific asynchronous designs developed using Verilog and synthesized into gate level netlist using Proteus. Different bit widths are applied to some of the benchmarks to create designs with different sizes.

We need to set the parameter α, which is the tradeoff between leakage power consumption and circuit performance. As our flow starts optimization with all cells assigned with the minimum cell size which have the minimum possible leakage power, we set $\alpha = 2$ to put a similar effort on optimizing power and cycle time of the given circuit. The *limit* for the total number of iterations is set to be 50, which is shown to be able to provide enough improvement within a short amount of runtime. We do not perform any benchmark specific parameter tuning during our experiment. After running the gate selection flow, we run our STA algorithm until convergence to measure the cycle time and the leakage power consumption of our gate selection result.

First, we do the experiment by running our flow using the original single-V_{th} library in order to compare with Proteus. Both flows use the same input netlist starting with cells assigned with minimum size. Comparison results on the transformed ISCAS89 benchmarks are shown in Table I. "Total itr."

column shows the total number iterations performed for each benchmarks until termination. PP_{obj} denotes the achieved objective value for the primal problem. LRS_{obj} denotes the achieved objective value for the $\mathcal{LRS}$. "Gap" column shows the duality gap which helps us to know how close our solution is to the optimal solution. The duality gap here is calculated using $(PP_{obj} - LRS_{obj})/PP_{obj}$. "Init" columns show the initial cycle time and leakage power, which all the cells are assigned with the minimum size. "Proteus" and "Ours" columns show the cycle time and leakage power for gate sizes generated by the Proteus flow and our flow respectively. The results show our approach is consistently better in both cycltime and power consumption for all benchmarks. For the cycle time, on average, our approach is 2.19X better than the initial minimum sized circuit and 21.3% better than the gate sizing results from Proteus. For the leakage power consumption, our approach is 9.5% better than Proteus and only consumes 3.7% more power than the initial minimum sized circuits, which have the minimum possible leakage value. Table II shows the comparison results on the specific asynchronous benchmarks and have achieved similar improvements. Proteus does not have a separate gate sizing step and makes us not able to measure the time it spends only for gate sizing. On average, the runtime of our algorithm is around 5 minutes, which is less than 10% of the total runtime of the entire Proteus flow. This indicates our flow runs fast enough and will not be a runtime bottleneck during the entire design process.

Next, we run our flow using the expanded multi-V_{th} library. The results are shown in the "Multi-Vt" columns in Table I and Table II. In Table I, compared with running our flow on the single-V_{th} library, the V_{th} assignment technique using multi-V_{th} library can save 70.5% of total leakage power consumption with only 2.4% increase on the cycle time. Similar improvement is shown in Table II.

The convergence sequence of our largest circuit s38417 is shown in Fig. 4, which is generated by running the flow on single-V_{th} library without limiting the total number of iterations. The corresponding changes of cycle time and leakage power at each iteration is shown in Fig. 5. It can be seen that our algorithm converges smoothly and the final results are very close to the optimal solution. It also shows that the improvement after 50th iteration is small and suggests that terminating the algorithm earlier can save a large amount of runtime without sacrificing the quality too much.

VII. Conclusions

This paper proposes a gate selection flow for asynchronous circuits. To solve the gate selection problem, we incorporate the linear-programming-based performance evaluation method with the Lagrangian relaxation framework. The relaxed problem is simplified and the $\mathcal{LRS}$ can then be easily solved. We also proposed an iterative slew updating approach for the static timing analysis of asynchronous circuits. Our STA approach is simple yet effective which can directly handle timing loops. We compared our approach with an asynchronous design flow which is leveraging synchronous EDA tools and significant improvement is achieved.

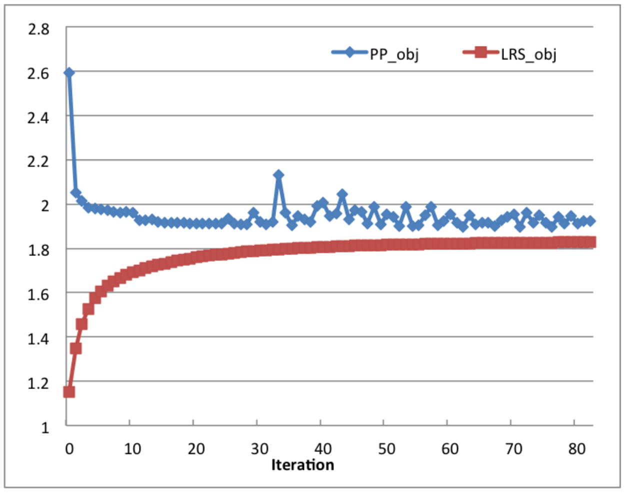

Fig. 4: The convergence sequence of s38417.

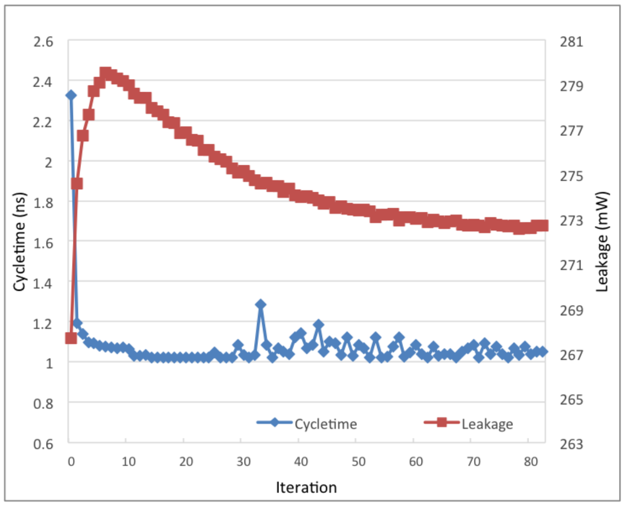

Fig. 5: Cycle time and leakage power trends of s38417.

References

[1] A. J. Martin, S. M. Burns, T.-K. Lee, D. Borkovic, and P. J. Hazewindus, "The First Aysnchronous Microprocessor: The Test Results," *SIGARCH*, pp. 95–98, 1989.
[2] A. J. Martin, A. Lines, R. Manohar, M. Nystroem, P. Penzes, R. Southworth, and U. Cummings, "The Design of an Asynchronous MIPS R3000 Microprocessor," in *Advanced Research in VLSI*, pp. 164–164, 1997.
[3] M. Davies, A. Lines, J. Dama, A. Gravel, R. Southworth, G. Dimou, and P. Beerel, "A 72-Port 10G Ethernet Switch/Router Using Quasi-Delay-Insensitive Asynchronous Design," in *ASYNC*, pp. 103–104, 2014.
[4] J. Fishburn, "TILOS: A Posynomial Programming Approach to Transistor Sizing," in *ICCAD*, 1985.
[5] M. R. Berkelaar and J. A. Jess, "Gate Sizing in MOS Digital Circuits with Linear Programming," in *DATE*, pp. 217–221, 1990.
[6] C.-P. Chen, C. Chu, and D. F. Wong, "Fast and Exact Simultaneous Gate and Wire Sizing by Lagrangian Relaxation," in *ICCAD*, 1998.
[7] D. Nguyen, A. Davare, M. Orshansky, D. Chinnery, B. Thompson, and K. Keutzer, "Minimization of Dynamic and Static Power Through Joint Assignment of Threshold Voltages and Sizing Optimization," in *ISLPED*, pp. 158–163, 2003.

Table I. Comparison on transformed ISCAS89 benchmarks

Design	Total Itr.	PP_{obj}	LRS_{obj}	Gap	Cycle Time (ns)				Leakage (μW)			
					Init	Proteus	Ours	Multi-Vt	Init	Proteus	Ours	Multi-Vt
as27	12	2.204	2.183	0.96%	0.62	0.41	0.35	0.38	454.12	547.43	483.15	139.45
as298	20	2.104	2.040	3.04%	0.85	0.48	0.44	0.46	3111.09	3837.54	3322.60	1025.05
as344	24	2.101	2.051	2.39%	1.14	0.65	0.59	0.61	4044.90	5448.73	4266.78	1331.77
as386	21	2.027	1.999	1.42%	1.14	0.67	0.54	0.58	3574.20	4435.43	3875.56	1127.87
as349	43	2.032	2.005	1.32%	1.20	0.65	0.59	0.60	4052.51	5153.59	4261.94	1321.23
as382	19	2.085	1.996	4.30%	0.94	0.58	0.49	0.50	3950.90	4713.29	4159.64	1247.96
as400	25	2.032	1.992	1.99%	1.00	0.59	0.49	0.51	4228.76	5163.96	4447.18	1398.12
as420	15	2.114	2.047	3.14%	0.74	0.43	0.39	0.41	4354.56	5563.47	4673.89	1415.92
as444	19	2.015	1.963	2.56%	0.98	0.58	0.47	0.49	4008.27	4716.75	4209.41	1270.43
as510	28	2.028	1.980	2.39%	1.41	0.74	0.66	0.69	7565.18	9401.01	8269.52	2435.62
as526	16	2.029	1.946	4.08%	0.99	0.58	0.48	0.50	4877.80	5951.92	5162.57	1558.83
as641	50	1.916	1.891	1.31%	1.36	0.65	0.61	0.61	7957.79	11109.70	8178.28	2488.67
as713	31	1.915	1.863	2.72%	1.21	0.64	0.53	0.54	7173.27	8931.69	7427.64	2228.43
as820	19	1.902	1.845	2.99%	1.63	0.80	0.68	0.69	9646.39	12654.50	10347.30	3174.51
as832	27	1.918	1.860	3.02%	1.69	0.76	0.73	0.73	10178.60	13679.50	10795.90	3365.11
as838	16	2.161	2.015	6.76%	0.86	0.55	0.47	0.47	10450.30	12322.00	11290.10	3541.02
as953	26	1.918	1.866	2.69%	1.57	0.73	0.67	0.69	14065.20	18748.80	14962.40	4586.63
as1196	20	2.028	1.938	4.43%	0.78	0.45	0.37	0.39	21147.30	26729.40	22739.80	6596.47
as1488	31	1.867	1.824	2.33%	1.96	0.92	0.77	0.81	21038.70	27106.10	22678.30	6611.33
as1238	18	2.013	1.935	3.89%	0.77	0.44	0.36	0.39	21815.70	26416.30	23438.60	6483.97
as1423	44	2.021	1.944	3.81%	1.94	1.04	0.95	0.95	18684.50	22470.90	19514.60	5901.81
as5378	42	1.964	1.866	4.95%	1.19	0.57	0.54	0.55	40032.20	52317.60	42443.10	12548.60
as9234	39	1.880	1.780	5.35%	1.69	0.82	0.71	0.70	32586.60	40344.70	33821.10	9963.48
as13207	44	1.856	1.697	8.58%	1.68	0.83	0.68	0.70	86949.50	99875.60	90722.80	26414.00
as15850	50	1.831	1.691	7.69%	2.44	1.29	0.99	0.96	105889.00	123374.00	107923.00	31622.60
as38417	50	1.901	1.813	4.66%	2.32	2.04	1.02	1.01	267671.00	267062.00	273556.00	80887.90
			Normalized	3.57%	2.192	1.213	1.000	1.024	0.963	1.095	1.000	0.295

Table II. Comparison on asynchronous benchmarks

Design	Total Itr.	PP_{obj}	LRS_{obj}	Gap	Cycle Time (ns)				Leakage (μW)			
					Init	Proteus	Ours	Multi-Vt	Init	Proteus	Ours	Multi-Vt
ALU8	38	2.100	2.010	4.32%	0.68	0.40	0.35	0.38	14849.70	20252.20	15668.10	4223.92
ALU16	16	1.988	1.848	7.08%	0.84	0.78	0.39	0.40	41103.60	41017.20	43402.50	12194.50
ALU32	17	1.762	1.605	8.87%	1.26	1.18	0.45	0.50	118062.00	118125.00	123546.00	34077.00
ALU64	19	1.848	1.567	15.18%	1.38	1.25	0.55	0.59	493119.00	494252.00	520688.00	144837.00
ACC16	27	2.114	2.073	1.91%	0.99	0.65	0.50	0.55	7198.16	8565.35	7945.34	2216.16
ACC32	50	2.092	1.987	5.01%	1.30	0.73	0.67	0.67	16291.60	22786.80	17217.10	5046.62
ACC64	37	2.049	1.975	3.63%	1.13	0.59	0.56	0.57	48672.90	63712.10	51894.60	15904.20
ACC128	50	2.081	1.895	8.94%	1.35	0.89	0.69	0.69	125107.00	144956.00	131821.00	39325.50
GCD	35	1.971	1.925	2.30%	1.77	1.59	0.82	0.84	7017.06	6967.30	7359.21	2139.78
FU	40	1.903	1.855	2.52%	1.77	0.80	0.74	0.76	75562.70	106374.00	80706.60	24465.00
			Normalized	5.98%	2.180	1.550	1.000	1.041	0.947	1.027	1.000	0.284

[8] P. A. Beerel, G. Dimou, and A. Lines, "Proteus: An ASIC Flow for GHz Asynchronous Designs," *Design Test of Computers*, vol. 28, no. 5, pp. 36–51, 2011.

[9] Y. Thonnart, E. Beigne, and P. Vivet, "A Pseudo-synchronous Implementation Flow for WCHB QDI Asynchronous Circuits," in *ASYNC*, pp. 73–80, 2012.

[10] B. Ghavami and H. Pedram, "Low Power Asynchronous Circuit Back-End Design Flow," *Microelectronics Journal*, 2011.

[11] M. Prakash, *Library Characterization and Static Timing Analysis of Asynchronous Circuits*. ProQuest, 2007.

[12] M. S. Bazaraa, H. D. Sherali, and C. M. Shetty, *Nonlinear Programming: Theory and Algorithms*. 2013.

[13] W. Ning, "Strongly NP-Hard Discrete Gate-Sizing Problems," *TCAD*, vol. 13, no. 8, pp. 1045–1051, 1994.

[14] S. S. Sapatnekar, V. B. Rao, P. M. Vaidya, and S.-M. Kang, "An Exact Solution to the Transistor Sizing Problem for Cmos Circuits Using Convex Optimization," *TCAD*, vol. 12, no. 11, pp. 1621–1634, 1993.

[15] J. Hu, A. B. Kahng, S. Kang, M.-C. Kim, and I. L. Markov, "Sensitivity-Guided Metaheuristics for Accurate Discrete Gate Sizing," in *ICCAD*, pp. 233–239, 2012.

[16] M. M. Ozdal, C. Amin, A. Ayupov, S. Burns, G. Wilke, and C. Zhuo, "The ISPD-2012 Discrete Cell Sizing Contest and Benchmark Suite," in *ISPD*, pp. 161–164, 2012.

[17] M. M. Ozdal, C. Amin, A. Ayupov, S. M. Burns, G. R. Wilke, and C. Zhuo, "An Improved Benchmark Suite for the ISPD-2013 Discrete Cell Sizing Contest," in *ISPD*, pp. 168–170, 2013.

[18] G. Flach, T. Reimann, G. Posser, M. Johann, and R. Reis, "Effective Method for Simultaneous Gate Sizing and Vth Assignment Using Lagrangian Relaxation," *TCAD*, vol. 33, no. 4, pp. 546–557, 2014.

[19] V. S. Livramento, C. Guth, J. L. Gntzel, and M. O. Johann, "Fast and Efficient Lagrangian Relaxation-Based Discrete Gate Sizing," in *DATE*, pp. 1855–1860, 2013.

[20] M. M. Ozdal, S. Burns, and J. Hu, "Algorithms for Gate Sizing and Device Parameter Selection for High-performance Designs," *TCAD*, vol. 31, no. 10, pp. 1558–1571, 2012.

[21] J. Wang, D. Das, and H. Zhou, "Gate Sizing by Lagrangian Relaxation Revisited," *TCAD*, vol. 28, pp. 1071–1084, July 2009.

[22] P. A. Beerel, A. Lines, M. Davies, and N.-H. Kim, "Slack Matching Asynchronous Designs," in *ASYNC*, pp. 11–pp, 2006.

[23] M. Najibi and P. A. Beerel, "Performance Bounds of Asynchronous Circuits with Mode-Based Conditional Behavior," in *ASYNC*, 2012.

[24] J. Magott, "Performance Evaluation of Concurrent Systems using Petri Nets," *Information Processing Letters*, vol. 18, no. 1, pp. 7–13, 1984.

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

[26] K. S. Stevens, R. Ginosar, and S. Rotem, "Relative Timing," *VLSI Systems*, vol. 11, no. 1, pp. 129–140, 2003.