Simultaneous Slack Matching, Gate Sizing and Repeater Insertion for Asynchronous Circuits

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

***Abstract*—Slack matching, gate sizing and repeater insertion are well known techniques applied to asynchronous circuits to improve their power and performance. Existing asynchronous optimization flows typically perform these optimizations sequentially, which may result in sub-optimal solutions as all these techniques are interdependent and affect one another. In this paper, we present a unified leakage power optimization framework by performing simultaneous slack matching, gate sizing and repeater insertion. In particular, we apply Lagrangian relaxation to integrate all these techniques into a single optimization step. A methodology to handle slack matching under the Lagrangian relaxation framework is proposed. Also, an effective look-up table based repeater insertion technique is developed to speed up the algorithm. Our approach is evaluated using a set of asynchronous designs and compared with both a sequential approach and a commercial asynchronous optimization flow. The experimental results have achieved significant savings in leakage power and demonstrated the effectiveness of our approach.**

I. Introduction

Asynchronous designs have been demonstrated to be able to achieve both higher performance and lower power compared with their synchronous counterparts [1] [2] [3]. However, due to the lack of proper EDA tool support, the design cycle for asynchronous circuits is much longer compared with the one for synchronous circuits. Thus, even with many advantages, asynchronous circuits are still not the mainstream in the industry, and it is very important to develop EDA tools for asynchronous circuits design.

Stalls are major obstacles limiting the performance of pipelined asynchronous circuits [4]. Due to the slack elasticity for most asynchronous designs, adding pipeline buffers to the design will not change its input/output functionality, but can help remedy the stalls [5]. Thus, slack matching, which inserts minimum number of pipeline buffers to guarantee the most critical cycle meets the desired cycle time, is widely used for asynchronous circuits [6]. Most previous works related to slack matching formulate the problem as a mixed integer linear program (MILP) [5] [7] [8], which is NP-Complete and the integral constraints need to be relaxed in order to solve the problem efficiently. In [9], a heuristic algorithm is proposed to solve the problem by leveraging the asynchronous communication protocol.

Other than slack matching, gate sizing and repeater insertion are also very effective techniques to reduce the delay and power consumption for asynchronous circuits. Gate sizing and repeater insertion for synchronous circuits has been studied for decades and there are many works tackling these problems [10] [11] [12]. However, those works cannot be directly applied to asynchronous circuits, due to the intrinsic differences between asynchronous and synchronous circuits in terms of performance analysis and optimization. In [13], a gate sizing and V_{th} assignment approach for asynchronous circuits has been proposed. It achieved significant improvements compared with previous asynchronous gate sizing approaches. To the best of our knowledge, there is no work on repeater insertion for asynchronous circuits.

Most automated asynchronous design flows apply slack matching, gate sizing and repeater insertion separately either in a sequential manner [14] [15] or in an iterative manner [16]. In Proteus [16], a MILP based slack matching optimization is performed first, followed by gate sizing and repeater insertion which are done by leveraging synchronous EDA tools. Since all these three optimization techniques are closely related to each other, doing them separately may not explore the solution space sufficiently thus yields sub-optimal results. During our experiments, in term of the number of gates, the circuits optimized by Proteus contain 27.1% pipeline buffers on average. This huge amount of pipeline buffers inserted at the slack matching step create a serious area and power overhead, which can be even more critical for asynchronous circuits, since asynchronous designs intrinsically have higher gate count than its corresponding synchronous design. Another disadvantage is that earlier steps of the separated optimization approach have to perform optimizations based on inaccurate delay values. In Proteus, the slack matching is performed based on a rough unit delay model, which simply counts the number of gates along the timing path. Considering the dominating interconnect delays in advanced technologies and the gate sizing and repeater insertion operations performed in later steps, this unit delay model can be very inaccurate and even misleading to the optimization algorithms.

In this paper, we address the problem of minimizing the total leakage power consumption while guaranteeing a target cycle time for unconditional asynchronous pipelined circuits. Three different optimization techniques: slack matching, gate sizing and repeater insertion are effectively joined together under the Lagrangian relaxation (LR) framework. As far as we know, this is the first work that formulates and solves this simultaneous optimization problem combining all these three techniques together.

Our approach is distinctive from previous ones by offering the following benefits:

- Much fewer pipeline buffers can be used for the slack matching purpose, since some stalls can simply be fixed

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

by either gate sizing or repeater insertion which have much less area overhead and consume less power.
- Our approach can prevent excessive sizing when a gate is driving a large load, as we consider repeater insertion together with the gate sizing.
- When design contains extremely long wire delays, i.e., cross chip interconnections, our approach can explore the solution of adding pipeline buffers to break the channel into multiple pipeline stages, which is more beneficial than just doing gate sizing or repeater insertion.
- More accurate delay estimation at each optimization step can be achieved by calculating the delay using the non-linear delay model (NLDM).

The main contributions of this paper are as follows:

- A unified LR framework incorporating gate sizing, repeater insertion and pipeline buffer insertion together.
- Methodology for handling pipeline buffer insertion under the LR framework, especially how to update the corresponding Lagrangian multipliers.
- A fast look-up table based repeater insertion approach.
- Results which show significant improvements compared with both the sequential approach and a commercial asynchronous optimization flow.

The rest of this paper is organized as follows. In Section II, a motivating example is presented. Section III shows an overview of the framework. Section IV introduces the backgrounds of LR. Section V presents our optimization algorithm. Finally, the experiments are presented in Section VI.

II. A Motivating Example

In this paper, we use the Full Buffer Channel Net (FBCN) [5] to model our asynchronous circuits. A FBCN is a specific form of Petri net [17]. The idea is to model each leaf cell as a *transition*. Channels between cell ports are modeled with a pair of *places* which are annotated with delay information. The handshaking signals are modeled as *tokens*. A simple asynchronous three-stage pipeline and its corresponding FBCN model is shown in Fig. 1 (a) and (b).

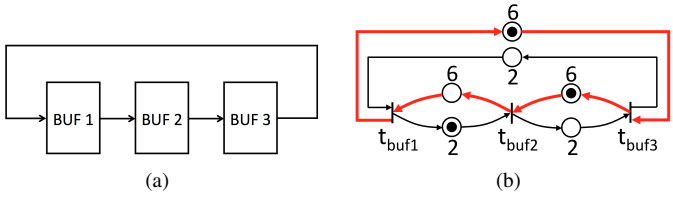

Fig. 1: (a) Three-stage pipeline. (b) FBCN model.

Here, *transition* t_{buf1}, t_{buf2} and t_{buf3} represent the buffer cells. Circles are *places* which represent the channels between neighboring buffer cells. In particular, *places* containing *tokens* are represented by circles marked with a black dot. In Fig. 1, the propagation delay is $2+2+2=6$, which is the time for tokens propagate from t_{buf1} to t_{buf3} and back to t_{buf1}. The local cycle time is $2+6=8$, which is the shortest time for a buffer to complete a handshake with its neighbors. Since the propagation delay is less than local cycle time, stall happens. The performance of this design is bounded by the highlighted most critical cycle. The corresponding cycle time is $(6+6+6)/2=9$, which is calculated by the cycle delay divided by the total number of tokens along this cycle.

The stall can be resolved by slack matching, which adds an extra pipeline stage in the design as shown in Fig. 2 (a). The slack matched design operates at desired local cycle time. The most critical cycle is highlighted, which is same as the local handshaking cycle.

Another way to resolve the stall is to improve the acknowledgment (*ack*) time, as shown in Fig. 2 (b). This can be done by simply sizing up BUF3 or inserting repeaters at its output *ack* pin. Sizing gates or inserting repeaters are much more economical than inserting pipeline buffers, since pipeline buffers, which contain extra handshaking circuits, are much bigger. For the cell library we have, the smallest size pipeline buffer is 4.8X bigger than the smallest size repeater.

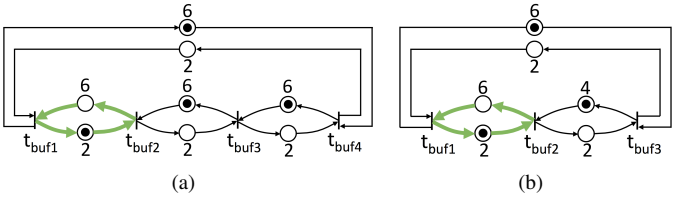

Fig. 2: (a) Stall fixed by inserting pipeline buffers. (b) Stall fixed by gate sizing or repeater insertion.

Considering typical asynchronous flows which perform the optimization sequentially, if a slack matching solution as shown in Fig. 2 (a) is applied first, it is very difficult for the flow to go back to the better solution as shown in in Fig. 2 (b). Therefore, we develop the unified optimization approach which is able to achieve much better results.

III. Optimization Framework Overview

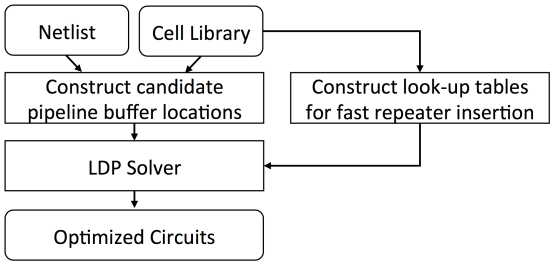

Fig. 3: High-level View of Our Framework.

A high-level view of our optimization framework is shown in Fig. 3. The fundamental idea is to size the gates or insert proper repeaters / pipeline buffers to minimize the leakage power while satisfying the timing constraints. However, if we simply enumerate all the gate size, repeater or pipeline buffer choices, the runtime will not be affordable due to the enormous number of possible combinations. Thus, before solving the problem, the first thing we need to consider is how to limit the

solution space and speed up the evaluation process, while still keeping a good solution quality. We do this by constructing candidate pipeline buffer locations and performing table look-up for repeater insertion.

The generated candidate buffer locations and look-up tables are then fed into our Lagrangian dual problem (LDP) solver, where the gate sizing, buffer insertion and repeater insertion problems are joined by Lagrangian multipliers, acting as "weights" associated with each timing arc. The weights help us to find a proper sizing and buffer / repeater insertion solution for the circuit, and the LR framework provide us a systemic way to adjust the weights at each iteration.

IV. Lagrangian Relaxation Framework

LR is a very useful mathematical approach which transforms the constrained primal problem ($\mathcal{PP}$) into an unconstrained and easier LR subproblem ($\mathcal{LRS}$). Inspired by [10], the special circuit structure allows us to further transform $\mathcal{LRS}$ into an equivalent but even simpler problem $\mathcal{LRS}^*$. For a given set of non-negative LR multipliers $\boldsymbol{\lambda}$, solving $\mathcal{LRS}^*$ provides us a lower bound of $\mathcal{PP}$. Then, the LR dual problem ($\mathcal{LDP}$) which provides us a solution to $\mathcal{PP}$ can be solved by iteratively solving a sequence of $\mathcal{LRS}^*$.

For an asynchronous circuit modeled with FBCN, our primal problem which minimizes total leakage power subject to performance constraints can be formulated similar to [13] as shown below:

$$\begin{aligned} \mathcal{PP}: \quad & \textbf{minimize} \quad \text{leakage}(\mathbf{g}, \mathbf{b}, \mathbf{r}) \\ & \text{Subject to} \quad a_i + D_{ij} - m_{ij}\tau \le a_j \quad \forall\, p(i,j) \in P \end{aligned}$$

where $\boldsymbol{g}$ is the set of select gates, $\boldsymbol{b}$ is the buffer solution and $\boldsymbol{r}$ is the repeater solution. τ is the given target cycle time. Let T be the set of transitions and P be the set of places in the FBCN model. a_i and a_j denote the arrival times associated with transitions t_i and t_j. $p(i,j)$ denotes the place between t_i and t_j. D_{ij} is the delay associated with $p(i,j)$. $m_{ij} = 1$ if $p(i,j)$ contains a token and 0 otherwise.

By relaxing all constraints into the objective function, we can obtain the $\mathcal{LRS}$ as:

$$\begin{aligned} \mathcal{LRS}: \quad & \textbf{minimize} \quad \text{leakage}(\mathbf{g}, \mathbf{b}, \mathbf{r}) \\ & + \sum_{\forall(i,j)} \lambda_{ij}(a_i + D_{ij} - m_{ij}\tau - a_j) \end{aligned}$$

Similar to [10], $\mathcal{LRS}$ can be further simplified into $\mathcal{LRS}^*$ by applying $\mathcal{KKT}$ optimality conditions:

$$\mathcal{KKT}: \quad \sum_{\forall(k,j)} \lambda_{kj} = \sum_{\forall(i,k)} \lambda_{ik} \quad \forall\, k \in T$$

$$\begin{aligned} \mathcal{LRS}^*: \ & \textbf{minimize} \ \ \text{leakage}(\mathbf{g}, \mathbf{b}, \mathbf{r}) \\ & - \sum_{\forall(i,j)} \lambda_{ij} m_{ij}\tau + \sum_{\forall(i,j)} \lambda_{ij} D_{ij} \end{aligned}$$

Finally, we obtain $\mathcal{LDP}$ by maximizing $\mathcal{LRS}^*$:

$$\begin{aligned} \mathcal{LDP}: \quad & \textbf{maximize} \quad \mathcal{LRS}^* \\ & \text{Subject to} \quad \boldsymbol{\lambda} \ge \mathbf{0}, \ \boldsymbol{\lambda} \in \mathcal{KKT} \end{aligned}$$

V. Simultaneous Gate Sizing, Repeater Insertion and Pipeline Buffer Insertion

A. Constructing Candidate Pipeline Buffer Location

Fig. 4 shows a three-stage asynchronous pipeline implemented using the PCHB template [18]. Stage 1 and stage 3 are computation stages. In particular, domino logic cells (LOGIC) are used for computation and control circuit (CTRL), C-elements (C) are used to perform handshaking. The dual rail channel contains two data wires ($A[0].0$, $A[0].1$), and one wire (L_e) for the ack signal. Thus, we can easily identify all the channels in the circuit and pre-insert a candidate pipeline buffer inside each channel, similar to stage 2 in Fig. 4.

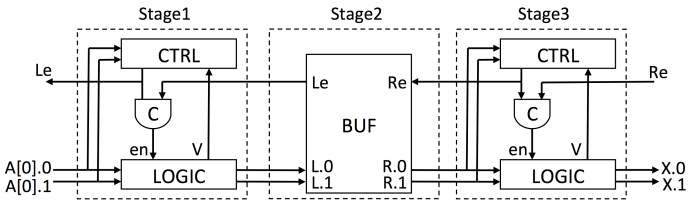

Fig. 4: A three-stage PCHB pipeline.

Different from regular pipeline buffers, we assign the pre-inserted pipeline buffer with two modes: transparent and opaque, as shown in Fig. 5 (a) and (b) respectively. The transparent mode is used to model the situation in which no buffer is inserted and the opaque mode is used to model the opposite situation. In transparent mode, the pipeline buffer has three timing arcs denoted as $t1$ to $t3$, acting as wires connecting the corresponding ack or data pins. It contributes zero leakage power to the circuit. Also, during static timing analysis, the slew values seen at its input pins will be propagated to the corresponding output pins for all its fanout cells. Similarly, the load capacitance seen at its output pins will be forwarded to the input pins for the fanin cells. In opaque mode, the pre-inserted buffer acts as a normal pipeline buffer and there are 9 timing arcs denoted as $t1$ to $t9$ from each input pin to each output pin. Then, instead of actually modifying the netlist, the algorithm can simply switch the buffer between transparent and opaque mode to achieve the same effects as removing / inserting the buffer.

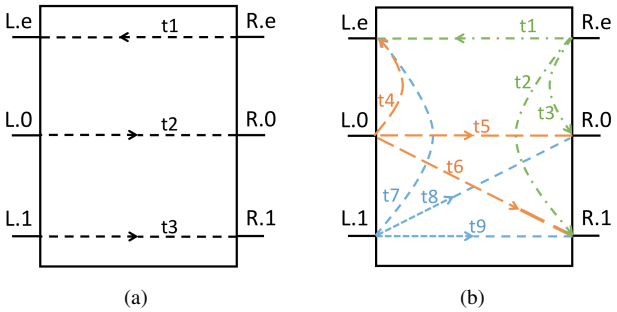

Fig. 5: Pre-inserted pipeline buffer: (a) Transparent mode (b) Opaque mode

As described in Sec. IV, the set of Lagrangian multiplier $\boldsymbol{\lambda}$ needs to satisfy $\mathcal{KKT}$ conditions during the update process. Let λ_{t1} denote the λ associated with timing arc $t1$ and similarly for all other λs. Let us consider the λ sum at pin $L.e$ and $R.e$. Since these two pins are connected by a single timing arc in transparent mode, the λ sum at pin $L.e$ and the λ sum at pin $R.e$ should be equal, and they should keep to be equal when the buffer transforms between transparent mode and opaque mode. This requires us to have: $\lambda_{t1}+\lambda_{t2}+\lambda_{t3} = \lambda_{t1}+\lambda_{t4}+\lambda_{t7}$ for the λs in opaque mode. However, the λ update in opaque mode might not follow the above rule, which can make the λs violating the $\mathcal{KKT}$ conditions when the buffer is transformed back to transparent mode. Similarly, for other pins, the same issue will also happen.

A simple solution is to only update λ_{t1}, λ_{t5} and λ_{t9} while keeping all other λs to be 0 in opaque mode. However, this might put too much restrictions on λ values and make them unable to accurately reflect the criticality of each timing arc. Thus, we propose a better solution where we enforce the following equality constraints during λ update:

$$(\lambda_{t2} = \lambda_{t7}) \wedge (\lambda_{t3} = \lambda_{t4}) \wedge (\lambda_{t6} = \lambda_{t8})$$

These constraints guarantee the updated λs always satisfy the $\mathcal{KKT}$ conditions in both modes, while it avoids putting too much restrictions on the original λ values.

B. Constructing Look-up Tables for Fast Repeater Insertion

If we do not consider any blockages, repeaters can be inserted at any location of the wires. In order to limit the solution space and simplify our algorithm, here we only consider inserting repeaters at the input / output pins of each gate. However, even under such an assumption, the possible choices for repeater insertion are still too many even for one gate, because we can have different size or number of repeaters at each pin. Therefore, we propose a look-up table technique to speed up our evaluation process. In particular, we construct a 2D look-up table for each pin of each gate in our standard cell library. The X-axis of the look-up table is the load capacitance driven by the repeaters. The Y-axis is the sum of the values of λs at this pin, as shown in Fig. 6.

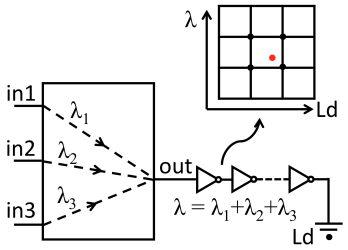

Fig. 6: Look-up table at each pin.

Given the load and λ value of each look-up table entry, we evaluate all the possible repeater insertion choices at this pin based on a typical input slew. The best choice, i.e., the one providing the smallest cost, will be stored in the table. The cost is evaluated based on the following cost function:

$$\text{Cost}(g_i) = \text{leakage}(g_i) + \sum_{(u,v)\in Arc_i} \lambda_{uv} D_{uv} \qquad (1)$$

Here, Arc_i is defined to be the set of timing arcs of repeater g_i and all the fanin and fanout gates of g_i.

It might happen that the actual λ and load value calculated by the LDP solver does not match any of the index values in the look-up table. In this situation, we simply evaluate all four choices surrounding this point and pick the best one.

C. Solving $\mathcal{LDP}$

We apply a direction finding approach inspired by [19] to solve $\mathcal{LDP}$, as shown in Fig. 7.

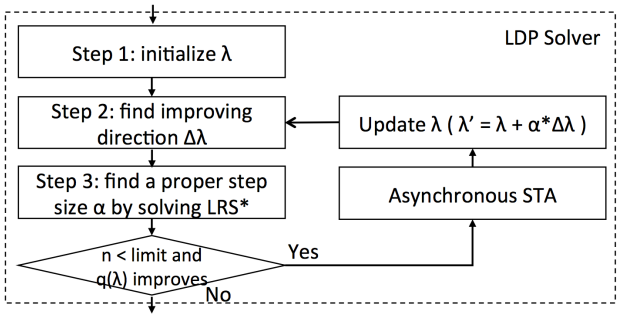

Fig. 7: Lagrangian dual problem solver.

In step 1, we find an initial set of non-negative λs satisfying the $\mathcal{KKT}$ conditions.

In step 2, an improving feasible direction $\Delta\boldsymbol{\lambda}$ can be found by solving the following linear program, which maximize the first order approximation of $\mathcal{LRS}^*$:

$$\begin{aligned} \mathcal{DF}: \quad & \textbf{maximize} \quad \sum_{\forall(i,j)} \Delta\lambda_{ij} D_{ij} - \sum_{\forall(i,j)} \Delta\lambda_{ij} m_{ij} \tau \\ & \text{Subject to} \quad \boldsymbol{\lambda} \geq \mathbf{0},\ \boldsymbol{\lambda} \in \mathcal{KKT} \\ & \qquad\qquad \max(-u, -\lambda_{ij}) \leq \Delta\lambda_{ij} \leq u \end{aligned}$$

here u is used to bound the objective function and avoid it goes to infinity, similar to [19].

In step 3, we find a proper step size α by optimizing along the feasible direction $\Delta\boldsymbol{\lambda}$ using line search techniques. In particular, we solve the $\mathcal{LRS}^*$ for a given set of $\boldsymbol{\lambda}$ at each potential step. The step size which achieves $q(\boldsymbol{\lambda}+\alpha\Delta\boldsymbol{\lambda}) > q(\boldsymbol{\lambda})$ will be selected as α. Here, $q(\boldsymbol{\lambda})$ denotes the optimal objective value of $\mathcal{LRS}^*$.

We keep iterating between steps 2 and 3 until $q(\boldsymbol{\lambda})$ does not improve or the number of iterations (n) exceeds the limit.

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

Since asynchronous circuits contain loops, it does not has a topological order which is commonly used in synchronous optimization algorithms. Thus, here we use a sequential update technique as shown in Algorithm 1. The idea is to traverse all the gates in a sequential order and locally pick a best solution which minimize $\mathcal{LRS}^*$. If the newly picked solution

is different from the old one, we will reevaluate all its fanout gates. Please note that in Algorithm 1, the *gate* refers to all the gates in the original circuit and the pre-inserted candidate pipeline buffers, but it does not represent the repeaters inserted by our algorithm. For a regular gate, a *solution* at a gate means a proper size and repeater insertion choices of this gate. For a candidate pipeline buffer, the *solution* also denotes whether the buffer is in opaque or transparent mode.

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

Ensure: a proper solution for each gate which minimize $\mathcal{LRS}^*$
1: Assign all the gates with an initial solution;
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 solution be s_i^j;
5: Select a better solution s_i^k for gate g_i; /* Algorithm 2 */
6: **if** $s_i^j \neq s_i^k$ **then**
7: Assign g_i with this new solution s_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**

Algorithm 2 shows our local evaluation algorithm which find the best local solution at each gate based on the following cost function:

$$\text{Cost}(g_i) = \text{leakage}(g_i) - \sum_{(u,v)\in Arc_i} \lambda_{ij} m_{ij} \tau + \sum_{(u,v)\in Arc_i} \lambda_{uv} D_{uv} \quad (2)$$

Similar to equation (1), Arc_i is the set of timing arcs of gate g_i and all the timing arcs of g_i's fanin and fanout gates.

Algorithm 2 Local Evaluation

Ensure: Best solution for g_i which locally minimize $\mathcal{LRS}^*$
1: **if** g_i is a regular gate **then**
2: Select a proper sizing, repeater insertion option for g_i;
3: **else** /* g_i is a candidate pipeline buffer */
4: **if** g_i is in opaque mode **then**
5: Change g_i to transparent mode;
6: Local timing update;
7: If the cost is not reduced, recover to opaque mode;
8: **else** /* g_i is in transparent mode */
9: Change g_i to opaque mode;
10: Update λ for all the timing arcs of g_i;
11: Select best sizing and repeater insertion solution for g_i;
12: If the cost is not reduced, recover to transparent mode;
13: **end if**
14: **end if**
15: **return** bestSolution;

In step 2 of Algorithm 2, the method we used to pick proper size and repeaters of this gate is simply evaluating all its possible sizing and repeater insertion options. In particular, we first select a certain size for this gate, then the repeater insertion options at each of its pin can be found using the look-up tables. The cost of each sizing and repeater insertion combination will be calculated based on equation (1).

VI. Experiments

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

Our unified optimization approach is tested using two sets of asynchronous benchmarks. First is a set of asynchronous benchmarks transformed from ISCAS89 benchmarks. Second is a set of specific asynchronous designs. In particular, we use different bit width on the datapath of ALU and Accumulator designs to generate the set of benchmarks with different number of gates. All the designs are transformed or synthesized using the Proteus front-end flow [16].

Accurate non-linear delay model are used to calculate delay and slew value based on the look-up tables from Proteus standard cell library. Cell interconnections are modeled as lumped capacitance, which is obtained by extraction after placement and routing. Since the original cell library does not contain leakage power, we assign a leakage power for each cell which is proportional to its area. The cycle time achieved by Proteus is used as a timing constraint for our unified flow and the sequential approach described below.

For comparison purpose, we implemented a sequential approach similar to our unified optimization flow, but it only performs one type of optimization at each iteration. In particular, the sequential approach starts with 10 iterations of pipeline buffer insertion, followed by 10 iterations of repeater insertion and 30 iterations of gate sizing. We use this order because gate sizing is the more fine-grained optimization and it is better to be applied at last.

Comparison results on the transformed ISCAS89 benchmarks are shown in Table I. "# of gates" column shows the total number of gates. "Cand." column shows the number of pre-inserted candidate pipeline buffers. "Target" column shows the target cycletime obtained from Proteus flow. The "Proteus", "Seq." and "Ours" columns show the results of the Proteus, the sequential approach and our approach respectively. Leakage power, the number of inserted buffers and the number of inserted repeaters are compared among different flows. All results satisfy the target cycletime constraints. The results show our approach is much better in power consumption and insert fewer buffers and repeaters. Comparing the leakage power, on average, our approach is 56.5% better than the Proteus flow and 14.1% better than the sequential flow. Table II shows the comparison results on the specific asynchronous benchmarks. Similar improvements are achieved. For the leakage power, our approach is 27.1% better than the Proteus flow and 11.1% better than the sequential flow.

The significant improvements in both sets of benchmarks suggest that all these techniques are very closely related to each other and the proposed joint optimization approach can provide significant benefits compared with the non-simultaneous ones. Proteus does not have a separate circuit optimization step and so we are not able to measure its runtime. The average runtime of our algorithm is around 6.5 minutes, which indicates our flow runs fast enough and will

Table I. Comparison on transformed ISCAS89 benchmarks

Design	# of gates	Cand.	Cycle Time (ns)			Leakage (μW)			Buffers			Repeaters		
			Target	Seq.	Ours	Proteus	Seq.	Ours	Proteus	Seq.	Ours	Proteus	Seq.	Ours
as27	47	13	0.40	0.39	0.39	601.34	488.056	395.16	7	6	3	6	3	1
as298	223	98	0.52	0.45	0.49	3528.58	2940.16	2351.60	63	36	7	14	42	13
as386	268	115	0.73	0.68	0.63	4227.10	2953.01	2834.33	70	21	21	22	59	21
as349	324	115	0.73	0.65	0.73	4972.49	3676.08	2718.56	85	43	3	30	49	14
as382	268	119	0.62	0.61	0.58	4437.92	3705.66	2757.96	69	40	0	15	51	14
as400	281	126	0.59	0.57	0.58	4774.81	3176.96	3175.65	83	18	19	14	62	20
as420	324	122	0.44	0.39	0.43	5217.87	4456.1	3394.69	88	25	12	20	45	20
as444	270	119	0.56	0.51	0.54	4571.60	3252.37	2845.32	73	19	5	14	52	13
as510	572	294	0.92	0.91	0.92	8935.97	8862.22	6486.70	118	111	29	53	144	44
as526	323	147	0.60	0.60	0.53	5448.18	4659.59	3846.67	98	61	25	16	65	25
as641	723	190	0.79	0.78	0.78	9860.11	6157.28	4799.62	213	59	2	87	103	30
as713	666	175	0.70	0.64	0.69	9031.29	5896.07	4677.90	177	29	8	82	97	32
as832	803	361	0.99	0.99	0.97	12194.40	9511.26	8190.31	190	80	38	97	171	55
as838	747	300	0.58	0.54	0.58	12049.80	8014.05	7371.23	226	34	31	40	168	53
as953	1015	469	1.01	0.98	0.99	16281.70	12698	12528.00	271	107	140	84	239	94
as1488	1430	772	1.14	1.03	0.98	24655.10	20808.1	19559.20	410	275	196	116	135	134
as5378	2742	1242	0.71	0.67	0.71	46083.00	31520.9	27859.40	870	233	67	187	591	204
as9234	2236	1021	1.05	1.01	1.05	36528.00	24193.2	22925.20	686	83	58	130	152	155
as13207	6088	2583	1.02	0.93	0.99	96456.20	68898.5	59234.00	1945	567	154	451	353	336
				Normalized		1.565	1.141	1.000	7.020	2.258	1.000	1.156	2.020	1.000

Table II. Comparison on asynchronous benchmarks

Design	# of gates	Cand.	Cycle Time (ns)			Leakage (μW)			Buffers			Repeaters		
			Target	Seq.	Ours	Proteus	Seq.	Ours	Proteus	Seq.	Ours	Proteus	Seq.	Ours
ALU8	957	364	0.46	0.46	0.41	17107.00	15006.80	18086.70	146	120	92	41	219	73
ALU16	2511	1002	0.66	0.48	0.56	41069.20	40319.60	37103.50	594	382	256	69	622	188
ACC16	609	215	0.62	0.60	0.59	8974.96	7585.64	6386.76	100	72	18	69	105	31
ACC32	1323	469	0.88	0.83	0.80	20363.20	16475.50	15534.40	268	175	192	136	218	97
ACC64	3619	1188	0.71	0.71	0.69	57698.60	44570.90	33062.80	1291	294	111	264	587	200
FU	5805	2107	1.42	1.21	1.32	75910.00	67900.00	63584.70	1692	590	609	501	1047	385
GCD	475	223	1.58	1.20	1.30	6988.86	7544.86	5710.49	89	73	7	30	106	35
				Normalized		1.271	1.111	1.000	3.253	1.328	1.000	1.100	2.878	1.000

not be a runtime bottleneck of the design process.

VII. Conclusions

In this paper, we have proposed a simultaneous slack matching, gate sizing and repeater insertion approach for asynchronous circuits. We apply Lagrangian relaxation to integrate all these techniques into a single optimization step. The relaxed problem is further simplified using KKT conditions. Effective techniques to handle pipeline buffer insertion and repeater insertion under the Lagrangian relaxation framework are proposed. A local evaluation algorithm is also developed to solve the relaxed problem efficiently. The experimental results show significant improvements on power consumption and demonstrate the benefits of performing these optimizations simultaneously rather than sequentially.

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*, 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. Spars and S. Furber, *Principles of Asynchronous Circuit Design*. Springer, 2002.

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

[6] P. A. Beerel, R. O. Ozdag, and M. Ferretti, *A Designer's Guide to Asynchronous VLSI*. Cambridge University Press, 2010.

[7] P. Prakash and A. J. Martin, "Slack Matching Quasi Delay-insensitive Circuits," in *ASYNC*, pp. 10–pp, 2006.

[8] M. Najibi, P. Beerel, *et al.*, "Integrated fanout optimization and slack matching of asynchronous circuits," in *ASYNC*, pp. 69–76, 2014.

[9] G. Venkataramani and S. C. Goldstein, "Leveraging Protocol Knowledge in Slack Matching," in *ICCAD*, pp. 724–729, 2006.

[10] C.-P. Chen, C. Chu, and D. F. Wong, "Fast and Exact Simultaneous Gate and Wire Sizing by Lagrangian Relaxation," in *ICCAD*, 1998.

[11] J. Fishburn, "TILOS: A Posynomial Programming Approach to Transistor Sizing," in *ICCAD*, pp. 326–328, 1985.

[12] L. P. Van Ginneken, "Buffer Placement in Distributed RC-tree Networks for Minimal Elmore Delay," in *Circuits and Systems, 1990., IEEE International Symposium on*, pp. 865–868, IEEE, 1990.

[13] G. Wu, A. Sharma, and C. Chu, "Gate Sizing and Vth Assignment for Asynchronous Circuits Using Lagrangian Relaxation," in *ASYNC*, 2015.

[14] A. Yakovlev, P. Vivet, and M. Renaudin, "Advances in Asynchronous Logic: From Principles to GALS & NoC, Recent Industry Applications, and Commercial CAD Tools," in *DATE*, pp. 1715–1724, 2013.

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

[16] P. A. Beerel, G. Dimou, and A. Lines, "Proteus: An ASIC Flow for GHz Asynchronous Designs," *Design Test of Computers*, pp. 36–51, 2011.

[17] J. L. Peterson, *Petri Net Theory and the Modeling of Systems*. Prentice Hall PTR, 1981.

[18] A. M. Lines, *Pipelined Asynchronous Circuits*. Master's thesis, California Institute of Technology, 1998.

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