Automatic Slack Matching of Asynchronous Circuits Utilized in the Synthesis Tool Framework

Behnam Ghavami Hossein Pedram

Abstract— **Slack matching is the problem of adding pipeline buffers to an asynchronous pipeline design in order to prevent stalls and improve performance. This paper proposes a methodology for automatic slack matching of asynchronous circuits utilized in a framework of asynchronous synthesis toolset. The proposed technique is based on Simulated Annealing method and exploits the advantages of both static and dynamic performance analysis to provide enough results in an acceptable time. The utilized performance model is a Timed Petri Net which can extended to support choice places to capture the conditional behavior of the system. We implemented this method in the framework of asynchronous synthesis tool named as AsyncTool and optimized circuits using this technique during synthesis process. Experimental results on a large set of ISCAS benchmark indicate that our proposed technique can achieve on average 38% enhancement for performance while there is 24% area penalty.**

Index Terms—**Asynchronous Circuit, Performance, Slack Matching, PetriNet, Simulated Annealing.**

I. Introduction

Pure synchronous design strategy has been shown not promising enough to fulfill the need for more transistors on a single die in high performance applications. Asynchronous components are currently appeared in most synchronous circuits to tolerate clock skew and large interconnect delays[1]. In asynchronous circuits, local signaling eliminates the need for global synchronization which exploits some potential advantages; beside the elimination of the clock skew and tolerating interconnect delays, asynchronous circuits are more tolerant to process variations and external voltage fluctuations. They are more modularly synthesizable, potentially faster and have lower energy consumption[2].

As asynchronous circuits gain popularity due to their potential advantages, the lack of a unified automated synthesis, evaluation and optimization framework is highlighted. Performance evaluation and optimization are the key problems in asynchronous designs[2][3][4]. The basic problem is that the complex dependencies between highly concurrent events make direct optimization for performance very difficult. While synchronous performance estimation is based on a static critical path analysis affected only by the delay of components and interconnecting wires, it has been shown that the performance of an asynchronous circuit depends on dynamic factors like the number of tokens in the circuit and the value of input data items[3]. Therefore evaluating an optimizing the performance has more complexity.

There are two basic approaches to the performance optimization of asynchronous circuits. The first approach involves using performance analysis techniques to guide manual or semi-automated design changes (e.g.,[5]). The alternative approach is to develop synthesis techniques that directly optimize for performance. Successful efforts in this area have addressed transistor sizing[3], technology mapping [6] and allocation and scheduling (e.g., [7]) in high-level synthesis.

This paper presents a new method for optimizing pipelined of asynchronous circuits under design parameter constraints. In our case, the system to be optimized is a network of asynchronous leaf-cells connected with channels. The local cycle times in the leaf-cells and of the channels are assumed to meet the desired cycle time. A framework for automatic performance optimization of asynchronous systems is introduced, and an abstract performance model of the circuit on which the basic pipeline optimization problem can be defined is proposed. This abstract performance model is sufficient to characterize a variety of pipelining schemes. This model can be extended to support choices and is not restricted to deterministic pipelines. A heuristic algorithm is presented that demonstrates the feasibility of the optimization method for moderately sized models. The experimental results on large scalable models of asynchronous systems demonstrate that the proposed algorithm can successfully find the optimal solution. The remainder of this paper is organized as follows. Next section presents the flow of asynchronous synthesis. Section 3 presents the background and related work. Section 4 describes Timed Petri-Nets as the dominant performance analysis model. Section 5 discusses the Performance Optimization algorithm in detail while Section 6 gets on with the results and analysis. And finally Section 7 concludes the paper.

Manuscript received September 9, 2009.

Behnam Ghavami is with the Amirkabir University of Technology, Tehran, Iran (corresponding author to provide phone: +98-21-64542701; e-mail: ghavamib@ aut.ac.ir).

Hossein Pedram, was with Rice Amirkabir University of Technology, Tehran, Iran. He is now with the Department of CEIT, (e-mail: pedram@aut.ac.ir).

II. Synthesis of Asynchronous Circuits

Asynchronous circuits rely on exchanging local requests and acknowledge signaling for the purpose of synchronization. While the first generation of asynchronous synthesis tools were mainly focused on control synthesis which made them inapplicable for practical circuits, the new generation of synthesis methods target the use of pre-designed asynchronous buffer templates [9][17]. Figure 11 shows the synthesis flow of an asynchronous synthesizer. The design process usually starts with a high level circuit specification. Verilog-CSP, standard Verilog powered by PLI [18], is considered here as the specification language. `READ and `WRITE blocking macros encapsulate asynchronous data communications in Verilog-CSP.

By applying a set of functions preserving transformations to the high level specification, Data Driven Decomposition[19] method divides the original circuit to a set of communicating fine grain pipeline templates. Different types of pre-designed templates are developed to address different requirements [20]. Decomposition can potentially improve the overall performance by introducing more concurrency. The resulted circuit then can be directly implemented down to the layout or some optimization such as clustering may be done to improve the overall performance of the circuit.

III. Slack Matching: Background

Asynchronous circuits are typically designed as collections of independent processes that share values exclusively through message passing. Instead of a global clock, local handshakes are used for synchronization. It has been observed that the cycle time of an asynchronous system can be greater than that of the slowest module in the system running on its own.

While it is well-known that good pipelining design styles in asynchronous circuits are critical to reduce the asynchronous control circuit overhead (e.g.[5][5]), it is less well-known how to balance pipelines to optimize the performance. Unlike synchronous designs, one of the benefits of a large class of asynchronous designs (including so called slack elastic asynchronous designs [11]) is the ability to add pipeline buffers to the design without changing the input/output functionality of the circuit. This feature is particularly useful in asynchronous interconnects in which long wires can be pipelined. However, the addition of pipeline buffers is also essential in high-speed asynchronous designs in order to balance the pipelines and avoid pipeline stalls. One reasonable objective for this pipeline optimization problem is to identify the minimal number of additional pipeline buffers to satisfy a given performance constraint, thereby implicitly minimizing area and power for a given performance.

The static slack of a pipeline is the maximum number of messages that can be inserted into the pipeline (with none being removed) before deadlock occurs. If a pipeline of n instances of a process has static slack of n/2, the process is said to be a half buffer. If a pipeline of n instances of a process has static slack of n, the process is said to be a full buffer. Our work applies on full buffer systems. Stalls that are caused by miss-matches in communication rates are a major performance obstacle in pipelined circuits. If the rate of data production is faster than the rate of consumption, the resulting design performs slower than the case where the communication rate is matched. This can be remedied by inserting pipeline buffers to temporarily hold data and allowing the producer to proceed if the consumer is not ready to accept data. This type of modification is a well-known optimization technique referred to as slack matching [12]. This technique reduces a system's cycle time by inserting buffers into communication channels. We hope to be able to adjust the property of slack in order to improve performance by maximizing available parallelism between various paths. The goal is to present a method for automatic insertion of slack matching buffers in order to improve performance. Slack matching has been compared to the retiming problem in synchronous design. In slack matching, the structure of the circuit does not change significantly (like retiming). The key difference between the two problems, however, is that in the synchronous domain an initial assignment of latches must be given and the number of latches along any cycle must not be changed. In contrast, in asynchronous systems, the initial latch assignment is not necessary and the correctness requirements on the number of latches along a cycle are different. Interestingly, the basic version of retiming can be solved in polynomial time, while the optimal solution for the slack matching problem has been shown to be NP-complete[8].

Recently, many techniques have been proposed to performance optimization of asynchronous circuits. A related work is how the processes should be decomposed in order to ensure both local and global cycle times are optimized [5][8][9]. Kim showed that this problem is NP-complete and it was shown a branch-and-bound algorithm was presented in[5][8]. Wong et al. presented. A heuristic clustering algorithm to increase the system performance was presented in[9].

Recent investigations cast the slack matching problem as an integer programming problem and use generic IP solvers to get a target cycle time[13][14]. Some researchers (e.g.[8] [5][15]) were solved the problem using a branch and bound algorithm to obtain a target cycle time. Alternative work[15][16] use leveraging protocol knowledge in slack matching and present a heuristic that uses knowledge of the communication protocol to explicitly model these bottlenecks. They introduced an iterative algorithm to remove these

bottlenecks by inserting buffers. Modeling in most of these techniques are complex, and sometimes appear to be unsolvable for large circuits. In addition, there are some other problems for exploiting these techniques, such as handling hierarchical circuit models and modeling systems with choice. Because of these restrictions, it is difficult to apply these techniques for optimization of real asynchronous systems.

The goal of this work is to characterize where slack matching buffers can improve performance and then to automatically insert them into the netlist. Systems are often designed with a target cycle time, while in this paper we study the problem of adding buffers to a slack elastic system to reduce the overall cycle time of the system with area and power overhead considerations. We introduce an abstract template model for handling slack matching problem that is comprehensible and capable to handle hierarchical circuit models and non-deterministic systems with a little extension. In addition, our technique is scalable for any size of problem and gives an acceptable solution in a reasonable time. Therefore, our technique is applicable on real systems.

IV. Performance Model

Our circuit model is based on Petri-Nets which have been already used as a description of concurrent systems[21]. The main advantage of our model is that it can be used for performance analysis in addition to simulation. This sub-section covers the necessary PetriNet background for this paper.

A Petri net is a four-tuple $N = (P, T, F, m_0)$ where P is a finite set of places, T is a finite set of transitions and $F \subseteq (P \times T) \cup (T \times P)$ is a flow relation, and $m_0 \in \aleph^{|p|}$ is the initial marking. A Petri net is usually represented as a bipartite graph in which P and T are the nodes. For any two nodes x and y, if $(x, y) \in F$ then there is an arc from x to y. A marking is a token assignment for the place which represents the state of the system. Formally, a marking is a $|p| -$ vector m, where the number of tokens in place p under marking m, denoted by m(p), is a natural number. We say for an element $x \in P \cup T$, that $\bullet x$ is the preset of x defined as $\bullet x = \{y \in P \cup T \mid (y, x) \in F\}$ and $x \bullet$ is the postset of x defined as $\bullet x = \{y \in P \cup T \mid (x, y) \in F\}$. A transition t is enabled at marking m if each place in $\bullet x$ is marked with at least one token. When a transition t is enabled, it can fire by removing one token from each place $\bullet x$ and adding one token to each place $x \bullet$. A marked graph (MG) is a type of Petri net in which each place has at most one input and output transition, i.e., $|\bullet p| \le 1 \wedge |p \bullet| \le 1, \forall p \in P$. A timed MG is a MG in which delays are associated with transitions. We instead associate a delay with each place d(p) because the models are more intuitive for our purposes and less constrained. That is, every MG with delays associated with transitions can be translated into one in which this delay is instead associated with all places in its pre-set $\bullet t$. Because each place has at most one transition in its post-set no ambiguity in place delays exists. The semantics of a timed MG state that a transition t fires after tokens in places $p \in \bullet t$ reside in p at least d(p) time units. A cycle c is a sequence of places p1 p2… p1 connected by arcs and transitions whose first and last places are the same.

The cycle metric (CM(c)) is the sum of the delays of all associated places along the cycle c, d(c), divided by the number of tokens that reside in the cycle, m(c), i.e., $CM(c) = d(c) / m_0(c)$. The cycle time of a MG is defined as the largest cycle metric among all cycles in the timed MG, i.e., $\max \forall c \in C[CM(c)]$, where C is the set of all cycles in the timed MG[23]. The intuition behind this well-known result is that the performance of any computation modeled with a timed MG is dictated by the cycle time of the timed MG and thus the largest cycle metric.

A. Simple Templates

Different types of models can be developed for different kind of asynchronous templates as the target implementation style. We have developed a class of models that fully supports full buffer templates [20]. In the proposed model, Timed PetriNet (TPN), the detailed structure of the original circuit including the handshaking channels are preserved which is the key that makes our model suitable for applications in which precise performance modeling is required. One important application is the slack matching[20] problem which our model can be readily applied to for addressing performance metrics.

The simplest form of an empty full buffer is a simple buffer that only reads a value from its input and writes it to its output. This behavior can be modeled simply as shown in **Error! Reference source not found.**.

Transition tW is analogous to the write statement while place pWa emulates the write acknowledge. Similarly pR can be seen as the dual for read statement while tRa is the corresponding acknowledge. We considered delays on the place therefore forward delay and backward delay can be put on pR and pB. Full buffer is the same. But the token is at the

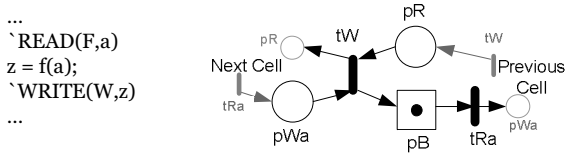

Figure 1-A simple empty buffer modeled using Petri-Nets

pR place instead of pB place in the empty buffer. This model is very similar to FCBN model presented in[3] and the only

difference is that we added tRa. The reason for this is that the used definition of the hierarchical Petri-Nets has a restriction on the input and output ports; all outputs must be transitions and all inputs must be places. This convention ensures that unwanted choices or merged constructs cannot be formed when connecting Petri-Net modules to each other. The model for simple buffer can be extended to more reads and more writes as can be seen in Figure 2. It is notable that this form of modeling needs the least amount of transitions and places, as it requires only one place and one transition for each Read or Write basically. Additional constructs are needed when special functions like forking or decision making are required.

For choice support in this model, we must add choice places for conditional reads and writes and two transitions for a choice place that identifies the state of condition (taken or not). Our methodology for buffer assignment is not restricted to deterministic models and can be applied on circuits containing choices. This extension has not been used in this work and is under investigation.

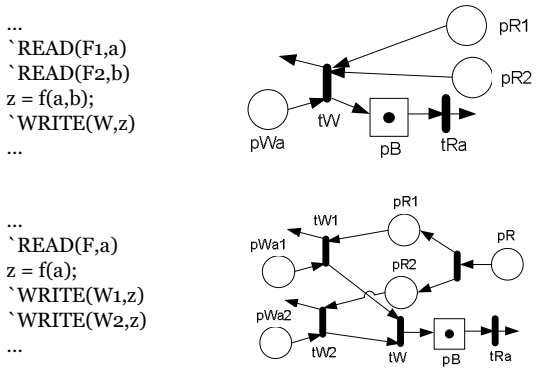

Figure 2- Model for more than one read buffer above and for more than one write below.

B. *The Intuition*

The performance of the network is represented by modeling the performance of the channels as they are connected to the templates. In our model, the local cycle time is attributed to the output channels to which the templates are connected. d(f) represents the forward latency of a channel while the corresponding d(b) represents the backward latency of the channel. Intuitively, the forward latency represents the delay through an empty channel (and associated cell) and the backward latency represents the time it takes for the handshaking circuitry within the neighboring cells to reset, and enabling a second token to flow. The cycle metric associated with the cycle f and b, c (f o b), represents the local cycle time of the channel, and is represented by the following relationship: $c(f\ o\ b\ ) = d(f) + d(b)\ /[m(f) + m(b)]$

As an example, we illustrate two channels in Figure 1. The pR place represents forward place and the pB presents backward place. The pWa place has zero delay always and can be ignored.

$c(f\ o\ b) = d(pR) + d(pB) + d(pWa)\ /\ [m(pR) + m(pB) + m(pWa)] = d(pR) + d(pB)$.

The cycle time of the circuit is captured by the maximum cycle metric of the corresponding TPN. The throughput of the circuit is the reciprocal of this value. Often additional buffers (also known as slack) must be added to the model to balance the pipelines, and thereby improve the cycle time. We model the addition of slack between cells by creating new transitions, places, and arcs that represent buffers and their corresponding channels.

As an example, consider a homogeneous non-linear pipeline fork-join channel structure in which there is three buffers in one path and one buffer in the other path. The proposed Timed Petri-Net model of this structure is illustrated in Figure 3. Notice also that the forking transition, which represents the leaf-cell fork, has both of its output channels initially full, whereas all other channels are initially empty, indicating this leaf-cell is the only token buffer in the system. The cycle yielding the maximum cycle metric consists of the forward latency path through the long fork and the backward latency path of the short fork. It has a cycle metric of $(1+1+1+1+4+0+4+0)\ /2 = 12/2 = 6$.

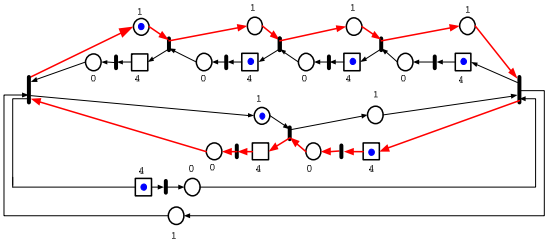

Figure 3- Timed PetriNet model of un-balanced fork-join pipeline

If a second buffer is inserted in the short forked path, as illustrated in Figure 4, the worst-case cycle metric reduces to $(1+1+1+1+4+0+4+0+4+0)\ /\ 3 = 16\ /\ 3 = 5\ +1/3$

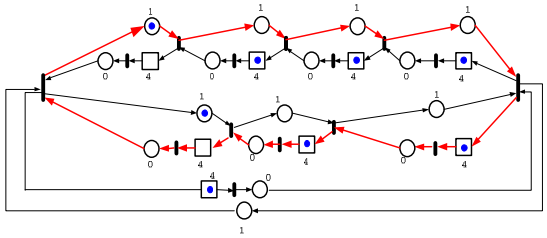

Figure 4- Adding a pipeline buffer can improve the maximum cycle metric of a TPN

An intuitive understanding of the slack matching problem can be obtained by analyzing the unbalanced fork-join pipeline above [13]. In particular, one key observation is that if tokens arrive at a join stage at different times, the early token will stall and the stall will propagate backwards, and slow the entire system down.

An intuitive understanding of the slack matching problem can be obtained by further analyzing the unbalanced fork-join pipeline above. In particular, one key observation is that if tokens arrive at a join stage at different times, the early token

will stall and the stall will propagate backwards and slow the entire system down.

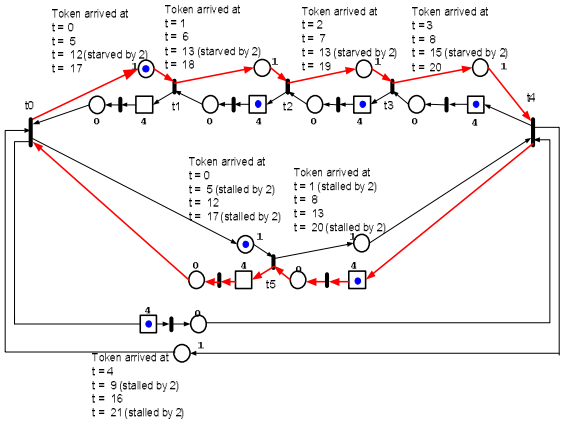

Figure 5. Illustration of how an unbalanced fork-join pipeline causes pipeline stalls

We illustrate this intuition by annotating the TPN in Figure 3 with token arrival times, as illustrated in Figure 5. The unbalanced nature of the pipeline causes the first token processed by the lower "short" fork, stage t5, to stall for 2 time units while waiting for the top "longer" fork to propagate its first token. t5's backward latency is 4, and the output channel of t5 can accept a second token no sooner than t = 8, i.e., t5 fires a second time at t = 8. This means that the second token that arrives at the input channel of t5 is stalled for 2 time units during this reset period. This stall delays when t0 generates the third token by 2 time units, thus starving t1 by 2 time units. This propagation of starving/stalling continues and every other token that is processed by every channel is delayed by 2 time units. Consequently, instead of each channel operating at a peak local cycle time of 5 time units, they all operate at an average cycle time of 6 time units. The intuition gained from this example is as follows: for the global cycle time to be equal to the local cycle time, all tokens must arrive at all join stages at the same time.

The above intuition is a necessary condition for the global cycle time to be equal to the local cycle time of the channels in a homogeneous pipeline in which all channels have the same cycle time. However it is not sufficient. To clarify, consider another important case when a token can propagate around a loop of leaf cells faster than the local cycle time. In this case a token will be stalled while the local channel resets.

Like the above case, this stall will propagate backward and increase the global cycle time. In particular, it is the backward propagation of empty places for tokens to move into that becomes the throughput bottleneck. This is illustrated in Figure 6.

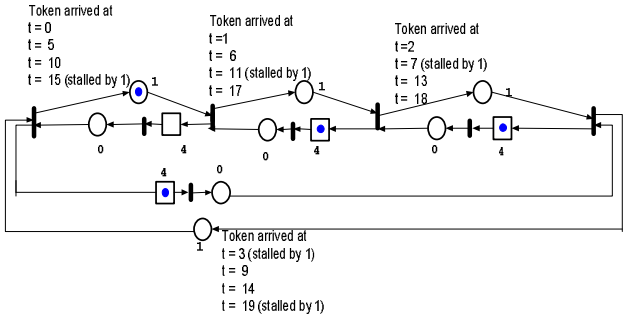

Figure 6- Illustration of how a short loop can cause pipeline stalls

Alternatively, if the latency around a one-token-loop is larger than t, the cycle time will necessarily be greater than t. Consequently, another necessary condition is that for one-token-loops the propagation delay along every cycle is equal to the local cycle time. Multi-token loops, with multiple token buffers, yield a generalization of this condition. Specifically, for the global cycle time to equal the local cycle time, the latency along any m-token-loop must be m t.

This intuition must be modified to consider the case in which one channel resets faster than others, i.e., has smaller backward latency. If a token is stalled in this channel by a small amount, the channel may still be able to reset in time to accept the next token within the desired global cycle time. If the forward latency + stall time + backward latency = the desired global cycle time, then this stall will not act as the bottleneck the design. The difference between desired global cycle time and the local cycle time has been called free slack [13].

V. Automatic Slack Matching

In this section, we describe a heuristic slack matching algorithm for performance optimization of asynchronous circuits. Our algorithm is based on the simulated annealing (SA) method [23]. Given a feasible PetriNet structure, we perturb (Insert/Remove Buffer) it to obtain another feasible PetriNet structure through a set of pre-defined SA operations. After perturbation, we perform the suitable cost function to obtain a structure with respect to the precedence constraints. Finally a performance analysis along with simulation of the circuit is performed to evaluate the quality of the solution.

In simulated annealing, the physical process of cooling a liquid to its freezing point to obtain an ordered structure is simulated at several such temperature steps by letting the system reach equilibrium at each temperature. An objective function is associated with the energy of a physical system and system dynamics are imitated by random local modifications of the current solution. A feasible solution corresponds to a system state and an optimal solution corresponds to a state of minimal energy. Thus, key parameters in any formulation are the initial temperature T, Cost function and the number of iterations at each step. Figure 7 shows the overall structure of the introduced SA algorithm. Subsequent sections describe institution of this

algorithm in detail.

A. Initialization

Since the performance of an asynchronous circuit depends on the number of tokens and their distribution in the circuit, we must determine the situation of tokens in the circuit. The first step of this algorithm is to initialize a circuit (in PetriNet model) with a token assignment. The token assignment is the stable state of tokens in the circuit and is determined by simulating the movement of tokens from an initial token assignment. In our model, initial tokens are located in the primary inputs of the circuit. We use a PetriNet Simulator[10] to simulate our PetriNet model of a circuit to obtain the stable state of tokens. Furthermore, this simulation provides such information as throughput which is needed at next steps. The time complexity of the used PetriNet Simulator is $O(P^3)$, where P is the set places in the PetriNet model.

B. Perturbation

For the slack matching of QDI circuits, we introduce a type of SA operations. At iteration step we insert some buffers in the circuit and delete some buffers from the circuit randomly.

```
-------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------
T = T initial;
While (T > T final)
{
    for (i=0; i<Inner Iteration; i++)
     {
        Old cost = cost ();
        Perturbation ();
        Newcost = cost ();
        delta = new cost - old cost;
        if (delta>0)
         {
             ran = rand () / RAND_MAX;
            if (exp (-delta/T) > ran)
             {
               Undo Perturbation ();
             }
         }
      T = Température (T);
     }
}
-------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------
```

Figure 7- Simulated annealing algorithm

1) Buffer Insertion

At this step the buffers are placed in different channels of the circuit and all needed communications are established. The Run time of SA depends on the number of iterations of the algorithm, and will be prolonged if we add buffers into our circuit one by one. In order to decrease the run time, we add some pipeline buffers into the circuit simultaneously, per iteration. But the number of the added buffers per iteration must be calculated initially. We first run algorithm for a test case with different numbers of buffers added per iteration and then calculate the run time. The result is shown in Figure 8. As shown, in this case we must add 9 buffers per iteration to have the shortest run time. We repeat this process for a few test cases and draw the diagram in Figure 9 that shows the best number of added buffers per iteration with respect to the sizes of the test cases. The shortest run time determines the number of buffers to be added per iteration. At this step, assumed cooling functions of the algorithm for the all of the test cases are the same. As shown in Figure 9, the relationship that describes the best number of the added buffers per iteration with respect to the size of the circuit (number of nodes) is y=0.0454x- 0.1386. So, we use this relationship to determine the number of the buffers that must be added to the circuit per iteration. As mentioned, the number of buffers added per iteration only affects the run time; this means that we could run the algorithm with addition of one buffer per iteration; but this takes longer times for large circuits.

2) Buffer Removing

The Perturbation step must have a deletion step. By this way, we let the algorithm remove any buffer that had been inserted in the circuit. At this step we delete one buffer from the circuit. This buffer can be any buffer that we added to the circuit from the start of the SA algorithm. We form a list of buffers that have been added to the circuit, and randomly select a buffer from this list and delete that buffer. This procedure operates mostly at high temperatures rather than low temperatures in order to let the algorithm to remove any buffer that has been added at high temperatures.

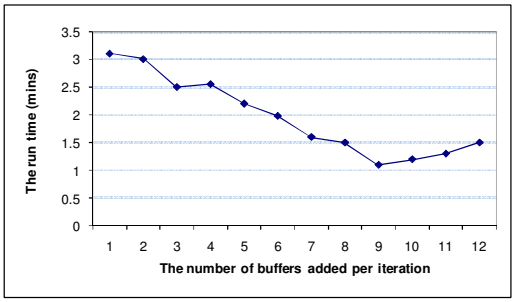

Figure 8 - The run time diagram for different numbers of buffers added per iteration

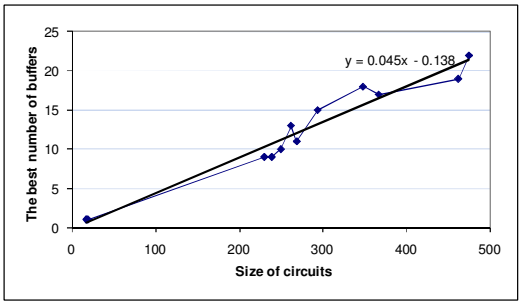

Figure 9- The best number of buffers added per iteration

3) Change location

Location of the buffers can be changed to find better location for every buffer that has been added to the circuit.

This operation helps the algorithm to find a better solution for this optimization problem.

C. Cost function

Our goal is to optimize Performance of the circuit with minimum penalties of area and power. So, parameters of the cost function are these terms:

1) Performance: Defined as throughput in asynchronous circuits. As mentioned, throughput is the inverse of the cycle time. The cycle time of a deterministic pipeline is defined as the largest cycle metric in its marked graph representation[2][3]. The cycle metric of each cycle is the sum of the delays of all associated transitions (or places) divided by the number of the tokens that can reside in the cycle. We use Karp method[23] to find the largest cycle metric of the circuits. This method creates a vertex for each marked place and edges between two vertices if there is a path between their corresponding places. The weight of an edge is the largest sum delay of such paths. Karp's algorithm will then find the maximum mean cycle.
2) Area: A good approximation is determined by the number of cells in the circuit and their channels (transitions and places in the PetriNet structure). So, Sum of the nodes and their connections (places and transitions) is a good metric for area estimation. We calculate the area of a new circuit by eliminating the old cells (after the buffer deletion step) and channels from the area of the old circuit and augmentation of new buffers (after the buffer insertion step) and channels to the area of the old circuit.
3) Power: Power in our work is estimated by transition counting presented in[25] that is applied on circuits after the decomposition step in the synthesis flow.

Figure 10 shows the area, power and performance increasing with respect to the size of an asynchronous pipeline with a constant token number. As shown in the figure, area and power increase linearly. But the performance increases initially by adding some buffers, and decreases afterwards by adding more buffers. Adding more buffers must stop when reaching the maximum point.

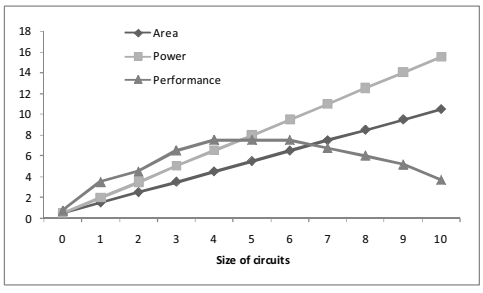

Figure 10- Area, power and performance of a pipeline

As the result, the cost function Φ that is optimized in our algorithm is given by

$$\Phi = \frac{\alpha(\mathrm{A}/\mathrm{A_i})^{\delta} + \beta(\mathrm{P}/\mathrm{P_i})^{\varepsilon}}{\gamma(\mathrm{T})^{\zeta}} \quad (1)$$

Where T is the throughput method of the circuit, A is the total area of cells and communications circuitry, and P is the estimated power using transition counting [25]. A_i and P_i are initial area and power respectively. α, β, γ, δ, ε, and ζ are user-specified constants and are used for normalization. This cost function allows the user to control different parameters on the inclination. Since we must improve performance, we penalize the excessive area and power. When SA minimizes the cost function, it automatically minimizes the penalty term. Thus, we can automatically optimize the circuit. As indicated, adding the buffers to the circuit may or may not improve the throughput. The penalty terms are area and power. If the perturbation is not accepted, we will undo the PetriNet structure changes.

VI. Experimental Result

Figure 11 shows the synthesis flow of an asynchronous synthesizer. For optimization purposes, the output of the decomposition block which is in CSP format serves as input to a model generator that transforms a Verilog-CSP specification to its PetriNet equivalent. Then, our simulator runs the PetriNet circuit and provides the dynamic information of the original circuit such as throughput and token assignment. Our optimizer includes a static analyzer in order to provide static information that is needed in cost function evaluation, and a buffer assignment engine that adds buffers to the circuit utilizing a heuristic method that uses information resulted in the previous stage. The optimizer runs until the circuit becomes unstable. Then, the simulator runs the resulted circuit again. The resulted throughput of the circuit is compared to its original throughput, which verifies that the optimization operation was successful.

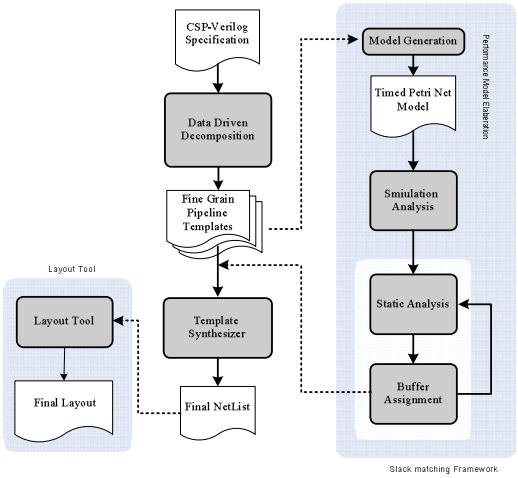

Figure 11 -Performance Evaluation Framework and its interface with asynchronous synthesizer

The resulted circuit in PetriNet format is converged to a

slack matched version of the input circuit with Verilog-CSP specification and serves as input to the next step of the synthesis flow in order to continue the rest of flow. Figure 11 shows the general structure of the proposed performance optimization scheme and its interface with a generic asynchronous synthesis flow.

We used the well-known sequential benchmark of synchronous circuits referred to as ISCAS'89, to better explore the usefulness of the proposed algorithms. The procedure is as follows. In the beginning, our synthesis tool automatically translates the circuit (in verilog) to its PetriNet equivalent. At this step, the number of nodes of a synchronous circuits increase, because all primary inputs and outputs are considered nodes in our model. Also, we can not have multi-fanout wires in asynchronous systems, so they are transformed to reconvergent fanouts. Inputs and outputs of the circuit are connected to each other in PetriNet structure to form a closed loop system. Initially all tokens placed in input nodes,. Then, the Simulator moves tokens through the PetriNet network until it gets the stable state. Next, our algorithm slack matches the translated circuit. In our case, cost function coefficients in the equation (1) have been specified on our goal to lay stress on performance optimization in comparison with the area and power overheads. α and β have been supposed 40% of the γ. Also δ and ε have been set to 70% of ζ. Now, the resulted circuit can be returned to the synthesis flow in order to complete the synthesis process. Persia[17] is a QDI synthesis toolset that is employed to synthesize our benchmarks.

For simplicity, we set the delay of each original or added buffer to 1 and the backward latency to 4, giving a local cycle time of 5. The results are shown in Table 1. Column 2 identifies the number of gates in the synchronous circuit. Column 3 shows the number of nodes in the PetriNet model of asynchronous circuit. Column 4 depicts the circuit throughput before adding slack buffers. Column 5 identifies the number of buffers that algorithm insert to the circuit. Column 6 shows the throughput of the resulted circuit after buffer insertion. Column 7 shows the improvement of throughputs mentioned. Column 8 identifies area overhead resulted from this algorithm. As demonstrated by experimental results, our proposed technique can achieve on average 38% optimization for performance with 24% increased area as its cost.

A comparison is presented between our method and another method that was based on ILP formulation to minimize the cost of additional buffers in order to achieve a performance target [13]. The results are shown in column 9. Our method represents 57% improvement in reducing the area overhead; also it is much simpler than ILP formulation. As shown in the table 1, our algorithm is better applied on the large circuits that need too many buffers for slack matching optimization. The heuristic algorithm with a number of buffers less than ILP formulation can improve the performance of such circuits. C1355 test case is an example of this problem. ILP formulation insert 1800 buffers to the circuit but our algorithm by obtain 38% performance improvement by the insertion of 302 buffers to the circuit. On the other hand, ILP formulation is better applied on the circuits that need a few buffers for slack matching. The ILP formulation solves the optimization problem for the S820 circuit by addition of only one buffer to the circuit but our algorithm inserts 55 buffers to the test case.

TABLE 1. PERFORMANCE OPTIMIZATION FOR ASYNCHRONOUS CIRCUITS

Circuit	Number of Gates	Number of Nodes	Throughput	Number of Buffers	Resulted Throughput	Percentage of Throughput	Percentage of Area overhead	Percentage of Area improvement compare to an ILP-based method
c17	6	17	0.366	5	0.448	22.40	29.41	-21.72
c432	160	250	0.281	79	0.38	35.23	31.6	205.12
c499	202	262	0.203	65	0.337	66.01	24.81	229.89
c880	383	509	0.185	121	0.279	50.81	23.77	53.84
c1355	546	713	0.192	302	0.266	38.54	42.36	248.43
c1908	880	1145	0.241	327	0.343	42.32	28.56	116.21
c7552	3512	4572	0.287	1037	0.399	39.02	22.68	82.62
s27	13	19	0.314	8	0.458	45.86	42.11	2.33
s382	181	230	0.297	37	0.402	35.35	16.09	-11.84
s400	186	239	0.24	58	0.312	30	24.27	-20.15
s420	234	294	0.196	86	0.268	36.73	29.25	64.42
s444	202	269	0.211	71	0.298	41.23	26.39	8.69
s641	398	462	0.272	42	0.342	25.74	9.09	-8.66
s713	412	475	0.208	64	0.277	33.17	13.47	-13.05
s820	294	367	0.348	55	0.47	35.06	14.99	-14.68
s832	292	348	0.339	52	0.441	30.09	14.94	-14.64

VII. Conclusion

In this paper, an efficient method for slack matching of asynchronous systems is presented. The Decomposed circuit in the CSP-Verilog format is used to generate a Timed Petri-Net model which captures the dynamic behavior of the system. To obtain a high precision performance optimization, our scheme first performs a dynamic performance analysis to compute the situation of tokens. The proposed method is based on Simulated Annealing algorithm. The experimental results on a set of ISCAS benchmarks show that our proposed technique can achieve on average 38% optimization for performance, while there is 24% are penalty. Considering the fact that better buffer assignment can lead to better optimization with less area penalty, the future work is to improve the static analyzer. Also this work can be extended to support choice in the system.

References

[1] Jan M. Rabaey, Anantha Chandrakasan, Borivoje Nikolic, "Digital Integrated Circuits", a design prespective 2nd Edition, McGraw-Hill, 1988.

[2] Alain J. Martin, Mika Nyström, Karl Papadantonakis, Paul I. Pénzes, Piyush Prakash, Catherine G. Wong, Jonathan Chang, Kevin S. Ko, Benjamin Lee, Elaine Ou, James Pugh, Eino-Ville Talvala, James T. Tong, Ahmet Tura: The Lutonium: A Sub-Nanojoule Asynchronous 8051 Microcontroller. ASYNC 2003.

[3] Steven M. Burns, Alian J Martin, "Performance Analysis and Optimization of Asynchronous circuits", Advanced Research in VLSI conference, Santa Cruz, CA, March 1991.

[4] Peter A Beerel, Nam-Hoon Kim , Andrew Lines, Mike Davies, "Slack Matching Asynchronous Designs", Proceedings of the 12th IEEE International Symposium on Asynchronous Circuits and Systems, Washington, DC, USA,2006.

[5] J. Sparso and J. Staunstrup, "Delay-insensitive Multi-ring Structures", VLSI J. Integr.,vol. 15, no. 3, pp. 313-340, oct. 1993.

[6] W.C. Chou, P. A. Beerel, and K. Y. Yun, "Average-case Technology Mapping of Asynchronous Burst-mode Circuits", IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst., vol. 18, no. 10, pp. 1418-1434, oct. 1999.

[7] R.M. Badia and J. Cortadella, "High-level Synthesis of Asynchronous Systems: Scheduling and Process Synchronization", in Proc. European Conf. Design Automation (EDAC). Paris, France: IEEE Comput. Society Press, Feb. 1993, pp. 70-74.

[8] S. Kim and P. Beerel, "Pipeline Optimization for Asynchronous Circuits: Complexity Analysis and an Efficient Optimal Algorithm", In Proc. International Conference on Computer-Aided Design, 2000.

[9] C G. Wong and Alain J. Martin, "High-Level Synthesis of Asynchronous Systems by Data Driven Decomposition", Proc. Of 40th DAC, Anneheim, CA, USA, June 2003.

[10] M. Najibi, M. Niknahad, H. Pedram, "Performance Evaluation of Asynchronous Circuits Using Abstract Probabilistic Timed Petri Nets with Choice", ISVLSI 2007.

[11] R. Manohar and A.J. Martin. "Slack Elasticity in Concurrent Computing", In J. Jeuring, editor, Proc. 4th International Conference on the Mathematics of Program Construction, Lecture Notes in Computer Science 1422, Pages 272-285. Springer Verilog, 1998.

[12] A. J. Martin, A. Lines, R. Manohar, M. Nystrom, P. Penzes, R. Southworth, U. Cummings, and T. K. Lee, "The Design of an Asynchronous MIPS R3000 Microprocessor", presented at Proc. 17th Conference on Advanced Research in VLSI, 1997.

[13] Beerel, Peter A., et. Al., "Slack Matching Asynchronous Designs", In Proc. Of the 12th IEEE International Symposium on Asynchronous Circuits and Systems, IEEE Computer Society Press, 2006.

[14] Prakash, Piyush, and Martin, Alain J., "Slack Matching Quasi Delay-Insensitive Circuits", In Proc. Of the 12th IEEE International Symposium on Asynchronous Circuits and Systems, IEEE Computer Society press, 2006.

[15] Fazel, Kenneth, et al., "Performance Enhancement in Phased Logic Circuits Using Automatic Slack-matching Buffer Insertion", In Proc. of the 2004 Great Lakes Symposium on VLSI (GLSVLSI'04), Boston, MA, ACM Press, 2004

[16] Girish, Venkataramani, and Seth C. Goldstein, "Leveraging Protocol Knowledge in Slack Matching", In Proc. of ICCAD'06, San Jose, CA, November 5-9, 2006.

[17] Ghavami B, Pedram H, High performance asynchronous design flow using a novel static performance analysis method, Comput Electr Eng (2008), doi:10.1016/j.compeleceng. 2008.11.025

[18] Arash Seifhashemi, Hossein Pedram, "Verilog HDL, Powered by PLI: a Suitable Framework for Describing and Modeling Asynchronous Circuits at All Levels of Abstraction", Proc. Of 40th DAC, Anneheim, CA, USA, June 2003.

[19] C G. Wong "High-Level Synthesis and Rapid prototyping of Asynchronous VLSI Systems" PhD's thesis, Caltech institute of technology, 2004.

[20] Andrew Matthew Lines. "Pipelined asynchronous circuits", Master's thesis, California Institute of Technology, Computer Science Department, 1995 CS-TR-95-21.

[21] Sangyun Kim, "Pipeline Optimization for Asynchronous circuits", PHD Thesis, University of Southern California, August 2003.

[22] J. L. Peterson, "Petri Net Theory and the Modeling of Systems", Englewood cliffs, N. J.: prentice-Hall, 1981.

[23] R. M. Karp, "A Characterization of the Minimum Cycle Mean in a Diagraph", Discrete Mathematics, 23: 309-311, 1978.

[24] S. Kirkpatrik, C. D. Gellatt, Jr. and M. P. Vecchi, "Optimization by Simulated Annealing", Science 220(4598), 1983.

[25] Ghavami B et al., An EDA tool for implementation of low power and secure crypto-chips, Comput Electr Eng (2008), doi:10.1016/ j.compeleceng.2008.06.014.

Behnam Ghavami was born in Esfarayen in North Khorasan of Iran, on April 9, 1982. He received his BS degree in Computer Engineering from Bahonar University in 2005. He graduated from the Tehran Polytechnic University. He is a research assitant of Asynchronous Design Laboratory in the same school.

Hossein Pedram Received his BS degree from Sharif University in 1977 and MS degree from Ohio State University in 1980, both in Electrical Engineering. He received his PhD degree from Washington State University in 1992 in Computer Engineering.

Dr Pedram Has served as a faculty member in the Computer Engineering Department al Amirkabir University of Technology since 1992. He teaches courses in Computer architecture and distributed systems. His research interests include innovative methods in computer architecture such as asynchronous circuits, management of computer networks, distributed systems, and robotics.