A Fast Timing Separation of Events Algorithm for Concurrent Systems

Pavlos Aimoniotis, Nikolaos Xiromeritis and Christos Sotiriou
University of Thessaly, EECE Department, Greece.
{paimoniotis, nxiromeritis, chsotiriou}@e-ce.uth.gr

***Abstract*—In this work, we present a fast, polynomial time implementation of the Timing Separation of Events (TSE) Algorithm, with complexity $O(E \times (V + E))$. TSE computation is a fundamental problem in the analysis of event-driven, asynchronous systems, when uncertainty is present, and event delays are specified as [min, max] intervals. The maximum or minimum TSE may be needed, based on the timing analysis required. In this work, we present a novel approach to solving the TSE computation problem, which utilises: (1) the Primal-Dual Method Algorithm of [1] to compute the system period and critical cycle, as well as annotate timing offsets to events, based on the minimum delay values, and (2) an unfolding scheme, on the event graph, to compute the maximum delay between source and target. We show that only a maximum of three unfoldings are necessary, based on the relative delays between source and target events. We present results on a set of classical TSE examples. Our results are correct and identical to [2], but our approach resolves the issue of deciding how many unfoldings are needed to achieve periodic behaviour, and is significantly faster than all other methods.**

***Index Terms*—EDA, Static Timing Analysis, Asynchronous Systems, Concurrent Systems**

I. Introduction

EDA tools are the main reason of hardware design improvement in recent history. EDA tools perform various complex procedures, during implementation, giving designers the opportunity to focus on design issues. In other words, it is a completely independent tool that comforts the implementation. Throughout the years various tools have been developed focusing on specific aspects of synchronous design. Those tools are not suitable for asynchronous design. Asynchronous design is not used today due to lack of a complete and mature asynchronous design flow. We find ourselves obliged to contribute on circuit's timing, the most important factor in design and implementation process. Asynchronous Static Timing Analysis (ASTA) requires new algorithms, as synchronous Static Timing Analysis are based on Directed Acyclic Graph (DAG) model for the circuit timing graph. Thus, when a synchronous STA engine is provided with a cyclic circuit, cycles are cut, leading to significant loss of timing information.

Our contribution is a fast, polynomial time TSE algorithm. Prior works include [2]–[4], where [2] is of not polynomial complexity, and depends on an amount of unfoldings, and [3] is not applicable to cyclic event graphs. Out of prior works, the best performance is achieved by [4], *i.e.* $O(n^3)$. However, our implementation is of better computational complexity, *i.e.* $O(E \times (V + E))$. Our implementation is based on a sequence of five steps, to compute the maximum TSE. First, (1) the minimum delays critical cycle of the Event Timing Graph is computed, using Burn's Primal-Dual Method [1]. Then, (2) the critical cycle, and remaining graph, is unfolded twice, and (3) the minimum occurrence times for events prior to the source are computed. Next, (4) the longest paths from the source, or pre-source events, to the target event are computed, and (5) the difference in timing occurrence computed between the target and source yields the maximum TSE result.

II. Background

A. Asynchronous Timing Models

Asynchronous control circuits are concurrent systems, thus their behavior and specification is often modeled by concurrent models, such as Petri Nets (PTnets) [5], Signal Transition Graphs (STGs) or Event Rule systems (ER). In this work, we use a both PTnets and ER systems, both of which are general models, and conveniently model causality, concurrency and specify behaviour in a closed loop model, *i.e.* the circuit along with its environment. ER systems are STGs, with a specified delay interval on their timing arcs. For example, for an edge $a \rightarrow b$, with a delay range $[d, D]$, for event b to occur, event a must occur first, and b will occur after it some time between d and D. To create and simulate our asynchronous circuit specification modelled as a PTnet, we used the Workcraft framework [6], and perform a PTnet to ER system transformation [7].

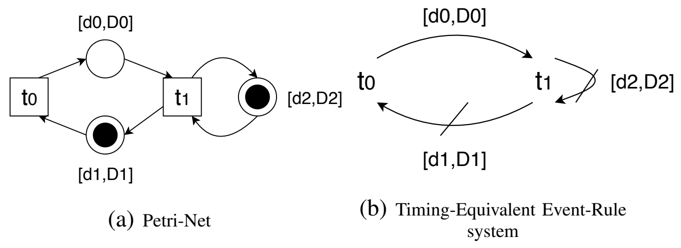


Fig. 1: PTnet to ER Transformation

B. Timing Analysis using Unfoldings

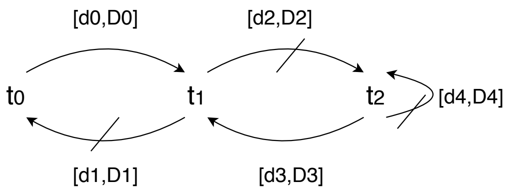
The transitive and repetitive timing behaviors of an PTnet or ER System may be modeled, by generating its unfolded, acyclic version. A key parameter of the unfolded graph, is the number of unfolding iterations. In the unfolded version, each event, u_k, is labelled by an occurrence index, which represents the unfolding iteration of the original graph. In this way, when the ER System includes a rule $u \rightarrow v$, with delays $[d, D]$, each event v_k may be assigned to an occurrence time $\tau(v_k)$, according to Equations 1 and 2.

$$\tau(v_k) \geq max\{\tau(u_{k-\varepsilon}) + d \mid u_{k-\varepsilon} \rightarrow u_k \in \Re\} \quad (1)$$

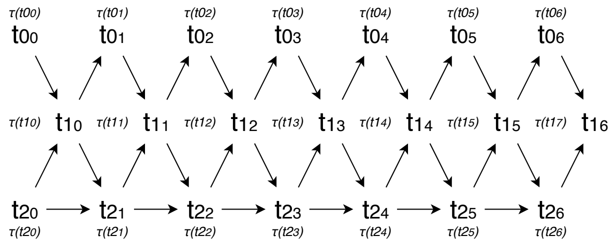
$$\tau(v_k) \leq max\{\tau(u_{k-\varepsilon}) + D \mid u_{k-\varepsilon} \rightarrow u_k \in \Re\} \quad (2)$$

where τ specifies the time occurrence of an event, ε the occurrence index, d and D the minimum and maximum delay values.

An unfolded graph may be used to compute the occurrence times of all acyclic graph events. Figures 2a, 2b illustrate an ER System and its unfolded version, for six unfoldings.



(a) Event Rule System



(b) Unfolding

Fig. 2: Unfolded Event-Rule system Example

C. Timing Separation of Events

TSE fundamentals and approaches for performing the TSE computation, are presented in detail in this section.

Definition 1 (Timing Separation of Events Computation): Given an ER System with bounded interval arc delays $[d, D]$, and any two events u and v, with u the source and v the target events, the maximum (or minimum) separation of events u, v is defined as the maximum (or minimum) occurrence time difference.
In order to identify the maximum (or minimum) separation between a pair of events, the occurrence times of the pair of events in question, must have reached a repetitive, periodic behaviour. This ensures that the system will have reached a steady-state behavior, determined by one or more critical event cycles.

Hence, maximum TSE may be computed as follows;

$$TSE_{max} = T(target) - t(source) \quad (3)$$

where T computes the maximum occurrence, and t the minimum event occurrence respectively.

Definition 2 (Critical Cycle): [4] A Critical Cycle of an ER System is the set of nodes and edges which form the simple cycle(s), which dictate the system's periodic behavior. A Critical Cycle possesses the maximum ratio of arc delays, to total number of tokens for the nodes and edges it contains. In general, a system may have more than one Critical Cycle, when multiple such Cycles have identical ratio.

$$\bar{C_T} = \max_{\forall \text{cycles}} \frac{\sum d(c)}{\sum T(c)} \quad (4)$$

where $\sum d(c)$ corresponds to the sum of delays across timing arcs, and $\sum T(c)$, the sum of tokens per cycle respectively.

Definition 3 (Steady-State): [4] Given a fixed-delay ER System, its steady-state is reached when its period is determined by critical cycle(s) [4], and event occurrences repeat a pattern, *i.e.* each event occurs periodically, based on a fixed time interval.

In general, an ER system with an arbitrary number of tokens may not reach a single value period, but its periodic state may be an oscillation between two or more sequences of period values, *e.g.* $\{2, 3, 4\}$. In such cases, unfolding-based methods are prohibitive, as it is very difficult to determine when the unfolding process should stop. For the purposes of maximum, or minimum, TSE computation however, determining the maximum or minimum periodic behaviour is sufficient.

Figure 3 will be used as a simple example, which TSE is determined, based on unfolding method.

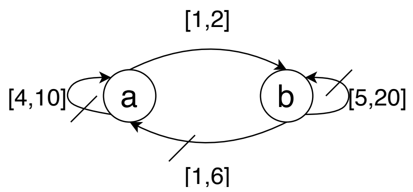


Fig. 3: Figure 1 Example, from [2]

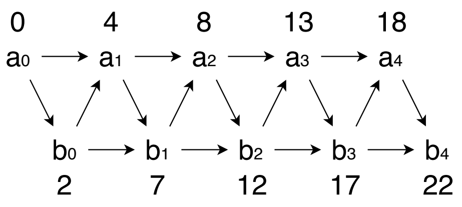


Fig. 4: Unfolding of Figure 1 Example, from [2], with timing distance from root annotated, based on d values

In Figure 4, steady-state behavior is reached at the third unfolding, whereas period stabilises to 5 timing units, and is determined by the $b \rightarrow b$ critical cycle. We will now illustrate how to compute the maximum TSE between two occurrences of event a.

As four unfoldings have been performed, event a_3 is selected as source, and event a_4 as target events. Both events are within the system's stable time window, *i.e.* within the third unfolding. Earlier events would not be appropriate. The timing distance of a_3 is minimum in Figure 4, as it is based on d values. Then, we must perform longest path computation, from event b_2, which occurs one period before the source event, a_3, to the target, *i.e.* a_4. The computed longest path from b_2 to a_4, through b_3 takes 26 timing units, using now D value. As b_2 occurs 1 unit before the source, a_3, TSE_{max} is $26 - 1 = 25$ time units.

D. Primal-Dual Method for Critical Cycle Computation

As mentioned in the previous section, one fundamental issue with unfolding methods is identifying the required number of unfoldings required for the system to reach steady-state. A direct method to compute an ER System's Critical Cycle, without unfolding it, is Burn's Primal-Dual Method, which is also of Polynomial complexity [1]. The latter algorithm supports a single timing delay per ER System arc, and performs iterative relaxation of the ER system period, until a Critical Cycle is identified. As this method does not explicitly enumerate all cycles, it avoids Exponential complexity.

The first step of the algorithm is to remove all arcs with occurrence index $\varepsilon > 0$. Then, it topologically sorts the resultant acyclic graph. Based on the topological order, x node values are set, based on the following equations. The x node values represent the occurrence time offset of an event, within the system period. The value a is the delay of an (u, v) arc connecting nodes u and v.

$$x_v(0) = \begin{cases} 0 & \text{if } v \text{ is root} \\ max\{x_u(0) + a\} & \text{as } (u, i-\varepsilon) \rightarrow (v, i) \wedge \varepsilon \leq 0 \end{cases} \tag{5}$$

A critical arc from node u to node v, indicates that x_v has assumed its value from this specific arc. Initial period is computed as:

$$p(0) = max\left\{\frac{x_v(0) - x_u(0) - a}{-\varepsilon}\right\} \tag{6}$$

where $(u, i-\varepsilon) \rightarrow (v, i) \in \Re \ \wedge \ \varepsilon > 0$.

For all arcs, $(u, i-\varepsilon) \rightarrow (v, i)$, if $x_\nu(k) = x_v(k) + a - \varepsilon p(k)$, then the arc is a critical arc. If the critical arc graph is cyclic, or period at iteration k becomes zero, *i.e.* $p(k) = 0$, then the algorithm exits. Else, it topologically sorts the critical arc graph, and sets $\dot{p}(k) = 1$, where $\dot{x}$ represents the maximum distance of an event from the root, in period units, and is computed in topological order, using:

$$\dot{x}_v(k) = \begin{cases} 0 & \text{if } v \text{ is root} \\ min\{\dot{x}_u(k) - \varepsilon\dot{p}(k)\} & (u, i-\varepsilon) \rightarrow (v, i) \ \textit{is critical} \end{cases} \tag{7}$$

As $\dot{x}$ values are by their definition negative, min is used instead of max. Next, the value θ is computed, which represents the minimum delay per period allowable reduction, or intuitively, the reduction that will make the smallest number of edges to become critical.

$$\theta(k) = min\{\frac{x_\nu(k) - x_v(k) + \varepsilon p(k) - a}{\dot{x}_\nu(k) - \dot{x}_v(k) + \varepsilon\dot{p}(k)} \\ |\dot{x}_\nu(k) - \dot{x}_v(k) + \varepsilon\dot{p}(k) > 0\} \tag{8}$$

The θ value is used to update x values and period value p.

$$x(k+1) = x(k) - \theta(k)\dot{x}(k) \tag{9}$$

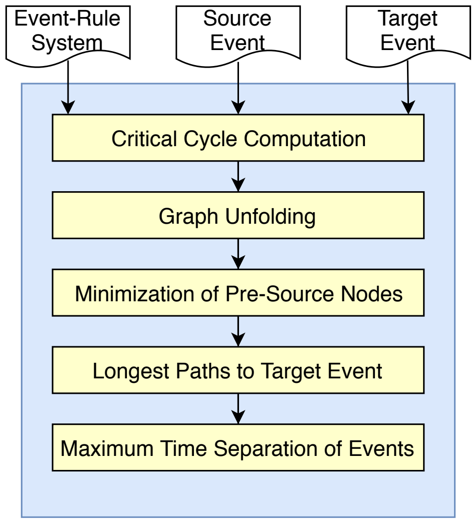
$$p(k+1) = p(k) - \theta(k)\dot{p}(k) \tag{10}$$

The algorithm iteratively updates x values and period p, until the critical edges graph becomes cyclic, at which point p has converged. Algorithm complexity is $O(E \times (V + E))$, as a topological sorting may be required per edge.

III. Novel Maximum TSE Algorithm Flow

In general, maximum TSE requires two steps. First, to minimize the occurrence of the source event, based on specified d values. This implies also minimizing the delay of any events before the source, *i.e.* pre-source events. This is the case, as the longest path to the target event, based on the D values set, may start not from the source, but from a pre-source event.

In our TSE flow, we avoid unfolding the ER system for computing its Critical Cycle, by using the Primal-Dual Algorithm [1] instead. The latter is run on the ER system labelled with d values. Then, the computed x values are used, representing time offsets are used to label the ER system, based on the Critical Cycle delays. In this step, x values must be transformed to be relative to the source event, so as to also minimise the timing of pre-source events. At this point, the ER system is unfolded, for TSE to be computed, but only twice, and the longest path to the target event is computed, using D values.

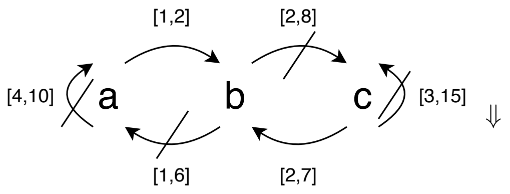


Event-Rule System
Source Event
Target Event
Critical Cycle Computation
Graph Unfolding
Minimization of Pre-Source Nodes
Longest Paths to Target Event
Maximum Time Separation of Events

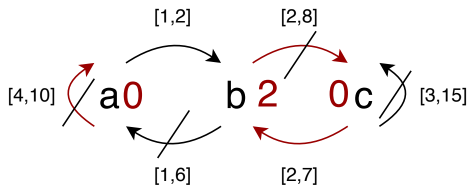
Fig. 5: Maximum Time Separation of Events Algorithm Flow

A. d-values Time Occurrences Annotation

Figure 6(b) illustrates an example of the Critical Cycle Computation Algorithm [1] result. Edges coloured in red illustrate the computed Critical Cycle. Values in red are the reported x values. In this example, two identical critical cycles exist, $a \to a$ and $c \to b \to c$. Root noted are a, c and b occurs x_b, *i.e.* 2 time units after root nodes.



(a) ER System



(b) Primal-Dual Method Algorithm Result

Fig. 6: Primal-Dual Method Algorithm Result Example

B. *Source to Target Unfolded Graph Construction*

To compute the TSE, the relative delay value to the source has already been computed, based on the specified d values. We now need to also compute the maximum delay to the target, using the D values. To do this, we create a minimum size unfolded version of the ER system graph, based on the original cyclic one. For the unfolded graph, we must decide both the number of necessary unfoldings, as well as pick the reference source and target events appropriately.

For taking into account all pre-source nodes in the path to target computation, we initially unfold the graph twice, with the source node event reference being in the second unfolding. This gives us one extra unfolding for taking into account all pre-source nodes, which may be relatively delayed with respect to it. With these two unfoldings, and the source reference in the second one, all nodes with time offsets within $x_{source} - period$ will exist in the unfolded graph, thus the graph is complete with respect to taking into account all pre-source events. An interesting observation is that the only case where the second unfolding will not be needed, *i.e.* all relative pre-source events are guaranteed to exist within the first unfolding, is when Critical Cycle Computation annotates the source with the maximum timing offset, *i.e.* x_{source} possesses the largest x value. For the target event, if $(x_{target} > x_{source})$, the target node reference will already be contained in the unfolded graph, as it occurs after the source, within the system's period. However, in the opposite case, where $(x_{source} > x_{target})$, a path connecting source and target will not exist in the already unfolded graph, as the target occurs before the source of the same unfolding iteration. Thus, an extra third unfolding is required.

Figures 8(a), (b) illustrate the Source to Target Graph Construction for computing the maximum TSE between events

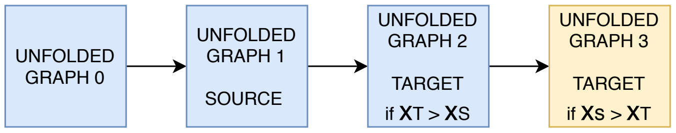
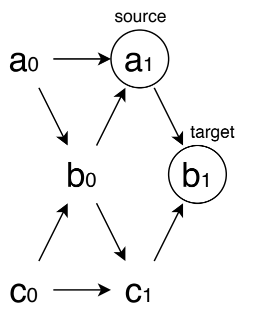
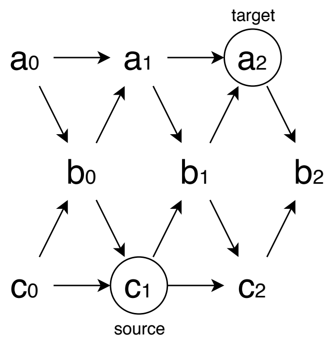


Fig. 7: Graph Unfolding for Max TSE Algorithm
Three default unfoldings. If $x_S \geq x_T$, then one extra unfold

a and b, and c and a respectively, for the Event RS of Figure 6. For the a, b TSE computation, two unfoldings are sufficient, whereas for the c, a TSE computation, an additional third unfolding is necessary.



(a) $x(source) < x(target)$



(b) $x(source) \geq x(target)$

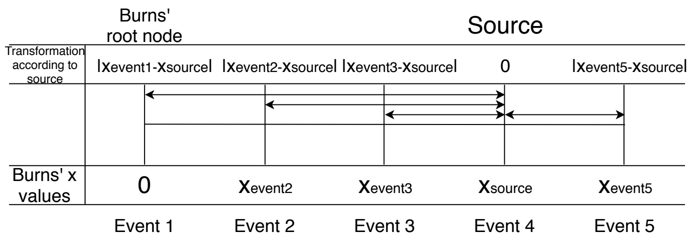
Fig. 8: Graph Unfolding Cases of Fig. 6

C. *x values to Source node Timing Offsets*

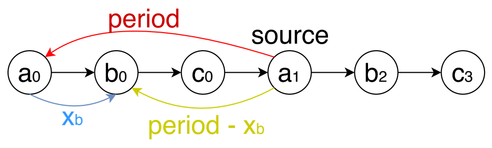
Algorithm [1] computes timing offsets, *i.e.* x node values with respect to the graph root nodes. Root nodes correspond to marked nodes, which are allowed to immediately occur, and will possess zero x-values. However, to compute TSE, we are interested in the timing offset to the source, not the root event. Thus, we need to transform the x values to source node timing offsets. This may be performed by subtracting x from x_source, for all nodes, and then setting x_source to 0, to render everything relative to the source event. The transformation is illustrated in Figure 9a, where nodes with x value offsets both before and after the source are shown. If a node has x offset after x_{source}, *i.e.* $x_{node} > x_{source}$, then after the transformation, it will occur $x_{node} - period$ before the source, but in the previous period, *i.e.* unfolding. This is illustrated in Figure 9b.

D. *D Path to Target Computation*

To compute the Maximum TSE, we now must compute the maximum delay to the target, from the timing occurrences labelled on the created unfolded acyclic graph. We do this by using a Longest Path algorithm with the source and all pre-source nodes as start points, with their timing offsets as labelled. Figure 10 shows an example, where the maximum delay path to $a2$ is from node b_0.



(a) x values and source node timing offsets differences



(b) $x_{node} > x_{source}$ transformation

Fig. 9: x values to source node timing offsets cases

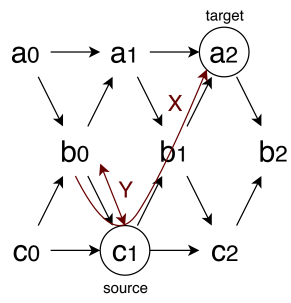


Fig. 10: Maximum Delay Path for Figure 8(b)

Algorithm 1: *compute_maximum_TSE*

```
Input: ER System $G$ = ($V$, $E$), Source, Target Events;
Output: Maximum Time Separation of Events;
// Compute Period, Critical Cycle and Critical Cycle Delay
  Offsets //
primal_dual_method_algorithm($G$); [1]
if ($x_{target} > x_{source}$) then
    // two unfoldings necessary //
    // target node is in second unfolding //
    graph_unfold(G, U, 2);
else
    // three unfoldings needed, //
    // target node in third unfolding //
    graph_unfold(G, U, 3);
forall (nodes before source within 1 period) do
    if $x_{source} > x_{node}$ then
        // node occurs before source event //
        $mindifference := x_{source} - x_{node}$;
    else
        // node occurs after source node //
        // its previous occurrence is 1 period back //
        $mindifference := period - |x_{source} - x_{node}|$;
    longestpathvalue := longest_path(U, node, target);
    $maxtsetemp := longestpathvalue - mindifference$;
    if $maxtsetemp > maxtse$ then
        $maxtse := maxtsetemp$;
end
```

IV. Minimum TSE Computation

So far, we have focused on Maximum TSE Computation. However, we may want to compute Minimum TSE instead. To do this, the same flow may be used, with some minor differences. The Minimum Time Separation of Events requires maximizing the source event time occurrence, and to minimize the target event occurrence. To achieve this, we run the Primal-Dual Method Algorithm on our ER system, but using D values, *i.e.* the maximum values of the time interval, instead of d values. Then, to be able to minimise the target event occurrence, we will use the same unfoldings strategy, as illustrated in Section III-B, but we must perform shortest path, rather than longest path computation, to the target, from all pre-source nodes. Thus, Minimum TSE may be straightforwardly implemented using the same algorithm.

V. Experimental Results

We verified our algorithm, by running a number of ER systems through it. We used published graphs, the TSE results of which were available, and compared against our own algorithm. We also implemented a k-unfoldings version of Acyclic TSE, which reports the TSE on the k times unfolded graph. We did this to validate that we get the exact same results using our approach, and validated correctness for several examples.

Table I illustrates a set of Maximum TSE computations for ER systems presented in the paper. It illustrates the source and target events, the corresponding pre-source node which produces the Maximum TSE, its time offset, the longest path delay to the target node, and finally the Maximum TSE value.

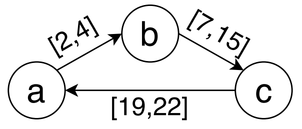


Fig. 11: Single Cycle ER System

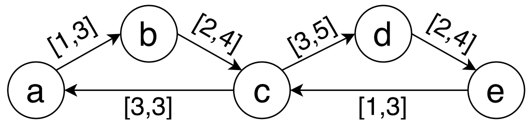


Fig. 12: Multiple Critical Cycles

VI. Conclusions and Future Work

We have a presented a Polynomial complexity Time Separation of Events algorithm based on Critical Cycles. As far as we know, only one previous work [4] computes Time Separation of Events based in cycles on polynomial time, but our complexity is even less than that, as Burns Primal-Dual Method Algorithm has less complexity than the algorithm used in that work. The algorithm presented by Peggy B. McGee and Steven M. Nowick is $O(n^3)$, as Floyd-Warshall algorithm has that complexity, though Burns Primal-Dual Method Algorithm has complexity of $O(E \times (V + E))$. To sum up, in this work we combined Burns Primal-Dual Method Algorithm, Graph

TABLE I: Experimental Results - Maximum Time Separation of Events of events on Critical Cycle

ER System	Source	Target	TSE Path Start Node	TSE Start Node Time Offset	TSE Path Delay to Target	Maximum TSE Result
Figure 6	a	a	b	2	21	**19**
Figure 6	b	c	c	2	15	**13**
Figure 6	c	b	b	2	15	**13**
Figure 11	a	a	b	26	78	**52**
Figure 11	a	b	b	26	41	**15**
Figure 11	a	c	b	26	56	**30**
Figure 11	b	a	c	21	63	**42**
Figure 11	b	b	c	21	67	**46**
Figure 11	b	b	c	21	41	**20**
Figure 11	c	a	a	9	41	**32**
Figure 11	c	b	a	9	45	**36**
Figure 11	c	c	a	9	60	**51**
Figure 12	a	a	b	5	19	**14**
Figure 12	a	b	b	5	10	**5**
Figure 12	a	c	b	5	16	**11**
Figure 12	a	d	b	5	21	**16**
Figure 12	a	e	b	5	13	**8**
Figure 12	b	a	e	5	18	**13**
Figure 12	b	b	e	5	15	**10**
Figure 12	b	c	e	5	15	**10**
Figure 12	b	d	e	5	20	**15**
Figure 12	c	a	a	3	10	**7**
Figure 12	c	b	a	3	13	**10**
Figure 12	c	c	a	3	19	**16**
Figure 12	c	d	a	3	12	**9**
Figure 12	c	e	a	3	16	**13**
Figure 12	d	a	b	5	19	**14**
Figure 12	d	b	b	5	10	**5**
Figure 12	d	c	b	5	16	**11**
Figure 12	d	d	b	5	21	**16**
Figure 12	d	e	b	5	13	**8**
Figure 12	e	a	c	5	15	**10**
Figure 12	e	b	c	5	18	**13**
Figure 12	e	c	c	5	12	**7**
Figure 12	e	d	c	5	17	**12**
Figure 12	e	e	c	5	21	**16**

Unfolding Method, and Path Finding Algorithms to compute Critical Cycles, Appearance of Source Event and Pre-source Nodes, Paths to Target Event and compute the Longest, and eventually Time Separation of Events on critical cycles. We can not perform our algorithm outside of critical cycles for the reason provided in Burns Primal-Dual Method Algorithm Section.

References

[1] S. M. Burns, "Performance analysis and optimization of asynchronous circuits," California Institute of Technology, Computer Science Department, Tech. Rep., 1991.

[2] H. Hulgaard, S. M. Burns, T. Amon, and G. Borriello, "An algorithm for exact bounds on the time separation of events in concurrent systems," *IEEE Transactions on Computers*, vol. 44, no. 11, pp. 1306–1317, 1995.

[3] S. Chakraborty, D. L. Dill, and K. Y. Yun, "Efficient algorithms for approximate time separation of events," *Sadhana*, vol. 27, no. 2, pp. 129–162, 2002.

[4] P. B. McGee and S. M. Nowick, "An efficient algorithm for time separation of events in concurrent systems," in *2007 IEEE/ACM International Conference on Computer-Aided Design*. IEEE, 2007, pp. 180–187.

[5] C. A. Petri and W. Reisig, "Petri net," *Scholarpedia*, vol. 3, no. 4, p. 6477, 2008.

[6] I. Poliakov, V. Khomenko, and A. Yakovlev, "Workcraft–a framework for interpreted graph models," in *International Conference on Applications and Theory of Petri Nets*. Springer, 2009, pp. 333–342.

[7] N. Xiromeritis, S. Simoglou, C. Sotiriou, and N. Sketopoulos, "Graph-based sta for asynchronous controllers," in *IEEE Power and Timing Modeling, Optimization and Simulation*. IEEE, 2019.