In Proceedings of the International Symposium on Circuits and Systems, pages V-367-V-370, Sydney, Australia, May 2001.

# THE HIERARCHICAL TIMING PAIR MODEL

*Nitin Chandrachoodan, Shuvra S. Bhattacharyya*[†]*and K. J. Ray Liu*

Department of Electrical and Computer Engineering,
University of Maryland, College Park, MD 20742
(nitin,ssb,kjrliu@eng.umd.edu)

## ABSTRACT

We present a new model for representing timing information for functions in High-Level Synthesis (HLS). We identify shortcomings of the conventional timing model, which is a very simple model derived from the combinational logic model, and show that our new model overcomes many of these defects. In particular, we are able to provide a unified timing model that describes hierarchical combinational and iterative circuits and provides a compact representation of the information, that can be used to streamline system performance analysis.

We present experimental results that demonstrate the effectiveness of our new approach, and describe an efficient algorithm to easily compute the required timing parameters from a description of the graph.

## 1. INTRODUCTION

High-Level Synthesis (HLS) refers to the task of constructing an architecture, binding and schedule for an algorithm that has been described at a high level of abstraction. The algorithm is usually represented as a dataflow graph whose vertices represent functions and edges represent communication or dependencies. To map such a dataflow graph onto an architecture (either hardware or software) efficiently, we need to annotate the application specification and architecture with information about the execution times of vertices, and the area utilization and power consumption of processing resources. The timing information is used to generate a set of constraints related to the system that the actual implementation must satisfy.

The conventional model for describing timing in this context is derived from the method used in combinational logic analysis. Here each vertex is assigned a single value (called the "propagation delay") representing the maximum delay among all its input-output pairs.

An important requirement of a timing description is the ability to represent systems hierarchically. For example, Fig. 1 shows the circuit of a full adder. If we were to consider this as part of a larger system (say a 4-bit adder made of 4 full adders), we would prefer to use the timing information for the hierarchical block view, rather than the expanded gate-level view. The reason for this is that algorithms for path length computations are typically $O(|V||E|)$ where $|V|$ is the number of vertices and $|E|$ is the number of edges in the graph. The hierarchical view uses 1 vertex and 5 edges to

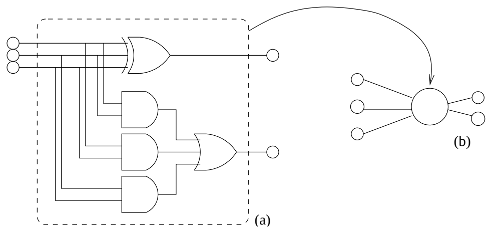

Figure 1: (a) Full adder circuit. (b) Hierarchical block view.

represent the same information as the expanded graph that has 5 vertices and 14 edges. In large systems, the savings offered by using hierarchical representations are essential to retaining tractability. A hierarchical representation would also be very useful in commonly used sequential circuits such as digital filter implementations.

A major disadvantage of the conventional approach is that it does not allow a hierarchical description of the system timing when the system contains delay elements (iterative systems) such as the digital filter mentioned above. These delay elements roughly correspond to registers in a hardware implementation, but are more flexible in that they do not impose the restriction that all the delay elements are activated at the same instant of time [1, 2, 3]. This allowance for variable phase clocking is an important way in which HLS differs from combinational logic implementation. The *rephasing* optimization in [1] provides a good example of how this can be used. Even in sequential logic synthesis, variable phase clocking has been considered in such forms as clock skew optimization [4] and shimming delays [5].

To the best of our knowledge, there does not appear to be any other timing model that addresses this issue. Using conventional models, a complicated subsystem containing sequential elements will need to be represented in full in the context of the overall system design, rather than using a more convenient condensed description of the timing parameters alone.

In this paper, we propose a different timing model that overcomes these difficulties. By introducing a slightly more complex data structure that allows for multiple input-output paths with differing numbers of delay elements, we are able to provide a single timing model that can describe both purely combinational and iterative systems. For purely combinational systems, the model reduces with minimal overhead to the existing combinational logic timing model. Further details are also available in [6].

Our model provides compact representations of the timing data for large systems. We have used the ISCAS 89/93 benchmark circuits to test our ideas and have obtained promising results.

[*]This research was supported in part by the US National Science Foundation Grant #9734275 and NSF NYI Award MIP9457397

[†]Also with the University of Maryland Institute for Advanced Computer Studies.

In the next section, we discuss the requirements that a timing model must meet, and examine some of the shortcomings of the conventional model. Section 3 then presents a new model that overcomes these defects, and explains how it can be efficiently stored and manipulated. Section 4 presents the results obtained by applying the new technique to benchmark circuits. Finally, we present our conclusions and some interesting directions for future research.

## 2. REQUIREMENTS OF A TIMING MODEL

In order to understand the requirements of timing descriptions for hierarchical systems, it is first useful to clarify certain assumptions that are made in describing simple combinational systems.

First, the combinational delay of a system is the *maximum* delay between any input/output pair in the system. So after the inputs are stable, we can wait for the amount of time specified by this delay, and be sure that the output is stable. In some cases, especially in HLS, the time is in terms of integer multiples of a system clock.

For multiple-input-multiple-output (MIMO) systems, we assume that the inputs are synchronized so that the overall system can be treated as single-input-single-output (SISO). This is a common assumption in combinational timing models [5, 7]. To see why, consider for example the full adder circuit from fig. 1. Since the output depends on all the inputs, it is acceptable to assume that the computation start only after all inputs are available, thus synchronizing the inputs. It is clear that this assumption breaks down when different outputs do not depend on all inputs, but in most cases, this is considered an acceptable tradeoff as it reduces the complexity of the analysis.

For dataflow graphs used in HLS, we use essentially the same combinational timing model that is described above. Delay elements, however, are treated differently [1, 2, 3]. In sequential logic circuits, all delays are treated as *flip-flops* that are triggered on a common clock edge. In HLS scheduling, we assume no such restriction on the timing of delays. We assume that each functional unit can be started at any time (possibly by providing a start signal).

Now we can see what exactly are the uses of a timing model. The timing information associated with a block is used primarily for the purpose of establishing constraints on the earliest time that the successor elements of the block can start operating (*i.e.* when its outputs become stable once the inputs are applied). By using these constraints, additional metrics can be obtained relating to the throughput and latency of the system, such as the *iteration period bound*, which is the same as the *maximum cycle mean* [8] for single rate graphs. The constraints are used for determining the feasibility of different schedules of the system, where a schedule consists of an ordering of the vertices on resources that can provide the required functionality.

## 3. THE HIERARCHICAL TIMING PAIR MODEL

Having identified the requirements of a timing model and the shortcomings of the existing model, we can now use Fig. 2 to illustrate the ideas behind the new model for timing. In this figure, we use $t_\_$ to refer to the propagation delay of a block, and $x_\_$ to refer to the *start time* of the block. $T$ is the iteration interval (clock period for the delay elements).

To provide timing information for a complex block, we should be able to emulate the timing characteristics that this block would

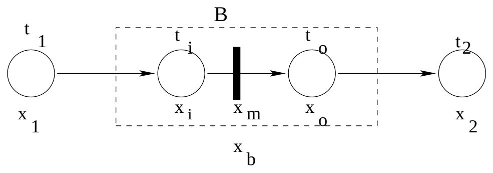

Figure 2: Timing of complex blocks

imply between its input and output. To clarify this idea, consider the block in Fig. 2. If we were to write the constraints in terms of the internal blocks $x_i$ and $x_o$, we would obtain

$$x_i - x_1 \geq t_1 ; x_o - x_i \geq t_i - 1 \times T ; x_2 - x_o \geq t_o$$

Now we would like to compute certain information such that if we were to combine the complex block $B$ under the single start time $x_b$, we would still be able to write down equations that would provide the same constraints to the environment outside the block $B$. We see that this is achieved by the following constraints:

$$x_b - x_1 \geq t_1 ; x_2 - x_b \geq t_i + t_o - 1 \times T$$

In other words, if we assume that the execution time of the block $B$ is given by the expression $t_i + t_o - 1 \times T$, we can put down constraints that exactly simulate the effect of the complex block $B$.

In general, consider a path from input $v_i = v_1$ to output $v_o = v_k$ through vertices $\{v_1, \ldots, v_k\}$ given by $p : v_1 \to v_2 \to \cdots \to v_k$, with edges $e_i : v_i \to v_{i+1}$. Let $t_i$ be the execution time (propagation delay assuming it is a simple combinational block) of $v_i$, and let $d_j$ be the number of delays on edge $e_j$. Now we can define the *constraint time* of this path as

$$t_c(p) = \sum_{i=1}^{k} t_i - T \times \sum_{j=1}^{k-1} d_j$$

We use the term "constraint time" to refer to this quantity because it is in some sense very similar to the notion of the execution time of the entire path, but at the same time is relevant only within the context of the constraint system it is used to build. Also, we use the term $c_p$ to refer to the sum $\sum_{i=1}^{k} t_i$, and $m_p$ to refer to the sum $\sum_{j=1}^{k-1} d_j$. The ordered pair $(m_p, c_p)$ is referred to as a *timing pair*.

We therefore see that by obtaining the pair $(m_p, c_p)$ (in the example of Fig. 2, $c_p = t_i + t_o$ and $m_p = 1$), we can derive the constraints for the system without needing to know the internal construction of $B$.

We can understand the constraint time as follows: if we have a SISO system with an input data stream $x(n)$ and an output data stream $y(n) = 0.5 \times x(n-1)$, the constraint time through the system is the time difference between the arrival of $x(0)$ on the input edge and the appearance of $y(0)$ on the corresponding output edge. This is very similar to the definition of *pairwise latencies* in [1]. It is obvious that $y(0)$ can appear on its edge before $x(0)$, since $y(0)$ depends only on $x(-1)$ which (if we assume that the periodicity of the data extends backwards as well as forwards) would have appeared exactly $T$ before $x(0)$. So the constraint time through this system is $t_m - T$, where $t_m$ is the propagation delay of the unit doing the multiplication by $0.5$ and $T$ is the iteration period of

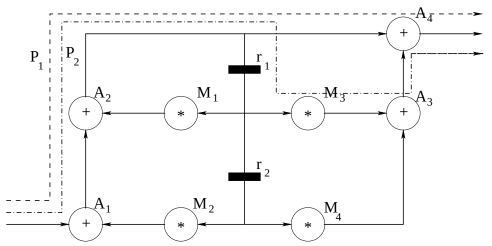

Figure 3: Second order filter section [3].

the data on the system. This number can be negative, and in fact, depends on the value chosen for $T$.

This description of timing pairs makes it clear that the actual constraint time of a path through the graph depends on the iteration interval $T$. In particular, for different values of $T$, it is possible that a different path through the circuit results in the largest constraint time. In other words, the longest path through the graph now depends on $T$. As a result, we need to efficiently compute and store enough information about all input-output paths so that we can easily find the actual value of the largest constraint time between the input and output.

An example of this is seen in Fig. 3, which shows a second-order filter section [3]. Here $P_1$ and $P_2$ are distinct I-O paths. Let the execution time for all multipliers be 2 time units and for adders be 1 time unit, except for $A_3$ which is 2 time units. In this case, for an iteration period ($T$) between 3 and 4, $P_2$ is the dominant path, while for $T > 4$, $P_1$ is the dominant path. So we now need to store both these $(m_p, c_p)$ values.

We therefore end up with a *list* of timing pairs that model the timing of the circuit. The actual constraint time of the overall system can then be readily computed by traversing this list to find the maximum path constraint time. The size of the list is bounded above by the number of delays in the system ($|D|$).

We now have a model where the *timing pairs* that we defined above can be used to compute a *constraint time* on a system, which can be used in place of the execution time of the system in any calculations. This model is now capable of handling both combinational and iterative systems, and can capture the hierarchical nature of these systems easily. We therefore refer to it as the *Hierarchical Timing Pair (HTP) Model.*

This definition of constraint time also results in a simple method for determining the iteration period or maximum cycle mean of the graph. Lawler's method [9] combined with the adaptive negative cycle detection techniques from [8] provides an efficient method of computing the maximum cycle mean of the system, since it operates by fixing $T$ and testing the system for consistency, using a binary search to iteratively improve the estimated value of $T$. Because the constraint time of each path depends on the iteration period which is as yet unknown, it is not obvious how other algorithms for the MCM can be extended to this model.

### 3.1. Data structure and Algorithms

We now present an efficient algorithm to compute the list of timing pairs associated with a system.

Consider a system where there are two distinct I-O paths $P_1$ and $P_2$, with corresponding timing pairs $(c_{p_1}, m_{p_1})$ and $(c_{p_2}, m_{p_2})$. Table 1 shows how the two paths can be treated based on their timing pair values. We have assumed without loss of generality that

| | Condition | Dominant path |
|---|---|---|
| 1. | $m_{p_1} = m_{p_2}, c_{p_1} < c_{p_2}$ | $P_2$ |
| 2. | $m_{p_1} > m_{p_2}, c_{p_1} > c_{p_2}$ | $T_0 \leq T < \frac{c_{p_1} - c_{p_2}}{m_{p_1} - m_{p_2}}$: $P_1$ |
| | | $T \geq \frac{c_{p_1} - c_{p_2}}{m_{p_1} - m_{p_2}}$: $P_2$ |
| 3. | $m_{p_1} > m_{p_2}, c_{p_1} < c_{p_2}$ | $P_2$ |

Table 1: Tests for dominance of a path.

**Algorithm 1** `relax_edge`

**Input:** edge $e : u \rightarrow v$ in graph $G$; $t(u)$ is the execution time of source vertex $u$, $d(e)$ is the number of delays on edge $e$; $list(u), list(v)$ are timing pair lists.

**Output:** Use the conditions from Table 1 to modify $list(v)$ using elements of $list(u)$. Return TRUE if a modification was made, else return FALSE

```
1: RELAXED ←FALSE
2: for all timing pairs $t_a$ from $list(u)$ do
3:    $t_a$←$t_a$ + $(t(u), d(e))$
4:    if $t_a$ dominates an element of $list(v)$ then
5:       insert $t_a$, adjust $list(v)$
6:       RELAXED ←TRUE
7:    end if
8: end for
9: return RELAXED
```

$m_{p_1} \geq m_{p_2}$. The minimum iteration interval allowed on the system is denoted $T_0$. This would normally be the iteration period bound of the circuit, but may be set to a higher positive value for design safety margins.

The conditions from Table 1 can be used to find which timing pairs are necessary for a system and which can be safely ignored. For the example of Fig. 3, $P_1$ has the timing pair $(0, 3)$ while $P_2$ has $(1, 7)$ with timing as assumed in section 3. Thus from condition 2 above, $P_2$ will dominate for $3 \leq T < 4$, and $P_1$ will dominate for $T \geq 4$.

The algorithm we use to compute the timing pairs is based on the Bellman-Ford algorithm for shortest paths in a graph. We have adapted it to compute the longest path information we require, while simultaneously maintaining information about multiple paths through the circuit corresponding to different register counts.

Algorithm 1 implements the *edge relaxation* step of the Bellman-Ford algorithm [10, p.520]. However, since there are now multiple paths (with different delay counts) to keep track of, the algorithm handles this by iterating through the timing pair lists that are being constructed for each vertex. An important point to note here is that the constraint time around a cycle is always negative for feasible values of $T$, so the `relax_edge` algorithm will not send the timing pair computations into an endless loop.

Using algorithm 1, the overall timing pairs are easily computed using the Bellman-Ford algorithm [10, p.532]. The complexity of the overall algorithm is $O(|D||V||E|)$ where $|D|$ is the number of delay elements in the graph (therefore a bound on the length of a timing pair list of a vertex), $|V|$ is the number of vertices, and $|E|$ is the number of edges in the graph. Note that $|D|$ is quite a pessimistic estimate, since it is very rare for all the delays in a circuit to be on any single dominant path from input to output.

| # timing pairs | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| # circuits | 21 | 13 | 5 | 4 | 1 |

Table 2: Number of dominant timing pairs computed for ISCAS benchmark circuits.

## 4. RESULTS

As was mentioned in the introduction, the main benefit of the model we propose is in the ability to hierarchically model systems. Also, the model allows us to represent all relevant input-output paths using a list of timing pairs as described in sec. 3.1. So a suitable measure of the merit of the system would be to see the size (number of elements) of the list required to represent the timing behavior of a graph.

We have run the algorithm described in section 3.1 on the ISCAS 89/93 benchmarks. A total of 44 benchmark graphs were considered. For this set, the average number of vertices is 3649.86, and the average number of output vertices in these circuits is 39.36.

First we consider the case where synchronizing nodes were used to convert the circuit into an SISO system. We are interested in the number of elements that the final timing list contains, since this is the amount of information that needs to be stored. Table 2 shows the breakup of the number of list elements. We find that the average number of list elements is 1.89.

Next, instead of assuming complete synchronization, we considered the case where inputs are synchronized, and measured the number of list elements at each output. The number of distinct values obtained for this was an average of 14.73. If we make an additional assumption that if two list elements have the same $m_p$ they are the same, this number drops to 3.68. This assumption makes sense when we consider that several outputs in a circuit pass through essentially the same path structures and delays, but may have one or two additional gates in their path that creates a slight and usually ignore-able difference in the path length. For example, the circuit `s386` has 6 outputs. When we compute the timing pairs, we find that 3 have an element with 1 delay, and the corresponding pairs are $(1, 53)$, $(1, 53)$, $(1, 57)$. Thus instead of 3 pairs, it seems reasonable to combine the outputs into 1 with the timing pair $(1, 57)$ corresponding to the longest path.

In order to compare these results, note that if we did not use this condensed information structure, we would need to include information about each vertex in the graph. In other words, if we accept the (in most cases justifiable) penalty for synchronizing inputs and outputs, we need to store an average of 1.89 terms instead of 3649.86.

We have not considered the case of relaxing the assumptions on the inputs as well. This would obviously increase the amount of data to be stored, but as we have argued, our assumption of synchronized inputs and outputs has a very strong case in its favor.

We have also computed the timing parameters for HLS benchmarks such as the elliptic filter and 16-point FIR filter from [3]. These are naturally SISO systems which makes the synchronizing assumptions unnecessary. If we allow the execution times of adders and multipliers to vary randomly, we find that the FIR filter has a number of different paths which can dominate at different times. The elliptic filter tends to have a single dominant path, but even this information is useful since it can still be used to represent the filter as a single block. In general, systems which have delay elements in the feed-forward section, such as FIR filters and filters with both forward and backward delays, tend to have more timing pairs than systems where the delay elements are restricted to a relatively small amount of feedback.

## 5. CONCLUSIONS AND FUTURE DIRECTIONS

We have presented the Hierarchical Timing Pair model, and associated data structures and algorithms to provide timing information for use in the analysis and scheduling of dataflow graphs. We have shown that the HTP model overcomes many limitations of the conventional timing models, while incurring a negligible increase in complexity.

Using the examples of the ISCAS and HLS benchmarks, we have demonstrated the power of our approach, and have shown that if we can accept the assumption of synchronizing nodes, we can obtain a reduction by several orders of magnitude in the amount of information about the circuit that we need to store in order to use its timing information in the context of a larger system.

It appears that the HTP model can be efficiently extended to also include multi-rate systems. With certain simple assumptions on the regularity of behavior of such systems, they can be analyzed in the same framework as single rate systems. We are currently working on extending the HTP model to such multirate systems.

## 6. REFERENCES

[1] M. Potkonjak and M. Srivastava, “Behavioral optimization using the manipulation of timing constraints,” *IEEE Transactions on Computer Aided Design*, vol. 17, pp. 936–947, Oct 1998.

[2] P. G. Paulin and J. P. Knight, “Force-directed scheduling for the behavioral synthesis of ASIC’s,” *IEEE Transactions on Computer Aided Design*, vol. 8, pp. 661–679, Jun 1989.

[3] S. M. H. de Groot, S. H. Gerez, and O. E. Herrmann, “Range-chart-guided iterative data-flow graph scheduling,” *IEEE Transactions on Circuits and Systems - I*, vol. 39, pp. 351–364, May 1992.

[4] J. P. Fishburn, “Clock skew optimization,” *IEEE Transactions on Computers*, vol. 39, pp. 945–951, Jul 1990.

[5] H. V. Jagadish and T. Kailath, “Obtaining schedules for digital systems,” *IEEE Transactions on Signal Processing*, vol. 39, pp. 2296–2316, Oct 1991.

[6] N. Chandrachoodan, S. S. Bhattacharyya, and K. J. R. Liu, “The hierarchical timing pair model for synchronous dataflow graphs,” Tech. Rep. UMIACS-TR-2000-75, University of Maryland Institute for Advanced Computer Studies, Nov 2000. *http://dspserv.eng.umd.edu/pub/dspcad/papers/*.

[7] N. Kobayashi and S. Malik, “Delay abstraction in combination logic circuits,” *IEEE Transactions on Computer Aided Design*, vol. 16, pp. 1205–1212, Oct 1997.

[8] N. Chandrachoodan, S. S. Bhattacharyya, and K. J. R. Liu, “Negative cycle detection in dynamic graphs,” Tech. Rep. UMIACS-TR-99-59, University of Maryland Institute for Advanced Computer Studies, September 1999. *http://dspserv.eng.umd.edu/pub/dspcad/papers/*.

[9] E. Lawler, *Combinatorial Optimization: Networks and Matroids*. New York: Holt, Rhinehart and Winston, 1976.

[10] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms*. Cambridge, MA: MIT Press, 1990.