# The Hierarchical Timing Pair Model for Synchronous Dataflow Systems

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 consider the problem of representing timing information associated with functions in a dataflow graph. This information is used for behavioral synthesis of appropriate architectures for implementing the graph. Conventional models for timing suffer from shortcomings that make it difficult to easily represent timing information in a hierarchical manner, especially for multirate systems.

We identify some of these shortcomings, and provide an alternate timing model for hardware implementations that does not have these problems. This model is capable of providing a unified view of hierarchical combinational and sequential circuits under the assumptions of high-level scheduling. The resulting compact representation of the timing information can be used to streamline system performance analysis.

We show that with some reasonable assumptions on the way hardware implementations of multirate systems operate, we can derive general hierarchical descriptions of multirate systems in a similar manner to single sample-rate systems. Several analytical results that previously applied only to single sample-rate systems can also easily be extended to multirate systems under the new assumptions.

We have applied our model to several signal processing applications, and obtained favorable results. We present an algorithm to compute the timing parameters, and have used this to compute timing parameters for a number of benchmarks circuits. We present the results obtained on several ISCAS benchmark circuits and also several multirate dataflow graphs corresponding to useful signal processing applications.

[*]Technical Report UMIACS-TR-2000-75, Institute for Advanced Computer Studies, University of Maryland, College Park, MD 20742, November 2000.

[†]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.

# 1 Introduction

Behavioral Synthesis refers to the task of constructing an architecture, binding and schedule for an algorithm that has been described in terms of the behavior of its constituent elements at a high level of abstraction. It is part of the broader field of high-level synthesis (HLS) and is often used to implement digital signal processing (DSP) applications. In behavioral synthesis for DSP, the algorithm is often 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.

One of the most popular dataflow models (especially for DSP applications) is synchronous dataflow (SDF) [12]. This is a pure dataflow model without control flow constructs. Though it is not Turing-complete, it is capable of representing a large class of useful DSP systems. This model includes the concepts of *consumption* and *production* parameters, which allow convenient representation of multiple sample rates. The special case when all sample rates are equal is referred to as the *single sample-rate (SSR)* case. A subgroup of this is the case of *homogeneous* graphs, where production and consumption parameters on all edges are equal. Systems where multiple sample rates exist in different parts are referred to as *multirate* systems. This model has been widely used to study DSP graphs and several techniques have been developed for mapping graphs represented in this model to both hardware and software architectures. Most analytical results that are known for graph performance metrics have been derived for homogeneous graphs. In our analysis, we first treat the homogeneous case since it is relatively simpler, and the results that we derive for multirate graphs will also apply to other SSR graphs that are not homogeneous.

The conventional model for describing timing in dataflow systems is derived from the method used in combinational logic analysis. Here each vertex is assigned a "propagation delay" value that is treated as the execution time of the associated subsystem. That is, once all the inputs are provided to the system, this propagation delay is the minimum amount of time after which we are guaranteed that the outputs of the system have reached their final stable values.

One major disadvantage of this approach is that it does not allow a hierarchical description of the system timing when the system contains delay elements (iterative systems). 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 [15, 14, 6]. This allowance for variable phase clocking is an important way in which HLS differs from combinational logic implementation. The *rephasing* optimization in [15] 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 [7] and shimming delays [10], and has been recognized as a very useful tool, though it is difficult to implement in practice.

In multirate systems, the most common interpretation of *execution time* is as follows: each vertex is assumed to be enabled when sufficient dataflow tokens have enqueued on its inputs. Once it is enabled in this fashion, it can *fire* at any time, consuming a number of tokens from each input edge equal to the consumption parameter on that edge, and producing a number of tokens on each output edge equal to the production parameter on that edge. The execution time of the vertex is the time between the (instantaneous) consumption and production events.

This model has been used in the context of SDF to derive several useful results regarding consistency, liveness and throughput of graphs modeling DSP systems. However the treatment is quite different from that for homogeneous graphs, and many analytical results for homogeneous systems cannot be extended to multirate systems.

To the best of our knowledge, there does not appear to be any other timing model that addresses the issue of hierarchical timing for dataflow-based DSP designs. Conventional models cannot easily be used to represent systems that are either hierarchical or contain multirate elements. Multirate systems are usually handled by some technique such as deriving the expanded homogeneous equivalent graph (which can lead to an exponential increase in the graph size), while hierarchical systems need to be completely flattened, again resulting in possibly large increases in the size of the graph to be analyzed.

In this paper, we propose a different timing model that overcomes these difficulties for dedicated hardware implementations of the dataflow graph. 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 (iterative systems are sequential systems with feedback, so that the execution of the overall system repeats infinitely over time). For purely combinational systems, the model reduces to the existing combinational logic timing model. For multirate systems, the new model allows a treatment very similar to that for normal homogeneous systems, while still allowing most important features of the multirate execution to be represented. The model also allows several analytical results for homogeneous systems to be applied to multirate systems. As an example, we derive an expression for the iteration period bound of a multirate graph.

We have used our hierarchical timing model to compute timing parameters of the ISCAS benchmarks, which are homogeneous systems. We have also used the model to compute timing parameters of a number of multirate graphs used in signal processing applications. The results show that the new model can result in compact representations of fairly large systems that can then be used as hierarchical subsystems of larger graphs. These results show the large savings in complexity that are possible with the new approach.

In the next section, we discuss the requirements that a timing model for dataflow systems must meet, and examine some of the shortcomings of the conventional model. Section 3 then presents a new model that overcomes these defects. Section 4 describes a data structure and efficient algorithms that can be used to compute the timing parameters according to our model. Section 5 then describes the requirements of timing models for multirate systems, and shows how our model can be extended to these systems. Section 6 presents results of applying the model to several examples from signal processing and the ISCAS benchmark circuits. Finally, we present our conclusions and some interesting directions for further work.

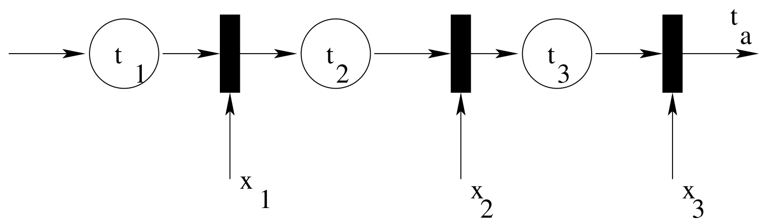
## 2 Requirements of a Timing Model for Hierarchical Systems

In order to understand the requirements that must be satisfied by a timing model for describing hierarchical systems, we first clarify certain assumptions that are often made in describing simple combinational systems. For such systems, the timing description involves specifying a single number that is the combinational propagation delay from the input to the output of the system. In doing so, we make certain assumptions:

1. This value is the *maximum* delay between any input and output pair. This is required for the assumption that by waiting for this much time after applying the inputs, we can safely assume that the outputs have attained stable values.

2. We concentrate on Single-Input Single-Output (SISO) systems. For general Multiple-Input Multiple-Output (MIMO) systems, each input/output pair can have different path lengths resulting in different values for the longest combinational path between them. However, we commonly assume a single value for the delay, which is equivalent to assuming a single dummy input vertex and a dummy output vertex, where all the inputs and outputs synchronize. More accurate models actually do provide "bit-level timing" where they provide further information that specifies the timing on input-output pairs, but these are used rarely. Note that in most cases, when we try to encapsulate timing information for a system, this system will usually have a small number of inputs and outputs with respect to the internal computational complexity. In addition, buses are usually treated as single outputs rather than as 8 or 16 separate outputs. It is worth emphasizing that this assumption is only made for convenience. Our model (as well as most conventional models) can handle MIMO systems by assigning separate timing values to each input-output pair, resulting in some increase in complexity. This results in a trade-off between the amount of information stored and the accuracy of the representation.

In analyzing most dataflow systems, we use essentially the same combinational model that is described above. One difference is that in several cases, instead of a propagation delay in, say, nanoseconds, the timing now refers to a small integer number of clock cycles, which agrees with the software model where each functional unit is assumed to be a primitive block of software that takes a small number of processor cycles. Certain algorithms for scheduling [14, 6] and computing circuit parameters such as the maximum cycle mean [13] actually use this fact to obtain more efficient algorithms.




Single phase clock: x1 = x2 = x3 = 0: ta >= 3 * max (t1, t2, t3)

Multi–phase clock: x1 = t1, x2 = t2, x3 = t3, ta >= (t1 + t2 + t3)

Figure 1: Ripple effects with clock skew (multiple phase clocks)

One major difference between the model used in dataflow scheduling and in circuit level timing regards the treatment of delays on edges. In sequential circuits, the most common policy is to treat all delays as *flip-flops* that are triggered on a common clock edge. In general 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). Because of this, as shown in Fig. 1 a signal applied to a dataflow graph can ripple through the graph much faster if appropriate *phase shifts* are used for triggering the flip-flops on the edges. This is because, in general, the propagation times through different elements can differ quite a bit from one another, but a single-phase clock has to take into account the worst case value. As mentioned before, this assumption is common in HLS, and has also been studied as a potentially useful tool in the context of general sequential synthesis.

In the discussion that follows, we use the term *block* to refer to a SDF system for which we are trying to obtain equivalent timing data. Since we are developing a model to describe hierarchical SDF representation, our block should itself be an SDF model. In particular, we permit the block to be composed of any normal SDF actors. The actors composing the block can in themselves be hierarchical SDF blocks, but for the present we consider them to be simple blocks (where the execution time is a fixed constant). As will be seen later, this does not impose restrictions on the generality of the result. In addition to this, we assume that the block has a single input and a single output, as discussed above.

The block we are considering therefore consists of a single input, single output, and internal simple blocks (composed of combinational units with constant execution times), and delays (registers) where the clock phase is not fixed and can be adjusted for obtaining the best possible performance. The graph representing this block will, in general, be cyclic, but with the restriction that every cycle must have at least one edge with a delay element on it (this is required for feasibility of the system). Now it is obvious that the propagation delay through this system is not a constant. This is because there are multiple paths from the input to the output, each of which may contain a different number of delays. Because of this, the overall delay through the network depends on the iteration period of the overall system of which our block is a part. Thus we can see that the execution time of the unit depends on the data rate on the inputs and outputs, and is not a constant.

We now try to clarify what is implied when we say that two descriptions of a system are equivalent for timing. Note that we are not trying to define the equivalence of circuits in the general case, as this is a considerably more complex problem.

The timing information associated with a block is used primarily for the purpose of establishing constraints on the earliest time that the block can start operating (*i.e.*, when its inputs are ready and stable). That is, since the edges of the dataflow graph denote dependencies that must be satisfied by the vertices, they imply the existence of constraints on the earliest time that a given vertex can obtain all its inputs and start executing its function.

By using these constraints, additional metrics can be obtained relating to the throughput and latency of the system. These constraints are used for determining the feasibility of different schedules of the system, where a schedule essentially consists of an ordering of the vertices on processing resources with required functionality. An important metric of this kind is the *Iteration Period Bound* [16], which is the minimum

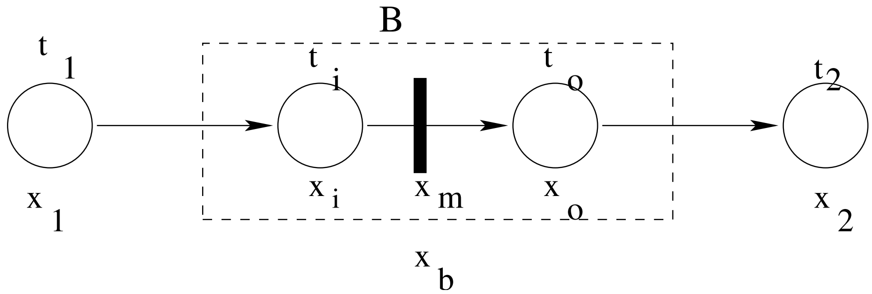



Figure 2: Timing of complex blocks

time within which the graph can complete a full cycle of execution (that is, execute each vertex as many times as necessary to return to a state it started from). For homogeneous SDF graphs, this bound is known to be equal to the *Maximum Cycle Mean (MCM)* of the graph (the maximum over all cycles of the sum of execution times of the vertices divided by the total number of delays on the cycle). It is easily derived from the constraints imposed by the edges of the graph: by concatenating the constraints around any directed cycle in the graph, we find that the result is a constraint on the minimum value of the iteration bound (this will be seen in more detail in the next section once we define the concept of constraint time).

# 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 $t_i$ to refer to the propagation delay of block $i$, and $x_i$ 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 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

$$x_i - x_1 \geq t_1, \tag{1}$$
$$x_o - x_i \geq t_i - 1 \times T, \tag{2}$$
$$x_2 - x_o \geq t_o. \tag{3}$$

Note that the second constraint equation in the list above has the term $(-1 \times T)$ because of the delay element on the edge. Because of this delay, the actor at the output of the edge actually has a dependency on the sample produced in the previous iteration period rather than the current one. This fact is captured by the constraint as shown.

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:

$$x_b - x_1 \geq t_1, \tag{4}$$
$$x_2 - x_b \geq t_i + t_o - 1 \times T. \tag{5}$$

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

$$t_c(p) = \sum_{i=1}^{k} t_i - T \times \sum_{j=1}^{k-1} d_j. \tag{6}$$

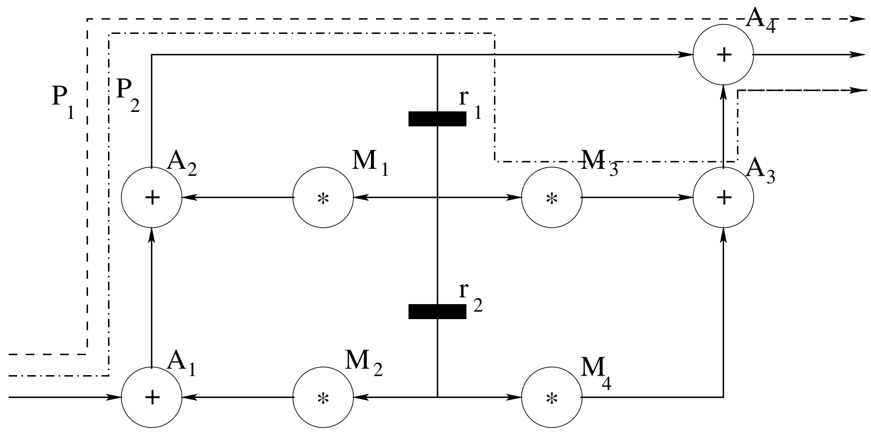



Figure 3: Second order filter section [6].

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*. The terms $m_p$ and $c_p$ were chosen because they are commonly used in mathematical literature to refer to the slope and intercept of a line, which is the role they play here. That is, the constraint time of a path varies as a straight line as the iteration period $T$ associated with the system changes. The slope of the line is given by $m_p$ and $c_p$ is the intercept corresponding to $T = 0$.

We therefore see that by using 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 $B$. The constraint time associated with the complex block $B$ is now given by

$$t_c(B) = c_p - m_p \times T. \tag{7}$$

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 [15]. 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 $T$ time-units 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 the data on the system.

We now need to extend the timing pair model to handle multiple input-output paths, as seen in Fig. 3, which shows a second-order filter section [6]. 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 has an execution time of 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. 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 $(|D|)$.

The constraint time of a path can be negative, and in fact, depends on the value chosen for $T$. This is an important way in which it differs from the simpler conventional concept of an execution time. Since any SDF graph can be looked at as a set of paths from the input to the output, it is possible to compute timing pairs for each of these paths, thereby making it possible to compute the constraint time of the whole SDF system represented by this graph easily. In this way, it is possible to use a hierarchical representation of an SDF graph as a subsystem of a larger graph without having to flatten the hierarchy.

Note that in addition to the timing pairs, we also need to specify a minimum clock period for which

| | 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.

the system is valid. That is, just specifying the timing pairs could result in the erroneous impression that the system can execute at any clock period. In reality, the minimum period for the system depends on the internal minimum iteration bound of the hierarchical subsystem, or it could be set even higher by the designer to take into account safety margins or other constraints that do not derive directly from the dataflow representation.

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. It is obvious that the constraint time around a cycle must be negative to avoid unsatisfiable dependencies. Also, note that for a fixed value of $T$, the constraint time of each subsystem becomes a fixed number rather than a list of timing pairs. Because of this, any algorithm that iterates over different values of $T$ in order to determine the best value that is feasible for the graph will only have to deal with the final constraint time values and not the timing pair lists. Lawler's method [11] provides an efficient way of doing this. It performs a sequence of successive approximations to find a close approximation to the iteration period $T$. Since we have shown [4] the efficiency of Lawler's algorithm on graphs of bounded degree, this algorithm provides an effective way of computing the iteration period for graphs described using the hierarchical model. It may be possible to find other algorithms that can operate directly on the timing pair lists and compute a closed-form analytical expression for the maximum cycle mean of the system. However, since Lawler's method is already known to be efficient in practice, this is not a very urgent requirement.

# 4 Data Structure and Algorithms

We now present an efficient algorithm to compute the list of timing pairs associated with a given dataflow graph. This algorithm returns a list of timing pairs such that no two have the same delay count. In addition, it removes all redundant list elements. This is based on the following observation:

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 $m_{p_1} \geq m_{p_2}$. 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.

Subroutine 1 shows an overview of the algorithm used to check for whether a $(m, c)$ pair is to be added to the list for a vertex. This computation is performed in accordance with the rules of Table 1. By stepping through the different elements of the source list, the routines checks whether there exists at least one path that results in a longer path to the sink vertex. The description in algorithm 1 leaves out endpoints and

**Algorithm 1** Subroutine `try_add_element`.

**Input:** list $t_l$, new element to be added $t_a = (m, c)$, minimum iteration period $T_{min}$
**Output:** if $t_a$ can be added to $t_l$ in the valid range of $T$, does so and returns `TRUE` else returns `FALSE`

start with $k$ at beginning of list $t_l$
**while** $k$ not at end of list $t_l$ **do**
  compare $t_a$ to $k$ and $succ(k)$ using table 1 to see where the corresponding lines intersect
  **if** intersection point such that $t_a$ dominates for some $T$ **then**
    insert $t_a$ after $k$
    return `TRUE`
  **else**
    advance $k$
  **end if**
**end while**
return `FALSE` // reached end of the list unsuccessfully

**Algorithm 2** Subroutine `relax_edge`.

**Input:** edge $e : u \to v$ 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`

RELAXED $\leftarrow$`FALSE`
**for all** timing pairs $t_a$ from $list(u)$ **do**
  RELAXED $\leftarrow$`try_add_elt(`$t_a$`,`$list(v)$`)`
**end for**
return RELAXED

special cases for simplicity. The actual algorithm would need to check for empty lists, insertion at the beginning of the list, etc.

Algorithm 2 implements the *edge relaxation* step of the Bellman-Ford algorithm [5, 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.

Algorithm 3 gives the complete algorithm for computing the timing information. Starting from the source vertex $u_0$ of the system, it proceeds to "relax" outgoing edges and adding the target vertices into a set if necessary. This process is an adaptation of the Bellman-Ford algorithm for shortest paths.

As we have already shown, as long as we restrict attention to $T$ in the valid range (namely $> T_{min}$), we will not encounter positive weight cycles in the graph. Recall that a positive constraint time around a cycle corresponds to an unsatisfiable constraint, which in turn would correspond to a choice of $T$ that is outside the feasible range for the system under consideration.

Using the above algorithm, the timing pairs for a single sample-rate graph are easily computed. 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.

# 5 Multirate Systems

In this section, we consider some problems that arise in the treatment of multirate systems. We examine some examples to see how these difficulties can be overcome, and motivate new assumptions that make it easier to handle these systems mathematically.

The conventional interpretation of SDF execution semantics has been based on token counts on edges. A vertex is enabled when each of its input edges has accumulated a number of tokens greater than or equal

**Algorithm 3** Algorithm `compute_timing`.

**Input:** Directed graph $G$ corresponding to a single-input single-output system

**Output:** Compute the timing lists for each vertex in the graph; the list for the output vertex is the actual timing for the overall system

```
Q ←source vertex u_0
while Q is not empty do
    u ←pop element from Q
    for all edge e : u→v adjacent from u do
        if relax_edge(G) succeeds then
            insert v into Q
        end if
    end for
end while
```

to the consumption parameter on that edge. At any time after it is enabled, the vertex may fire, producing a number of tokens on each output edge equal to the production parameter on that edge. In the following discussion, we use $c$ to refer to the consumption parameter on an edge, and $p$ to refer to the production parameter. The edge in question will be understood from the context.

This interpretation, though very useful in obtaining a strict mathematical analysis of the consistency and throughput of such multirate systems, has some unsatisfactory features when we consider dedicated hardware implementations. One such feature is the fact that it results in tokens being produced in bursts of $p$ at a time on output edges and similarly consumed in bursts of $c$ at a time. This is not the consumption pattern implied in the design of DSP applications, where tokens refer to data samples on edges, and as such will usually be strictly periodic at the sample rate specified for that edge. Moreover, in hardware designs at least, enforcing strict periodicity on the samples means that any buffering required can be built into the consuming unit and no special buffering needs to be provided for each edge.

A more important problem is with regard to the criterion used for firing vertices. Consider the example of the $3:5$ rate changer shown in Fig. 4. According to the SDF interpretation, this vertex can only fire after 5 tokens are queued on its input, and will then instantaneously produce 3 tokens on its output. However, a real rate changer need not actually wait for 5 tokens before producing its first output. In fact, in cases where such rate changers form part of a cycle in the graph, the conventional interpretation can lead to deadlocked graphs due to insufficient initial tokens on some edge, or even due to the distribution of tokens among edges. One real life example where this criterion shows this problem is with the DAT-to-CD data rate converter (used to convert between the data sample rates used in Digital Audio Tape (DAT) and Compact Disc audio (CD)). This is a sample rate conversion with a rate change ratio of $147:160$. The SDF model interprets this by saying that (when a DAT-CD converter is represented as a single block) 147 samples need to queue on the input before even a single output is produced, and that all 160 corresponding outputs are produced together. This is clearly not how it is implemented in practice, where a multiple stage conversion may be used, or even in case of a direct conversion, the latency before producing the first output does not need to be as high as indicated by the conventional SDF model. The Cyclostatic dataflow (CSDF) [2] model provides a way around this by introducing the concept of execution phases. We discuss the relation of our model to the CSDF model in section 5.2.

Note that when the dataflow graph is used to synthesize a software implementation, the token flow interpretation is more useful than in a dedicated hardware implementation. The reason for this is that in software, since a single processor (or more generally, a number of processors that is small compared to the number of actors in the graph) typically handles the execution of all the code, it is possible to construct the code from blocks directly following the SDF graph, using buffers between code blocks to group the data for efficiency. Without buffers, and assuming that the token production and consumption is periodic, the efficiency of the system would be greatly reduced. In hardware, however, each block could be executed by a separate dedicated resource, and since this is happening concurrently, the sample consumption pattern is basically periodic as opposed to the buffered bursts that would be seen in software.

Figure 4 illustrates the above ideas. This is an implementation of a $3:5$ fractional rate conversion that is implemented using an efficient multirate filtering technique (as used in the FIR filter implementation

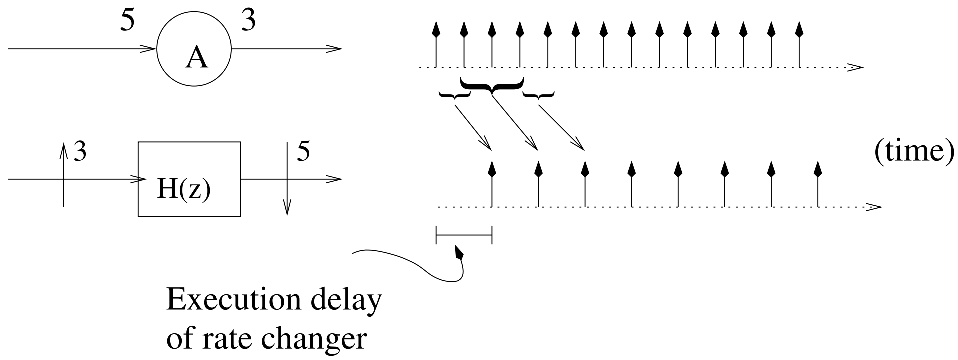



Figure 4: 3 : 5 sample rate changer.

provided with Ptolemy [3]). We have assumed a 7-tap filter ($H(z)$) used for the interpolation, which results in the input-output dependencies as shown in the figure. It is clear from the filter length and interpolation rate that the first output in each iteration (an iteration ends when the system returns to its original state) depends on the first 2 inputs only, the second depends on inputs 2 to 4, and the third depends on inputs 4 and 5. Therefore, the delay pattern shown in the figure is valid as long as there is sufficient time for the filters to act on their corresponding inputs. In other words, it is not necessary to wait for 5 inputs to be consumed before starting to produce the outputs.

It is clear that this implementation is more efficient in terms of number of multiplications. It can also be rearranged for a power savings by noting that each multiplication occurs only once in every 5 inputs.

One point to note here is the following: For a rate conversion as implemented above, the internal structure is such that some input samples are switched to different polyphase components at different time instants. Therefore when we consider internal branches, not all the branches receive the same input stream. Because of this, the algorithms we have described cannot directly be applied to them to compute the timing pairs for these systems. Instead, we need to use timing figures as shown in Fig. 4 to compute the timing. Fortunately, the multirate units for which this is necessary can usually be treated as primitive subsystems of larger circuits, and the examples in section 6 show how we can use the data for a rate converter to compute timing for several larger circuits. Note that even for these systems, it is possible to automate the computation of the timing pair lists. We need to ensure that each input-output pair is considered when computing the maximum constraint time. For simple systems such as the MR FIR filter, inspection shows that the timing list is the same as that of an ordinary FIR filter, possibly combined with some extra constant delay to take into account the offset required for matching up the I-O for all the polyphase components.

The implementation we considered avoids unnecessary computations, so it is possible to save power by either turning off the filters when they are not needed (using the clock inputs), or by using an appropriate buffering and delayed multiplication that will allow the multipliers to operate at $\frac{1}{5}^{th}$ of the rate of the input stream, using the observation that only one of the polyphase components [18] needs to operate for each output sample. This tradeoff would depend on whether we are considering an implementation with dedicated multipliers for each coefficient or shared multipliers. Real hardware implementations of multirate systems must resort to such efficient realizations as the performance penalties can otherwise be large.

The interpretation we use for execution of SDF graphs is therefore as follows: each node receives its inputs in a perfectly periodic stream, and can start computing its outputs some time after the first input becomes available (this time would depend on internal features such as the number of taps in the filter in the above example). The outputs are also generated in a periodic stream at the appropriate rate required for consistency of the system.

An important effect of this alternate interpretation is that it changes the criteria for deadlock in a graph. Under normal SDF semantics, the graph in Fig. 5 would be deadlocked if the edge $AB$ has less than 10 delays on it. On the other hand, 6 delays are sufficient on edge $BC$, while 16 delays are required on edge $CA$ in order to prevent deadlock. The CSDF interpretation mentioned in sec. 5.2 tries to avoid these difficulties by prescribing different token consumption and production phases, but introduces further complexity and

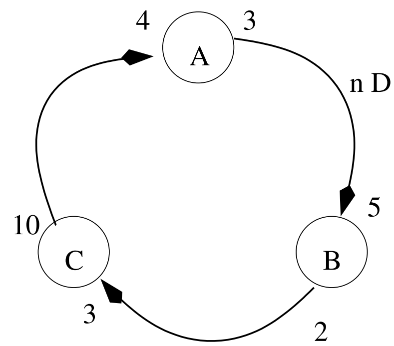



Figure 5: Deadlock in an SDF system: if $n < 10$ the graph deadlocks.

does not provide a complete solution to the timing problem. However, under our new interpretation, as long as each cycle in the graph contains at least one token, deadlock is broken and the system can execute. This is the same condition that applies to homogeneous graphs.

It is important to understand that this interpretation of multirate SDF execution is useful because dedicated hardware implementations of real multirate DSP systems rarely require the interpretation in terms of token consumption of the conventional SDF model. Typical multirate blocks in DSP applications are decimators and interpolators (rate changers), multirate filters (very similar to rate changers), serial-to-parallel converters and vice versa, block coders and decoders, etc. A notable feature of these applications is that few of the applications actually require a consumption of $c$ tokens before starting to produce $p$ tokens. In the case of block coders, in most implementations, for inter-operating with the rest of the system, the data are produced in a periodic stream at a constant sample rate, rather than in large bursts. For serial-to-parallel converters, even though the consumption of all inputs takes place in one burst, the data are still strictly periodic, and by using an appropriate value for the delay through the system, we can accurately model the converter's operation. As a result, the alternative interpretation of SDF execution suggested above is acceptable in most cases, especially when targeting fixed hardware architectures.

## 5.1 HTP Model for Multirate Systems

We now specify how the HTP model can be applied to the analysis of multirate systems. We assume that the multirate system is specified in the SDF formulation. For the execution semantics, we use the assumptions mentioned above: the data on each edge is periodic with a relative sample rate determined by the SDF parameters, and the unit can begin execution after the first input is received, instead of having to wait for $c$ tokens. As mentioned above, this "back-end" assumption is more useful for hardware realizations.

Note that the second assumption can coexist with the conventional SDF semantics. It is always possible to specify a timing delay for the unit that is greater than the time required for $c$ tokens to queue on the input edge. With this execution time, we can be sure that the system will now satisfy traditional SDF semantics in execution. However, in several cases, as pointed out previously, this is a pessimistic requirement, and it is possible to choose a smaller delay that still provides enough time for sufficient samples to be enqueued and for the appropriate computation to occur. As long as we choose the timing delay such that for one period of the samples there is sufficient time between consumption of input and production of the output, the periodicity of the system will ensure that this constraint is met for all successive periods.

For simplicity, we assume that the unit to be modeled is a SISO system and that the propagation delay through sub-units is constant. The model can be extended to handle more complicated units using a modification of the algorithms detailed for homogeneous systems in sec. 4.

Given a multirate system represented as an SDF graph, we follow the usual technique [12] to compute the repetitions vector for the graph. The *balance equation* on each edge $e : u \to v$ in the graph is given by

$$p_e \times q_u = c_e \times q_v, \tag{8}$$

where $p_e$ is the production parameter on $e$, $c_e$ is the consumption parameter, and $q_u$ and $q_v$ are the repetition counts for the source and sink of the edge. Let $T$ denote the overall iteration period of the graph. This is the time required for each actor $x$ to execute $q_x$ times ($q_x$ is the repetition count of the actor). Therefore, the sample period on edge $e$ is given by

$$T_e = \frac{T}{q_u \cdot p_e} = \frac{T}{q_v \cdot c_e}. \tag{9}$$

Now extending the analogy of the homogeneous case, we define the *constraint time* on a path as

$$t_c(p) = \sum_{i=1}^{k} t_i - \sum_{j=1}^{k-1} (d_j \times T_j), \tag{10}$$

where $T_j$ is the sample period on edge $j$. By noting that the effect of a delay on any edge (in both the homogeneous and multirate cases) is to give an offset of $-T_e$ to the constraint time of any path through that edge, we can see that this gives the correct set of constraints. Also, the values of the starting times for the different vertices that are obtained as a solution to the set of constraints will give a valid schedule for the multirate system.

It is possible to view the constraint times in terms of "normalized delays". Here the delays on each edge are normalized to a value of

$$d_n(e) = \frac{d_e}{q_u \cdot p_e} = \frac{d_e}{q_v \cdot c_e}. \tag{11}$$

In terms of the normalized delays, the expression for constraint time becomes the same as that for the homogeneous case.

For homogeneous graphs, the minimum iteration period that can be attained by the system is known as the iteration period bound and is known to be equal to the maximum cycle-mean (MCM) [16, 9]. So far, no such tight bound is known for multirate SDF graphs that does not require the costly conversion to an expanded homogeneous equivalent graph. However, some good approximations for multirate graphs have been proposed [17]. Under our model, it is easy to determine an exact bound that is similar to the bound for homogeneous graphs, but does not require the conversion to a homogeneous equivalent expanded graph. By considering the cumulative constraints around a loop for the single sample-rate case, we can easily obtain the iteration period bound [16]

$$T_{min} = \max_{c \in C} \frac{\sum_c t_u}{\sum_c d_e}, \tag{12}$$

where $C$ is the set of all directed cycles in the graph.

Similarly, for the multirate case, we can obtain the result

$$T_{min} = \max_{c \in C} \frac{\sum_c t_u}{\sum_c d_n(e)}, \tag{13}$$

where $T_{min}$ is the minimum admissible iteration period of the overall system as discussed above. In addition, the start times for each operation are directly obtained as a solution to the constraint system that is set up using the timing information.

One factor here is that, unlike the homogeneous case, the number of timing pairs in the list for a path is not bounded by the number of delay elements. The denominator in a normalized delay is obtained as the product of a repetition count of an actor and the consumption or production parameter for the edge with the delay. As a result, on any path, the total normalized delay is the sum of several terms with such denominators. The number of distinct such terms is therefore bounded by the least common multiple of all these possible denominators. However, in practice, it is rare for several such terms to exist in the timing pair list, so this limit is very pessimistic. One reason for this is that if we consider two paths between a pair of vertices such that one is dominated by the other, then for all paths that pass through these two vertices, only the dominant path can contribute to the timing pairs. Therefore, several paths are eliminated from the possibility of contributing to a timing pair, and the size of the final timing list is quite small. This is observed in the examples we have considered as well.

One possible source of misunderstanding in this context is the use of fractional normalized delays in the computation. It may appear at first sight that the HTP model allows fractional delays to be used in the graph even though such delays have no physical meaning in the context of signal processing. In this context, it is important to remember that the HTP model only specifies information about the *timing* parameters of the graph. The functional correctness of the graph must be verified by other means. In particular, any fractional normalized delays only refer to the fact that the resulting timing shift is a fraction of an iteration period interval, and does not indicate the use of actual fractional delays in a logical sense.

## 5.2 Relation of the HTP Multirate Model to other Models

It is worth taking note of how the HTP model for multirate dataflow graphs differs from some other models that have been proposed to deal with the difficulties posed by the SDF execution semantics.

An important point to keep in mind with regard to our model is that it is targeted towards describing the timing details in an actual hardware implementation of the graph. As discussed in [1], there is a distinction between the implementation dependent physical time consumption, and the implementation independent logical synchronization. Models such as the synchronous reactive systems described in [1] and the CSDF model discussed below are in general more concerned with the implementation independent logical synchronization and in verifying the functional correctness of the system. The model we are proposing, on the other hand, is aimed at the "back-end" of the synthesis process, that deals with the actual mapping to hardware and therefore is concerned with implementation dependent timing parameters. In other words, our model can be used in computing the actual performance metrics and the implementation dependent issues after the functional correctness of the graph has been verified by other means such as the SDF model.

Cyclostatic Dataflow (CSDF) [2] is an extension of SDF that introduces the concept of "execution phases". In this model, each edge has several execution phases associated with it. In each phase, the sink vertex will consume a certain number of tokens as specified by the consumption parameter of that phase for that edge (similarly for the production on the source vertex of the edge). Over a complete iteration, all phases of the edges are completed, and the system returns to its original state. This is therefore a generalization of the SDF execution model. It has been shown [2] that this model can avoid some of the deadlock issues that affect the SDF model (such as in the example of fig. 5), by specifying the phases at which the samples are consumed an produced.

The CSDF model is an execution model, not a timing model. As a result, it leaves the exact time of execution of vertices free, and this is determined by other means, such as run-time resolution of constraints, or construction of a static schedule. The HTP model, on the other hand, is a timing model, and results in static resolution of the times of execution. It also allows the computation of exact values for performance metrics like the iteration period bound. In this sense, the HTP model can in fact be used to complement the CSDF model, by making the assumption that the different phases of execution occur periodically. However, this is complicated a little by the fact that in the CSDF semantics, it is possible for data coming out of a vertex to activate only some of its neighboring vertices in certain phases and others in other phases. Because of this, the simple path based constraint computation that we used to derive the timing pairs in the SDF model may not be enough for CSDF. For example, in the example of the multirate FIR filter (fig. 6), the different polyphase components do not receive the same data in an efficient implementation (redundant computations are avoided). Furthermore, the polyphase filter components receive their inputs at different phases of the input clock. As a result, a time constraint that satisfies the relation between the first output and its corresponding inputs may not satisfy the constraint between the second output and its corresponding inputs. We can handle this situation by inspection, and we can compute the timing pairs of the MR-FIR filter by hand, and use this as a module in other systems such as filter banks. It is also possible to automate this process to check all output phases and ensure that the time delay is sufficient to satisfy all the timing constraints. The basic ideas of timing pairs and timing pair lists that underly the HTP model still hold.

Another model for timing of multirate hardware has been proposed by Horstmannhoff, Grötker and Meyr [8]. They consider an alternative timing model in which the samples occupy different phases of a master clock, and provide interface circuitry to adjust the relative phases when samples are made available on the edges. This requirement of a master clock means that the throughput is already constrained and largely determined beforehand. The interface circuitry can also be quite complex. The model we propose

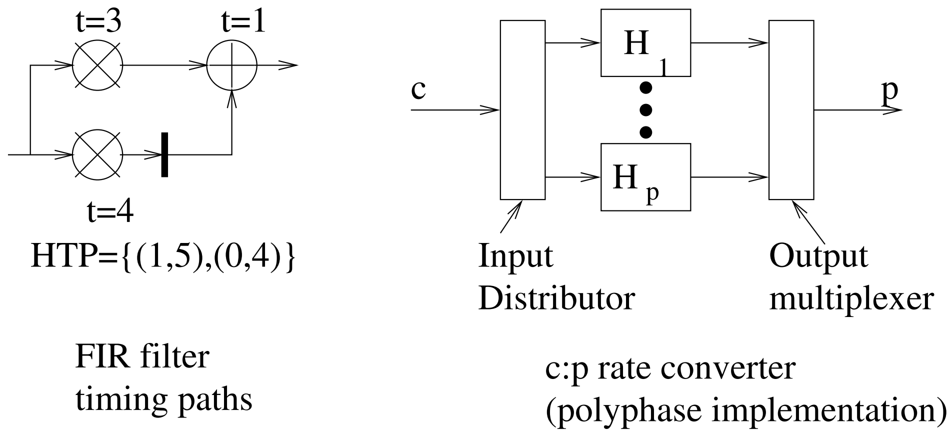



Figure 6: Multirate FIR filter structure.

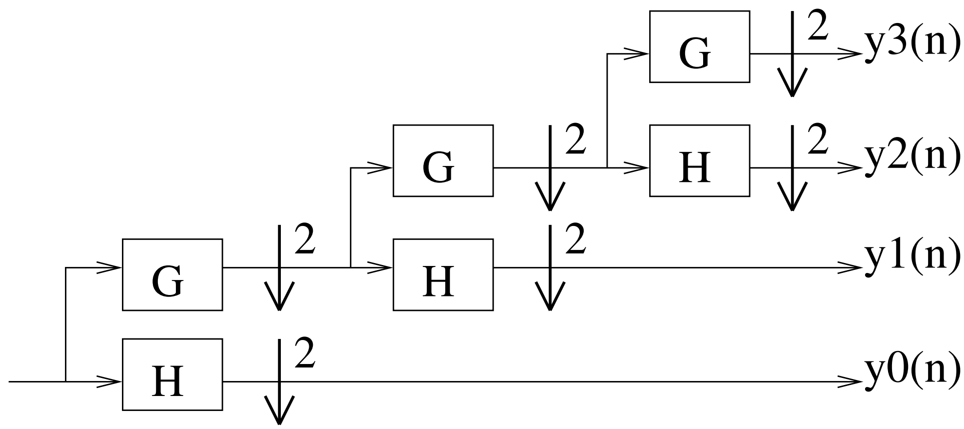



Figure 7: Binary tree structured QMF bank [18].

tries to avoid these problems by not restricting itself to cycle based timing. By enforcing periodicity on the data, it eliminates the need for interface adjustments. An additional benefit of our model is that it allows the iteration period of the system to be adjusted independently of any clock required for cycle-based timing, and also gives an analytical solution for the best iteration period that the system can attain.

# 6 Examples and Results

We have applied the HTP model to the SDF graphs representing typical multirate signal processing applications. The examples we have taken are from the Ptolemy system [3] (CD-DAT, DAT-CD converters and 2 channel non-uniform filter bank) and from [18, p.256] (tree-structured QMF bank).

The basic unit in several of these examples if the multirate FIR filter that is capable of performing rate conversion as described in section 5. As noted there, this must be treated as a primitive element of multirate systems. As shown in Fig. 6, the implementation uses a certain number of internal filters corresponding to the polyphase decomposition of the interpolating filter. We assume that these are implemented in a manner similar to the filter shown on the left of Fig. 6, and that the overall rate converting filter also therefore has similar timing parameters. In particular, we assume for the sake of the other multirate examples that any rate conversion is performed using a filter that has the timing parameters $\{(1,5),(0,4)\}$.

The rate conversions result in several I-O paths with different numbers of delays at different rates. The resulting timing pairs that are obtained for these systems are summarized in Table 2.

| Benchmark | Timing pairs |
|---|---|
| Multirate FIR | $\{(1,5),(0,4)\}$ |
| QMF bank (input to $y_3$) | $\{(7,15),(3,14),(1,13),(0,12)\}$ |
| CD-DAT (160:147) | $\{(93/32,20),(0,16)\}$ |
| DAT-CD (147:160) | $\{(15/7,15),(0,12)\}$ |
| 2 ch. Non.Unif. FB | $\{(5/2,10),(1,9),(0,8)\}$ |

Table 2: Timing pairs for multirate systems.

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

Table 3: Number of dominant timing pairs computed for ISCAS benchmark circuits.

To illustrate the benefit obtained by using the HTP model for hierarchical systems, we have computed timing parameters of a number of homogeneous systems, since even these benefit from the ability of the method to model subsystems.

The HTP model provides a replacement for using the entire subsystem in the timing analysis of a larger system of which the circuit is a part. If we employ conventional models, we would be forced to use the entire graph of the circuit within the larger system. An algorithm such as computing longest input-output paths takes time proportional to $O(|V||E|)$ where $|V|$ is the total number of vertices in the larger system. For example, in a circuit that has 10 blocks connected to each other by 2 edges each, with each block containing 10 vertices and 20 edges, a conventional model would require computation on the order of $100 \times 200$, while the HTP model would require on the order of $10 \times 20 \times |D|$, which is generally a substantial saving since the number of delay elements $|D|$ in a subsystem is usually much less than the number of vertices in the subsystem (in this case 10).

We have run the algorithm described in section 4 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 3 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 is 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 [6]. 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.

A general observation we can make about the timing model is that systems that 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. This is because feedback delay elements must necessarily exist in a loop that has a total negative constraint time, which means they will not contribute towards a dominant constraint time in the forward direction.

# 7 Conclusions and Future Directions

We have presented a timing model for dataflow graphs (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.

For homogeneous graphs, the HTP model allows hierarchical representations of graphs. This results in reducing the amount of information to be processed in analyzing a graph. Alternately, by using this hierarchical representation, the size of the graph that can be analyzed with a given amount of computing power is greatly increased.

The HTP model is able to efficiently store information about multirate graphs, and allows the computation of important system parameters such as the iteration period bound easily. Exact schedules for multirate systems can also be obtained as a solution to the constraints that can be set up using this model. We have shown that the HTP model overcomes many limitations of the conventional timing models, while incurring a negligible complexity increase.

We have considered several typical multirate DSP applications and computed timing pairs for these models. The results demonstrate the power of our approach. We have also considered several homogeneous graphs and shown that the hierarchical aspects of the model can be used to obtain large reductions 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.

The model as it exists now requires the ability to choose the start times of operations (variable phase clocking). We are currently examining ways of extending the model to more general kinds of circuits, which include some fixed phase registers along with other nodes where the delay can be adjusted.

# References


[1] A. Benveniste and G. Berry, "The synchronous approach to reactive and real-time systems," *Proceedings of the IEEE*, vol. 79, no. 9, pp. 1270–1282, Sep 1991.

[2] G. Bilsen, M. Engels, R. Lauwereins, and J. Peperstraete, "Cyclo-static dataflow," *IEEE Transactions on Signal Processing*, vol. 44, no. 2, pp. 397–408, 1996.

[3] J. T. Buck, S. Ha, E. A. Lee, and D. G. Messerschmitt, "Ptolemy: A framework for simulating and prototyping heterogeneous systems," *Int. Jour. Computer Simulation*, vol. 4, pp. 155–182, Apr 1994.

[4] 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://www.cs.umd.edu/TRs/TRumiacs.html*.

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

[6] 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, no. 5, pp. 351–364, May 1992.

[7] J. P. Fishburn, "Clock skew optimization," *IEEE Transactions on Computers*, vol. 39, no. 7, pp. 945–951, Jul 1990.

[8] J. Horstmannshoff, T. Grötker, and H. Meyr, "Mapping multirate dataflow to complex RT level hardware models," in *ASAP '97*, 1997.

[9] K. Ito and K. K. Parhi, "Determining the minimum iteration period of an algorithm," *Journal of VLSI Signal Processing*, vol. 11, pp. 229–244, 1995.

[10] H. V. Jagadish and T. Kailath, "Obtaining schedules for digital systems," *IEEE Transactions on Signal Processing*, vol. 39, no. 10, pp. 2296–2316, Oct 1991.

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

[12] E. A. Lee and D. G. Messerschmitt, "Synchronous data flow," *Proceedings of the IEEE*, vol. 75, no. 9, pp. 1235–1245, Sep 1987.

[13] J. B. Orlin and R. K. Ahuja, "New scaling algorithms for the assignment and minimum cycle mean problems," *Mathematical Programming*, vol. 54, pp. 41–56, 1992.

[14] 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, no. 6, pp. 661–679, Jun 1989.

[15] M. Potkonjak and M. Srivastava, "Behavioral optimization using the manipulation of timing constraints," *IEEE Transactions on Computer Aided Design*, vol. 17, no. 10, pp. 936–947, Oct 1998.

[16] R. Reiter, "Scheduling parallel computations," *Journal of the ACM*, vol. 15, no. 4, pp. 590–599, Oct 1968.

[17] R. Schoenen, V. Zivojnovic, and H. Meyr, "An upper bound of the throughput of multirate multiprocessor schedules," in *ICASSP 97*. IEEE, 1997.

[18] P. P. Vaidyanathan, *Multirate Systems and Filter Banks*, Prentice Hall Signal Processing Series, 1993.