Optimum and Heuristic Transformation Techniques for Simultaneous Optimization of Latency and Throughput

Mani B. Srivastava
AT&T Bell Laboratories
600 Mountain Avenue
Murray Hill, NJ 07974

and

Miodrag Potkonjak

Department of Computer Science
University of California
Los Angeles, CA 90024

Optimum and Heuristic Transformation Techniques for Simultaneous Optimization of Latency and Throughput

KEY WORDS: High-Level Synthesis; Transformational Synthesis;
Performance, Cost, and Power Driven Architectural Synthesis

Abstract

A common metric of speed for DSP systems is their throughput. Algorithm transformations are the key to obtaining high throughput ASIC as well as software implementations. However, increasingly DSP subsystems are being used in systems such as "signal processing servers" and embedded controllers where both throughput and latency are important, and independent, metrics of speed. For example, the subsystem implementing the control law in a robot controller is part of a feedback loop so that not only does it have to process the inputs arriving at a rate determined by the sample period of the control loop, but it also has to produce the output corresponding to an input sample within a specified latency constraint.

Although throughput alone can be arbitrarily improved for several classes of systems using previously published techniques, none of those approaches are effective when latency constraints are considered. After formally establishing the relationship between latency and throughput in general computation, we explore the effect of pipelining on latency, and establish necessary and sufficient conditions under which pipelining does not alter latency.

Many systems are either linear, or have subsystems that are linear. For such cases we have used a state-space based approach that treats various transformations in an integrated fashion, and answers analytically whether it is possible to simultaneously meet any given combination of constraints on latency and throughput. The analytic approach is constructive in nature, and produces a complete implementation when feasibility conditions are fulfilled. We also present a sub-optimal but hardware efficient heuristic approach for the special case of initially-relaxed single-input single-output linear time-invariant computations. A novel software platform consisting of a high-level synthesis system coupled to a symbolic algebra system was used to implement the proposed algorithm transformations.

Instead of optimizing to improve throughput and latency, our transformations can also be used to increase the implementation efficiency while achieving the same latency and throughput as the original design - we obtained large improvements in area and power on many examples when using the proposed transformations in this alternate role.

Optimum and Heuristic Transformation Techniques for Simultaneous Optimization of Latency and Throughput

1.0 Introduction

1.1 Throughput and Latency in System Design

The "open-loop" nature of tasks such as filtering has made throughput the commonly used performance metric for implementation of DSP computations. Many algorithm transformation techniques have been developed to obtain high-throughput implementations of such computation, whether as custom VLSI or as software for programmable processors. Transformations such as lookahead, unfolding, retiming, and algebraic manipulations restructure the original computation algorithm to a functionally equivalent algorithm that can achieve higher throughputs - in some cases even arbitrarily high throughput - although often at the cost of increased implementation cost and latency.

However, increasingly DSP-like computation is being used in systems where both throughput and latency are of importance. One example is that of DSP subsystems that are used in hard real-time embedded systems where the reactive nature of the system imposes hard constraints on throughput as well as latency. For instance, in an embedded system controlling an industrial process, not only does it have to keep up with the rate of arrival of sensory data, but also has to generate the corresponding output within a certain period of time. A proper addressing of these hard real-time constraints is widely recognized as one of the most important research issues [1, 2, 3]. A second example of both latency and throughput being important is that of subsystems used as "signal processing servers" by multiple clients in a distributed environment. In such a case one would like to maximize the throughput of the shared computation hardware while minimizing the latency seen by any single client.

Although throughput alone can be improved using previously published techniques, none of them is effective when latency constraints are simultaneously considered. Algorithm transformations that simultaneously address latency and throughput are the focus of this work.

The terms latency and throughput are used with a variety of different connotations in the high level synthesis literature. In order to facilitate inter-disciplinary research mandated by embedded systems, we use the following well-established definitions from areas such as control theory and digital signal processing. *Latency* is the physical (wall-clock) time needed to deliver the output data from the moment of arrival of the corresponding input data. *Throughput* of an embedded system is the highest rate at which it can receive and process the input data. The inverse of throughput is called *Sample Period*. In Section 3.0 we will recast these definitions in standard high level synthesis and system level design terminology.

1.2 Previous Work

Algorithm transformations are the most powerful algorithm optimization technique in a number of scientific and engineering areas. We will limit our overview to theoretical computer science, VLSI DSP and control theory applications, and compiler and high level synthesis domains - the areas which are most relevant and closely related to our work.

Transformations have been widely used by theoretical computer science researchers in the areas of algorithm design and computer algebra. In algorithm design the emphasis has been on using transformations for the development of generic and powerful parallelization techniques, such as doubling, parallel prefix computation, segmented scan and pointer jumping [4], which can be used for the parallelization of a large variety of important problems. The theoretical study of transformations in computer algebra [5, 6] has mainly concentrated on the development of highly optimized parallel or compact (which use few operations) algorithms for important and popular tasks such as FFT and polynomial computations. On the practical side, several computer algebra manipulation systems that make extensive use of algebraic transformations [7, 8] are in wide use. Example systems include MACSYMA [9], Derive, Reduce [10], Mathematica [11], Maple [12], and Axiom (Scratchpad) [13].

The use of transformations in compiler research and practice [14] has shifted from optimization for vector machines to addressing memory hierarchy access, and superscalar and superpipelined architectures, while preserving short compilation time as one of the most important criteria. Although a variety of transformations has been proposed [15], their interaction has been explored only marginally.

Transformations have long been used in control theory and DSP to derive a variety of functionally equivalent, but structurally different, computation schemes for generic tasks such as filtering [16, 17]. More recently, influenced by the dominance of performance as the key design metric, a significant importance has been placed on unfolding techniques coupled with various algebraic transformations [18, 19, 20, 21] to achieve an arbitrarily high level of throughput at the expense of additional hardware cost, but without taking latency into consideration.

Transformations have also gained importance in the high level synthesis of custom VLSIs where they have been used not only for improving conventional design metrics like area and performance [22, 23, 24, 25, 26, 27], but also for newer metrics such as power [28], and fault tolerance [29]. Sophisticated transformations have also been presented for functional [30] and software [31, 32] pipelining.

1.3 What is New?

We have developed rigorous graph theoretic definitions of latency and throughput for general synchronous computation that also expose the interdependence between these two metrics of speed. Since throughput alone has been extensively treated in literature, we first establish the properties of latency in isolation by studying its exact relationship to pipelining, and by developing a generic state-space based approach to latency minimization that efficiently employs retiming, pipelining, and all algebraic and redundancy manipulation transformations.

Next we study throughput and latency together while paying special attention to the widely used class of *Linear Time-Invariant* (LTI) systems. For this class of systems we have developed a novel transformation technique that restructures the initial computation algorithm to one that has provably optimal latency and throughput, even when no assumption are made about the initial state and values of the coefficients. Until now all approaches which were able to improve throughput to an arbitrary extent were based on a combination of unfolding of the computation with block processing or interleaving [33, 21, 34]. However, this comes at the expense of a proportional degradation in latency. This long-standing *Latency and Sample Period Bottleneck* is broken by employing a novel combination of unfolding with *"On-Arrival Processing"* where input samples are processed as soon as they arrive, and a provably optimal latency minimization technique. For the special case of single-input LTI systems with zero initial state, we also present cost efficient transformation techniques that produce optimal, or close to optimal, combinations of latency and throughput - these special case techniques, however, are numerically not as well behaved as the general technique, and also produce a point solution as opposed to the range of solutions corresponding to different optimal combinations of latency and throughput that can be produced by the general technique.

Our transformation techniques for latency and throughput optimization can be used in two distinct fashion in a design environment. First, and the obvious, use is to transform an initial algorithm that, even given any amount of hardware resources, cannot meet constraints on latency and throughput to a new algorithm that can satisfy the constraints. The second, and equally useful, application of these transformations is to improve the implementation cost even when the initial algorithm can meet the constraints on throughput and latency. In this scenario the transformations are used to obtain a new algorithm with even better latency and throughput characteristics (shorter critical paths) and without increasing the amount of computation and storage too much. This new algorithm gives the scheduler more flexibility in meeting the timing constraints, which can often result in better resource utilization and lower implementation cost (area and power).

In addition to the above theoretical results, the power of coordinated use of high level synthesis tools and symbolic algebraic manipulation systems is demonstrated in the software platform that we developed to implement our techniques. The effectiveness of new concepts, algorithms, and software platform is shown on a variety of real-life examples.

2.0 CDFG and State-Space Representations

2.1 System Representation by CDFG

The systems that we are interested in have multiple inputs, multiple outputs, and finite state. They accept streams of samples on each of the inputs, and produce streams of samples on each of the output ports. We represent an algorithm for a system by a hierarchical directed control-dataflow graph (CDFG). In a CDFG the nodes represent data operators or sub-graphs, data edges represent the flow of data between nodes, and control edges represent sequencing and timing constraints between nodes.

We restrict ourselves to operators that are synchronous in that they consume at every input, and produce at every output, a fixed number of samples on every execution. This restriction has two interesting ramifications. First, the operators, and hence the system, are *determinate* in that a given set of input samples always results in the same outputs independent of the execution times. Second, the system is *well behaved* in that the data sample rate at any given data edge in the CDFG is independent of the inputs, and the ratio between any two data sample rates will be a statically known rational number [35]. Mathematically, such a synchronous CDFG is equivalent to a continuous function over streams of data samples [36, 37]. Since CDFGs are of course causal, this means that they are equivalent to a function that expresses the i-th set of output samples in terms of the i-th and earlier sets of input samples.

The system state is represented in a CDFG by special delay operator nodes which are initialized to a user specified value. A delay operator node (often referred to as just *delay* or *state* in this paper) delays by one sample the stream of data on its sole input port. Intuitively, one can think of delay operators as representing registers holding states, and the other operators as combinational logic.

A system[1] is completely represented by a CDFG and the initial values for all the delays in the CDFG. We further restrict ourselves to single-rate systems where the data rate is identical on all the inputs, and also assume that the i-th samples on all inputs arrive simultaneously, aligned to a “sample clock”. In the rest of the paper we use just the term CDFG to refer to such a “single-rate synchronous CDFG”.

1. We use the term “system” interchangeably to mean “the system behavior specification”, “an algorithm that realizes the system behavior specification”, “an implementation of the system” in different places - the three are not equivalent. A CDFG is a representation of a specific algorithm. There can be multiple algorithms (CDFGs) that express the specified behavior, and there can be multiple implementations (hardware allocation and schedule) of an algorithm (CDFG).

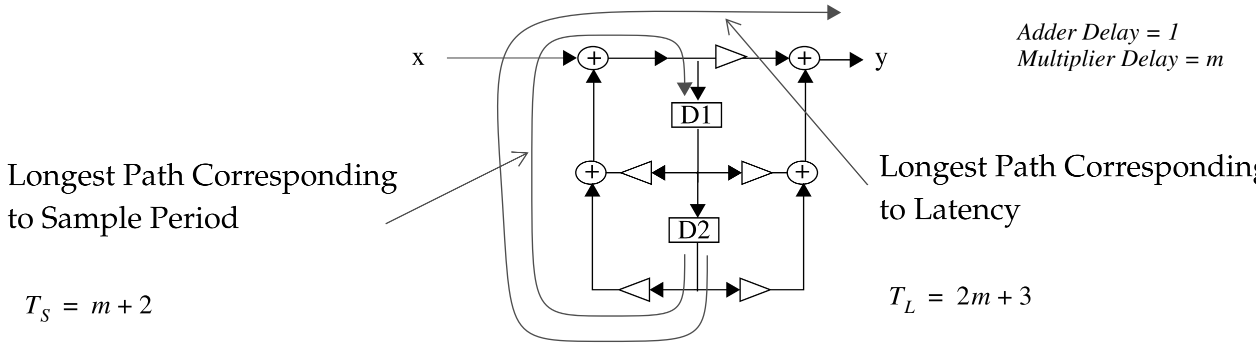


Figure 1: CDFG for a Biquad Filter with Paths Corresponding to Latency T_L and Sample Period T_S

Figure 1 shows an example CDFG.

2.2 System Representation in State-Space

A CDFG corresponds to an algorithm for computing the outputs and the new state (i.e. the new values of the delay nodes in the CDFG) given the inputs and the old state. An analytically powerful expression of such a CDFG is as a discrete-time finite-dimensional state-space system. An important thing to note is that there exists a discrete-time finite-dimensional state-space representation for every CDFG, and every such discrete-time finite-dimensional state-space system can be represented by infinitely many CDFGs. Some properties of the system are better analyzed in a state-space formulation, while others are better analyzed in the CDFG formulation. The state-space formulation is particularly powerful for analysis when the system behavior exhibits linearity, in part because of the well developed theory in that area. We will concern ourselves with the common case of real valued data.

A P-input, Q-output, R-state real-valued CDFG with real-valued data can be equivalently expressed by the following discrete-time finite-dimensional state-space system [38] where X is the input vector, S is the state vector, Y is the output vector, ϕ is the *state-transition mapping*, ρ is the *output mapping*, and n is the time index. $S[-1] \in \Re^{Rx1}$, the initial state when the system starts operating at time 0, is known.

$$\begin{aligned} S[n] &= \phi(S[n-1], X[n], n, S[-1]) \\ Y[n] &= \rho(S[n-1], X[n], n) \end{aligned} \qquad \textbf{(EQ 1)}$$

$$n \in \{0, 1, 2, 3, \ldots\} \qquad X[n] \in \Re^{P \times 1} \qquad S[n] \in \Re^{R \times 1} \qquad Y[n] \in \Re^{Q \times 1}$$

2.3 Special Case: Linear Time-Invariant Systems

A large fraction of systems are either linear, or have subsystems that are linear, over field F, additive operator ⊕, and multiplicative operator ⊗. Typically, F is the field over real numbers $\Re$, ⊕ is arithmetic addition +, and ⊗ is arithmetic multiplication *, as is the case in linear filters in DSP, linear process controllers etc. It is important to note that linearity is not restricted to arithmetic + and *. For example, many useful systems, such as in fuzzy control theory, exhibit linearity over *min* or *max* as the additive operator, and arithmetic addition + as the multiplicative operator.

A system is linear if it can be realized by a CDFG such that at any time instant n, the output samples and next state values are computed by linear combinations of input samples and previous state values. Equivalently, all the operators in the CDFG are either addition of two variables, or addition of a variable and a coefficient, or multiplication of a variable and a coefficient, where coefficients are known functions of n, and independent of the inputs.

An important sub-class of linear systems are linear time-invariant systems, which correspond to all the coefficients in the CDFG being constants, i.e. independent of n. In the state space formalism of (EQ 1), linear time-invariance implies that the state-transition mapping ϕ, and the output mapping ρ are both time-invariant, and linear in $X[n]$ as well as $S[n-1]$ at all $n \in \{0, 1, 2, 3, \ldots\}$. It is well known that this implies that ϕ at a given $n \in \{0, 1, 2, 3, \ldots\}$ can be characterized by constant coefficient matrices $A \in \Re^{R \times R}$ and $B \in \Re^{R \times P}$, and ρ at a given $n \in \{0, 1, 2, 3, \ldots\}$ can be characterized by constant coefficient matrices $C \in \Re^{Q \times R}$ and $D \in \Re^{Q \times P}$ as below:

$$
\begin{aligned}
S[n] &= AS[n-1] + BX[n] \\
Y[n] &= CS[n-1] + DX[n]
\end{aligned}
\qquad \text{(EQ 2)}
$$

$$
n \in \{0, 1, 2, 3, \ldots\} \qquad X[n] \in \Re^{P \times 1} \qquad S[n] \in \Re^{R \times 1} \qquad Y[n] \in \Re^{Q \times 1}
$$

$$
A \in \Re^{R \times R} \qquad B \in \Re^{R \times P} \qquad C \in \Re^{Q \times R} \qquad D \in \Re^{Q \times P} \qquad S[-1] \in \Re^{Rx1}
$$

A further sub-case of the linear time-invariant systems are those systems where the initial state of the system, $S[-1]$, is zero. Such initially relaxed systems have a well developed theory, and in fact most DSP and control systems fall in this category, e.g. the popular 5th order elliptical filter benchmark. These systems are equivalent to linear input-output systems where the value of an output sample can be expressed as a linear combination of the current input sample, and the past values of the input and output samples. Many alternate algorithms (CDFG structures) for these systems are described in the literature and, as discussed in Section 7.0, some of these algorithms achieve excellent latency and throughput. Further, such initially relaxed systems are also amenable to frequency domain analysis - one can characterize them by a Z-transform - and thus optimizations like pole-zero cancellation can also be performed to reduce the computation complexity.

3.0 Metrics of Speed - Throughput and Latency

There are two independent metrics of speed, and a user may specify constraints on both of them as part of the system specification. The two metrics are Throughput, and Latency [39, 2].

Throughput of a system implementation is the maximum rate at which it can accept and process the data samples. The inverse of throughput is the **Sample Period**, T_S, which is the minimum required time between the arrival of successive input samples.

Latency, T_L, of an output in a system implementation is the delay between the arrival of a set of input samples, and the production of the corresponding output as defined by the specification.

Figure 2 shows T_S and T_L pictorially. It is important to realize that T_S and T_L are really independent quantities, and may have independent constraints specified on them. T_L may be less than, equal to, or greater than T_S.

An important point to be noted is that latency is defined in terms of a *correspondence* between input sample(s) and an output sample. For us this correspondence is defined by the initial CDFG given by the user to specify the system - latency of an output in the initial CDFG is the delay between the arrival of the n-th set of input samples, and the production of the n-th sample at the output. Transforming the initial CDFG by adding (or removing) pipeline stages changes the correspondence between the input and output samples. For example, if one were to add one level of pipelining to the initial CDFG, then the latency will be defined in terms of the (n-1)-th set of input samples, and the n-th output sample. Similarly, if the initial CDFG was already pipelined, then removing one level of pipelining will mean that the latency will be defined in terms of the (n+1)-th set of input samples, and the n-th output sample. There is nothing fallacious about this second scenario - if the initial CDFG was already pipelined then it is obvious that the i-th output sample had no causal relationship to the i-th set of input samples.

3.1 Throughput and Latency Achieved by a CDFG

The preceding definitions of latency and throughput (sample period) can be recast in standard high level synthesis and system level design terminology. In particular, the definitions can be used to find the latency and sample period that can be achieved by a particular CDFG (no transformations allowed to the CDFG) given any amount of hardware resources and using any implementation technique. We will show that in the general case the latency and sample period that can be achieved by a CDFG forms a range of solutions that is parameterized by certain

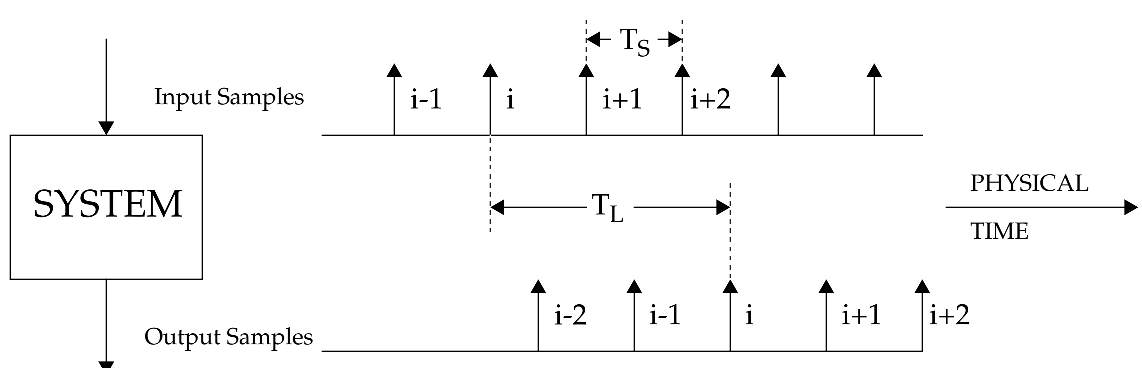


Figure 2: Definitions of Sample Period and Latency

variables that an implementation is in general free to choose.

To get an intuitive feeling, let us first consider the simple case where the implementation is restricted to those where, in terminology of the state-space representation of (EQ 1), the P elements of the n-th input vector $X[n]$ and the Q elements of the corresponding previous state vector $S[n-1]$ are all available at the same time - in other words, there is no time skew between them. These elements are used to calculate the n-th output vector $Y[n]$ and the n-th state vector $S[n]$. While this implementation model may sound restrictive, it is in fact the one that is assumed by the scheduler in most of the popular high-level synthesis tools such as HYPER [24]. Latency T_L of a particular primary output is the number of control steps that are needed to generate an output sample since the arrival of corresponding input samples. In our restrictive implementation model where the previous set of state values is available at the same time as the arrival of the new set of input values, latency T_L will be equivalent to kT_S + *the length of the longest computation path from any primary input or state to the primary output*. The path lengths are measured in control steps, and k is the number of pipeline stages that have been added (or removed if k is negative) to the initial specification. In a similar vein, T_S is *the length of the longest computation path measured in control steps from any primary input or state to any state*. One may wonder why T_S is not dependent on computation paths that go from primary inputs or states to primary outputs. The reason is that if the computation delay in any of these paths is greater than T_S, an implementation can always use parallel or pipelined hardware to overlap computation corresponding more than one successive output samples, and achieve a sample period of T_S. This will become clearer in Section 4.0 where the relationship between latency and pipelining is explored. Figure 1 illustrates the longest paths corresponding to latency and sample period for a biquad filter.

Having gained an intuitive understanding of latency and sample period achieved by a CDFG under a simple implementation model, we will now develop exact expressions for achievable latency and sample period under a more general implementation model where we remove the restriction that all the elements of $X[n]$ and $S[n-1]$ are available at the same time.

Consider a CDFG with P inputs, Q outputs, and R state nodes. Using the same notation as in (EQ 1), let:

$P_{IS}(i,j)$ = length of the path (in control steps) from the i-th primary input node to the j-th state node

$P_{IO}(i,j)$ = length of the path (in control steps) from the i-th primary input node to the j-th primary output node

$P_{SS}(i,j)$ = length of the path (in control steps) from the i-th state node to the j-th state node

$P_{SO}(i,j)$ = length of the path (in control steps) from the i-th state node to the j-th primary output node k = number of pipeline stages that have been added (or removed if k<0) to the initial CDFG that was given by the user as a specification, and from which the current CDFG under consideration was obtained after some transformations

$T_{IA}(i)$ = skew in the arrival of the i-th element of $X[n]$ (n-th sample at the i-th input) relative to arrival of the 1-st element of $X[n]$ (n-th sample at the 1-st input). Note that by definition $T_{IA}(1)=0$.

$T_{SA}(i)$ = skew in the arrival of the i-th element of $S[n-1]$ ((n-1)-th value at the i-th state node) relative to arrival of the 1-st element of $X[n]$ (n-th sample at the 1-st input)

T_S = sample period

$T_L(i,j)$ = latency from i-th input node to the j-th output node

Note that the input arrival time skews, $T_{IA}(i)$ $\forall$ $i \in 1..P$, are usually timing constraints that are in general specified by the user. On the other hand, the skews in state arrival, $T_{SA}(i)$ $\forall$ $i \in 1..R$, are parameters that an implementation is free to choose so as to satisfy design constraints while optimizing design cost metrics.

A little thought shows that:

$$T_S = \max \{P_{SS}(r,s)+T_{SA}(r)-T_{SA}(s)\ \forall r,s \in 1..R\} \cup \{P_{IS}(p,r)+T_{IA}(p)-T_{SA}(r)\ \forall p \in 1..P,\ \forall r \in 1..R\}$$

$$T_L(i,j) = k*T_S + \max \{P_{IO}(p,j)+T_{IA}(p)-T_{IA}(i)\ \forall p \in 1..P\} \cup \{P_{SO}(r,j)+T_{SA}(r)-T_{IA}(i)\ \forall r \in 1..R\}$$
$$\forall i \in 1..P \text{ and } \forall j \in 1..Q$$

The above expressions clearly show that the achievable values of T_S and $T_L(i,j)$ are coupled in general - choosing some may place constraints on the achievable values of the remainder.

As mentioned earlier, most high level synthesis systems, such as HYPER [24], assume an implementation model where $T_{IA}(i)=0$ $\forall$ $i \in 1..P$, and $T_{SA}(i)=0$ $\forall$ $i \in 1..R$. Under such a model the above expressions for T_S and $T_L(i,j)$ get simplified to the following expressions that are equivalent to the intuitive expressions that we derived earlier (note that $T_L(i,j)$ is now independent of i):

$$T_S = \max \{P_{SS}(r,s)\ \forall r,s \in 1..R\} \cup \{P_{IS}(p,r)\ \forall p \in 1..P,\ \forall r \in 1..R\}$$

$$T_L(i,j) = k*T_S + \max \{P_{IO}(p,j)\ \forall p \in 1..P\} \cup \{P_{SO}(r,j)\ \forall r \in 1..R\}$$
$$\forall i \in 1..P \text{ and } \forall j \in 1..Q$$

To avoid the analysis from becoming complicated, we also do not use the general implementation model where $T_{IA}(i)$ $\forall$ $i \in 1..P$, and $T_{SA}(i)$ $\forall$ $i \in 1..R$ are allowed to hold completely arbitrary values. While in some places we do use the more common and restricted model $T_{IA}(i)=0$ $\forall$ $i \in 1..P$, and $T_{SA}(i)=0$ $\forall$ $i \in 1..R$, in most of the paper we use a more permissive implementation model where $T_{IA}(i)=0$ $\forall$ $i \in 1..P$, $T_{SA}(i)=T_{SA}$ $\forall$ $i \in 1..R$, and T_{SA} is a single implementation timing parameter[2]. Intuitively, in this relaxed model it is assumed that the i-th sample arrives at the same time for all the inputs, and the previously state values arrive at the

2. The parameter Tj used in analysis later in the paper (Section 6.0 in particular) is similar to the parameter T_{SA} here.

same time for all the state nodes, and that the arrival of the state values is skewed with respect to the arrival of the input values by the time interval T_{SA}. We found that the ability to choose the state arrival skew T_{SA} (as opposed to assuming it to be 0 as the more common model does) is of enormous help - it enables combinations of latency and throughput to be achieved that are otherwise not possible. As later sections will show, this parameter plays a key role in our techniques in Section 6.3. For this model the expressions for T_S and $T_L(i,j)$ get simplified to (note that $T_L(i,j)$ is independent of i in this case too):

$$T_S = \max \{P_{SS}(r,s)\ \forall r,s \in 1..R\} \cup \{P_{IS}(p,r) - T_{SA}\ \forall p \in 1..P,\ \forall r \in 1..R\}$$

$$T_L(i,j) = k * T_S + \max \{P_{IO}(p,j)\ \forall p \in 1..P\} \cup \{P_{SO}(r,j) + T_{SA}\ \forall r \in 1..R\}$$
$$\forall i \in 1..P \text{ and } \forall j \in 1..Q$$

4.0 Pipelining - Latency Relationship

(Functional) pipelining is an implementation technique where several consecutive instances of the program body are partially overlapped during execution. It is one of the most often used performance optimization technique, as well as one of the most often addressed transformation in high level synthesis research. For extensive coverage of pipelining in general computing literature we refer to [33, 40], and in high level synthesis to [30, 41]. We will use the following definition of pipelining: *Pipelining* with k pipeline stages on a CDFG is a special form of retiming where on each primary output (or input) k new delays are introduced, and only those delays can be moved.

The widespread treatment and use of pipelining is a consequence of its extraordinary power in improving throughput of designs. In particular, it is effective, when feedback constraints are not present [42]. However, more detailed analysis shows that pipelining also has side-effects which can drastically deteriorate quality of design. For example, it has been observed that pipelining often significantly increases the register requirements [41]. Increased latency is the most often quoted as an unavoidable harmful side effect of pipelining. However, as we will demonstrate in the rest of this section, if pipelining is done in a particular manner (so that a set of simple, easy to satisfy, timing constraints are satisfied), this effect can be fully avoided.

We assume that the user wants to introduce k delays, which will partition the CDFG into k+1 pipeline stage. The length of the longest path that is relevant for latency calculation (i.e., the longest path from any of the delay nodes or primary inputs to the output) is denoted by T_L. Let T_S be the sample period. The following theorem establishes conditions under which pipelining can be done without altering latency:

Pipelining-Latency Theorem: *A necessary condition that pipelining does not increase latency is that k, the number of pipeline delays introduced, satisfies* $k \leq \lfloor T_L / T_S \rfloor$ *. Further, if micropipelining (fine-grained pipelining) of individual operations is allowed, or if all operations have equal delays, then the above* *condition is also a sufficient one.*

Proof: The latency T_{LP} of the pipelined CDFG will be at least kT_S . If the condition specified in the theorem does not hold, i.e. if $k \geq \lfloor T_L/T_S \rfloor + 1$ (note: that k is an integer), then $T_{LP} \geq kT_S \geq (\lfloor T_L/T_S \rfloor + 1) T_S > (T_L/T_S - 1 + 1) T_S = T_L$, which contradicts the hypothesis that latency will not increase.

In the case of operators that can be pipelined at a fine grain, as well as in the case of operators with identical delays, the following placement of pipeline delays (latches) will maintain the latency. Introduce the k pipeline delays one by one such that the i-th delay, $\forall i \in \{1 \ldots k\}$, is placed at a distance of $i \times T_S$ from the input - in other words, the pipeline delays are placed separated by T_S at $T_S, 2T_S, 3T_S, \ldots, kT_S$. This placement of pipeline delays will result in a latency of $k \times T_S + (T_L - k \times T_S) = T_L$ because it takes time $k \times T_S$ for an input sample to travel to the k-th pipeline delay, and a further time of $T_L - k \times T_S$ to travel from the k-th pipeline delay to the output. Such a placement is always feasible if fine-grained pipelining of operators is allowed, or if all operators have equal delays (in which case T_S will be a multiple of operator delay). On the other hand if neither of these prerequisites are not met then one may not be able to place the pipeline delays such that they are separated exactly by T_S . Observe that one cannot place adjacent pipeline delays at a distance greater than T_S because that will increase the sample period, and one should not place them at a distance smaller than T_S because a time interval of T_S is always devoted to the computation between adjacent pipeline delays - placing adjacent delays closer than T_S will therefore result in an increase in the latency.

We end this section with the observation that in the case of CDFGs that correspond to linear digital signal processing and control systems, pipelining with k delays is equivalent to introducing k poles at the origin in the transfer function of the system. If such a system is embedded in a feedback path of a larger system then the addition of these extra poles due to pipelining can have an effect on the stability and functionality of the larger system. Therefore, in such cases, the use of pipelining requires caution.

5.0 Techniques to Reduce Latency of a System

Recall from Section 3.0 that the latency that can be achieved by a CDFG is determined by the length of the longest combinational path that originates at a state (delay node) or at a primary input, and ends at the primary output of interest. Therefore, intuitively, to minimize latency one should try to reduce the length of such paths by suitably transforming the initial CDFG.

To reduce the length of any computation path going from a primary input to a primary output, transformations based on algebraic properties such as associativity, commutativity, and distributivity can help. For example, a chain of n adders that adds n+1 variables can be transformed into a maximally balanced binary tree of adders of depth $\lceil \log_2 (n+1) \rceil$. Retiming cannot help in reducing such input-output paths because they do not have any delay nodes to begin with, and no amount of retiming will change that. Introduction of pipeline stages is possible but this can never reduce the length of these computation paths, although, as shown in Section 4.0, it is not necessary that this will increase the latency either.

More options are available to reduce the length of computation paths that originate at state nodes and end at primary outputs. Algebraic transformations will of course help, but another technique that helps is to retime such that the state nodes are moved closer to the primary outputs. Intuitively this means that we precalculate as much of the contribution of the previous state values to a primary output as possible. However, retiming and algebraic transformations effect each other mutually so that this is not a straightforward optimization. Further, retiming to minimize latency may result in the sample period getting worse because computation paths originating at primary inputs or at state nodes, and ending at state nodes, may be elongated.

Fortunately, in certain special but important cases an analysis in the state space framework offers more insight than the CDFG framework. This is because in the state space framework, as shown in (EQ 1), all the algebraic properties are encapsulated into two functions - the *state-transition mapping* ϕ, and the *output mapping* ρ. Retiming can be viewed in the state space as mapping the old state vector in to a new one such that there is no change in the behavior of the system observable at the outputs. This mapping of state vector results in new ϕ and ρ being defined. One can thus analyze the effects of retiming and algebraic transformations in an integrated fashion.

For example, consider the case where ρ is separable such that

$$\rho(S[n-1], X[n], n) = \rho 1(S[n-1], n) \oplus \rho 2(S[n-1], X[n], n)$$

and where the time taken to calculate $\rho 2$ and $\oplus$ in sequence is less than that taken to calculate ρ alone. Then one can retime such that the term $\rho 1(S[n-1], n)$ is available as a state, and as a result the latency is reduced. The new state equations will then be:

$$S[n] = \phi(S[n-1], X[n], n, S[-1]), \qquad \tilde{S}[n] = \rho 1(\phi(S[n-1], X[n], n, S[-1]), n+1)$$

$$Y[n] = \tilde{S}[n-1] \oplus \rho 2(S[n-1], X[n], n)$$

$$n \in \{0, 1, 2, 3, \ldots\} \qquad X[n] \in \Re^{P \times 1} \qquad S[n] \in \Re^{R \times 1} \qquad \tilde{S}[n] \in \Re^{1 \times 1} \qquad Y[n] \in \Re^{Q \times 1}$$

However, this can increase the sample period because the time taken to calculate $\rho 1(\phi(\ldots), \ldots)$ will in general be greater than that taken to calculate ϕ alone.

Many important systems have ρ that exhibit such separability. For example, polynomial filters such as Volterra filters have ρ which is a multivariate polynomial of states and primary inputs, and is therefore separable as above. Linear systems, where ρ is characterized by matrices, also exhibit this separability. However, because ϕ is also linear in such systems, the sample period is not increased when the system is transformed as above to minimize latency. In fact, as shown in the next section, one can go a step further and simultaneously apply algorithm transformations such as unfolding that reduce the sample period.

6.0 Latency and Sample Period for Linear Time-Invariant Systems

CDFGs that are composed of variable-variable additions and variable-constant multiplications correspond to linear time-invariant systems. Although much work has been done towards algorithms for these systems that achieve low sample periods (high throughput), no work has been directed towards reducing both latency and sample period. In fact, some of the techniques used for improving the sample period end up making the latency worse. In this section we analyze linear time-invariant systems from the perspective of both latency and sample period, and present transformations that simultaneously address these two metrics.

6.1 Latency and Sample Period of a Linear Time-Invariant CDFG

Consider a CDFG with P primary inputs, Q primary outputs, R state nodes, and composed of additions of two variables, and multiplications of variables with constant coefficients. Its state-space representation is as shown in (EQ 2) where $X[n]$ is the current value of primary inputs in a vector form, $S[n-1]$ is the vector corresponding to the state values from the previous time step, and $Y[n]$ and $S[n]$ are the values of the output vector and state vector calculated in the current time step. Further, we assume the restricted implementation model (see discussion in Section 3.1) where all the elements of $X[n]$ and $S[n-1]$ are available at the same time.

A key advantage of the state-space representation is that the specific organization of linear computation in the CDFG is abstracted by the two linear equations, $S[n] = AS[n-1] + BX[n]$ (called the state update equation) and $Y[n] = CS[n-1] + DX[n]$ (called the output equation), which encapsulate all the algebraic information. From the definitions of sample period T_S and latency T_L in Section 3.0, it follows that T_S is decided by the time taken to compute $S[n] = AS[n-1] + BX[n]$, and T_L is decided by the time taken to compute $Y[n] = CS[n-1] + DX[n]$. These matrix equations are equivalent to a set of equations where the right hand sides are linear combinations of the P primary inputs and the R state values.

Noting that one of the maximally fast ways to evaluate a linear combination is by first doing the constant-variable multiplications in parallel, and then organizing the additions as a maximally balanced binary tree, we get the following expressions for the best sample period and latency that can be achieved when the four constant coefficient matrices are non-trivial (i.e., if we do not assume that any elements may be 0 or 1 or -1):

$$T_S = m + \lceil \log_2(R+P) \rceil \quad \text{and} \quad T_L = m + \lceil \log_2(R+P) \rceil \qquad \textbf{(EQ 3)}$$

where we assume that the time for an addition is 1 and the time for a multiplication is $m \geq 1$ [3].

These expressions, however, can in general be quite pessimistic because the four coefficient matrices often have many elements that are 0 or 1 or -1, in which case one can take advantage of such coefficients to reduce the number of adders and multipliers that are required. If we define $\mathrm{nr0}(M, r)$ and $\mathrm{nr1}(M, r)$ to be the number of elements that have magnitude 0 and 1 respectively in the $r-th$ row of matrix M, then the following exact expressions for best achievable latency and sample period are obtained[4]:

$$T_S = \max_{r \in \{1 \dots R\}} \left(m + \left\lceil \log_2 \left(R + P - \mathrm{nr0}(A, r) - \mathrm{nr0}(B, r) - (\mathrm{nr1}(A, r) + \mathrm{nr1}(B, r))\,(1 - 1/2^m)\right)\right\rceil\right) \quad \text{(EQ 4)}$$
$$T_L = \max_{r \in \{1 \dots Q\}} \left(m + \left\lceil \log_2 \left(R + P - \mathrm{nr0}(C, r) - \mathrm{nr0}(D, r) - (\mathrm{nr1}(C, r) + \mathrm{nr1}(D, r))\,(1 - 1/2^m)\right)\right\rceil\right)$$

Since the number of inputs P and number of outputs Q are fixed, the goal of an algorithm transformation that optimizes latency and sample period will intuitively be to reduce R and increase the number of 0, 1, and -1 valued entries across all rows of the coefficient matrices.

6.2 Transforming a Linear Time-Invariant CDFG for Minimum Latency

From the state-space representation of a LTI CDFG it becomes obvious that no amount of retiming and algebraic transformations will ever change the matrix D. Retiming has no effect because D corresponds to paths in the LTI CDFG that go from input nodes to output nodes without passing through state nodes - there are no state nodes to retime with. Algebraic transformations have no effect because D is the matrix representation of the set of linear expressions that is implied by those input-output paths, and the matrix representation of a set of linear expressions is unique and unaffected by the application of commutativity, distributivity, associativity etc. Therefore, from (EQ 4) for T_L it follows that if one is able to transform the algorithm such that every row of the transformed version of matrix C has one entry with value 1 and all other entries with value 0, then the value of T_L is the minimum possible. Intuitively, the new algorithm will be such that each output depends on one and only one state variable, and that too through a coefficient of 1. We show below that any LTI CDFG can indeed be transformed to such a minimum latency realization. Note that this solution is not unique, and is not necessarily the most efficient one either, because there exist infinitely many other solutions with minimum latency. For example, one can obtain other solutions with minimum latency by first applying some arbitrary state space transformation (a different state encoding) such that one still has R state values (infinitely many such transformations are possible because such a transformation is equivalent to an invertible RxR matrix; please refer to any linear systems books, such as [38]), and then apply the outlined algorithm.

3. Note that this is a reasonable assumption because the latency of a multiplication operation is in almost all cases - ASICs or programmable processors - larger than or equal to that of an addition operation. For example, in the case of fixed-point arithmetic hardware used in most ASIC implementations, the latency of an adder is typically one clock cycle whereas the latency of a multiplier is several clock cycles (pipelining a multiplier, as is commonly done, only improves the throughput).

4. The unusual looking $(1-1/2^m)$ terms trace their origin to the fact that during the first m time units the variables with coefficients of magnitude 1 can be begun to be added in a maximally fast fashion in parallel with the multiplication of the other variables with their non-trivial (not 0, 1, or -1) coefficients.

Consider a transformation of the original algorithm such that not only do we have the original R state variables, but also Q linear combinations of those R state variables that are obtained by taking the inner product of the original state vector with the rows of matrix C. If these Q new states are denoted by $\tilde{S}$, then the following state space representation is obtained which is equivalent to the original system in input-output behavior, and is a minimum latency realization because each output depends on one and only one state via a coefficient of value 1:

$$\begin{bmatrix} S[n] \\ \tilde{S}[n] \end{bmatrix} = \begin{bmatrix} A & 0_{RxQ} \\ CA & 0_{QxQ} \end{bmatrix} \begin{bmatrix} S[n-1] \\ \tilde{S}[n-1] \end{bmatrix} + \begin{bmatrix} B \\ CB \end{bmatrix} X[n]$$

$$Y[n] = \begin{bmatrix} 0_{QxR} & I_{QxQ} \end{bmatrix} \begin{bmatrix} S[n-1] \\ \tilde{S}[n-1] \end{bmatrix} + DX[n] \tag{EQ 5}$$

$$\text{where: } \begin{bmatrix} S[-1] \\ \tilde{S}[-1] \end{bmatrix} = \begin{bmatrix} I_{RxR} \\ C \end{bmatrix} S[-1]$$

In the general case where the initial coefficient matrices are non-trivial, i.e. the matrix elements are not trivially 0 or 1, the above transformation guarantees that given enough hardware one can always achieve the following sample period and latency:

$$T_S = m + \lceil \log_2 (R+P) \rceil \quad \text{and} \quad T_L = m + \lceil \log_2 (1+P) \rceil \tag{EQ 6}$$

where the time for an addition is 1 and the time for a multiplication is m. For example, using this minimum latency transformation on a single input system, a sample period better than $m + \lceil \log_2 (R+1) \rceil$ and a latency better than $m+1$ time units can always be achieved.

6.3 Transforming a Linear Time-Invariant CDFG to Jointly Improve Latency & Sample Period

Previous research [18, 21] has shown that one can use unfolding, look-ahead, and block-processing techniques to arbitrarily improve the sample period of LTI systems. On the other hand we just showed that one can always transform LTI systems to attain the minimum possible latency $T_L = m + \lceil \log_2 (1+P) \rceil$. Can we combine the two techniques to arbitrarily improve the sample period and simultaneously achieve the minimum latency? Unfortunately, the answer turns out to be no in the general case - for any given sample period there is a limit on the best latency that can be achieved, and this limit depends on the number of primary inputs. In the following discussion, we not only find the bounds, but in the process also generate a new CDFG, i.e. a new algorithm, that achieves the bound. This new CDFG can then be used as a starting point for scheduling and resource allocation.

In the following analysis we make two reasonable assumptions. First, time is taken to be an integer, and is measured in units of adder delay. The delay for a multiplication is a multiple

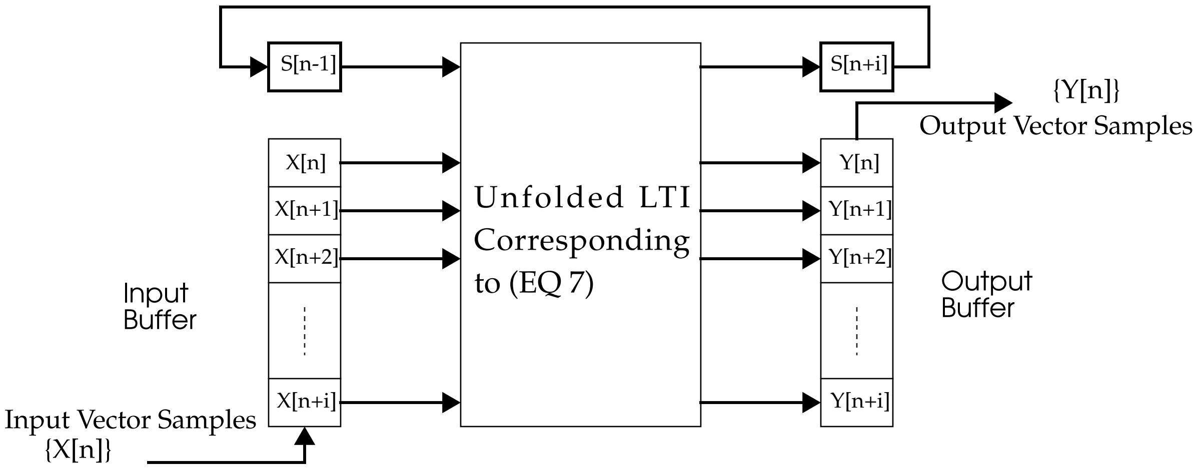


Figure 3: Block Processing

$m \geq 1$ of the adder delay. In effect we assume that addition takes one clock cycle, and multiplication takes one or more clock cycles. Second, the system is assumed to be arbitrary and non-trivial in the sense that no assumption is made about the values of the elements of the four coefficient matrices in the state space representation - in particular, we do not exploit coefficients that are 0 or 1. This makes the mathematics tractable, and the results conservative, at the expense of not being able to generate an analytic expression for the exact amount of hardware resources that are needed. However, this is not a problem because scheduling and allocation for a CDFG can be performed efficiently by various high-level synthesis systems.

6.3.1 Using Unfolding with Block Processing to Arbitrarily Improve the Sample Period

All algorithm transformations that can arbitrarily improve sample period are based on variants of unfolding where several input samples are processed together to produce one or several output samples. The computation overhead is amortized over several samples, and thus the effective sample rate is lowered. Further, the next state is now calculated in steps of the block size. Block processing [43] is one technique based on this theme where several samples are buffered, and then processed together as a block to produce the corresponding outputs. We use an adaptation of this idea for simultaneous improvement in latency and sample period, and therefore it is illustrative to look at how block processing works.

Figure 3 shows the idea behind block processing. In the original CDFG, $X[n]$ and $S[n-1]$ are used to compute $Y[n]$ and $S[n]$. In block processing, $i+1$ consecutive input samples $X[n] \ldots X[n+i]$ are collected in a buffer, and used together with $S[n-1]$ to compute the corresponding output samples $Y[n] \ldots Y[n+i]$ and the new state $S[n+i]$. The $i+1$ output samples are put in a buffer and shifted out at the sample rate. Here $i+1$ is the blocking size. The state space formalism again provides the means to calculate the sample period and latency that are achieved by block processing. Following are the equations for $Y[n] \ldots Y[n+i]$ and $S[n+i]$ in terms of $X[n] \ldots X[n+i]$ and $S[n-1]$ as obtained by unfolding the original state equations:

$$\begin{aligned} S[n+i] &= A^{i+1}S[n-1] + A^{i}BX[n] + A^{i-1}BX[n+1] + \ldots + BX[n+i] \\ Y[n] &= CS[n-1] + DX[n] \\ Y[n+1] &= CAS[n-1] + CBX[n] + DX[n+1] \\ &\ldots \\ Y[n+i] &= CA^{i}S[n-1] + CA^{i-1}BX[n] + CA^{i-2}BX[n-1] \ldots + CBX[n+i-1] + DX[n+i] \end{aligned} \quad \text{(EQ 7)}$$

Noting that the buffering delays at the input and the output affect the latency, and that the effective sample period is the block processing time divided by the block size, one obtains the following expressions for the latency and sample period for such a block processing algorithm:

$$\begin{aligned} T_S &= \frac{m + \lceil \log_2(R + (i+1)P) \rceil}{i+1} \\ T_L &= (2i+1)T_S = \left(\frac{2i+1}{i+1}\right)(m + \lceil \log_2(R + (i+1)P) \rceil) \end{aligned} \quad \text{(EQ 8)}$$

These expressions are obtained by organizing the computation of the various linear combinations as maximally balanced binary trees of adders, preceded by multiplications in parallel. As is obvious from these expression, $\lim_{i \to \infty} T_S = 0$, and $\lim_{i \to \infty} T_L = \infty$, so that throughput can be improved arbitrarily by increasing the amount of unfolding, but always at the cost of increased latency. This is a result of the buffering done at the input and the output.

As a side note, it is important to note that unfolding does not hurt numerical properties - in fact, as shown in [43] there is actually an improvement in round-off errors and other finite precision arithmetic effects.

6.3.2 Using Unfolding with On-Arrival Processing and Minimum Latency Transformation to Simultaneously Optimize Latency and Sample Period

We have found that the key to simultaneously improving sample period and latency is to combine the minimum latency transformation from Section 6.2 with unfolding. However instead of using block processing, which always degrades the latency, the samples are no longer buffered, but are processed as they arrive. Intuitively this makes sense too - if latency is a concern, there is no point in idle buffering of input samples.

One strategy for bringing the unfolded system into a minimum latency form is the same as that we adopted for the system which was not unfolded - new states are introduced such that all the outputs are dependent on one and only one state variable. A quick look at the unfolded system equations shows that if the linear combinations CS, CAS, $\ldots$, $CA^{i}S$ of the original state vector S are added as new, though redundant, states then one can express all the outputs such that they depend on one and only one previous state value, and that too through a multiplicative coefficient of 1. The number of states increases to $R + iQ$. However, since the dimension of state space is not more than R, it is possible to delete some of the original R states such that the

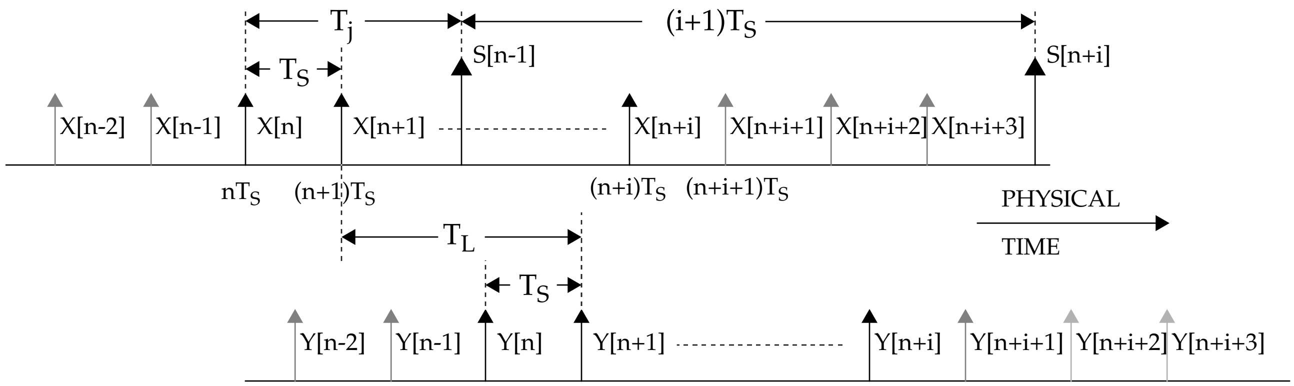


Figure 4: Unfolding + On-Arrival Processing

remaining states continue to form a basis set for the state space. To keep the analysis simple, we avoid doing this optimization as the only effect of it will be to reduce the amount of hardware resources that will be needed - the critical paths in computation will remain unchanged. Using this strategy of combining unfolding and minimum latency transformation we get the following system equations:

$$\tilde{S}[n] = \begin{bmatrix} I_{RxR} \\ C \\ \dots \\ CA^i \end{bmatrix} S[n] = \begin{bmatrix} A^{i+1} & 0_{RxQ} & 0_{RxQ} & \dots & 0_{RxQ} \\ CA^{i+1} & 0_{QxQ} & 0_{QxQ} & \dots & 0_{QxQ} \\ \dots & \dots & \dots & \dots & \dots \\ CA^{2i+1} & 0_{QxQ} & 0_{QxQ} & \dots & 0_{QxQ} \end{bmatrix} \tilde{S}[n-1] + \begin{bmatrix} A^iB \\ CA^iB \\ \dots \\ CA^{2i}B \end{bmatrix} X[n] + \dots + \begin{bmatrix} B \\ CB \\ \dots \\ CA^iB \end{bmatrix} X[n+i]$$

$$Y[n] = \begin{bmatrix} 0_{QxR} & I_{QxQ} & 0_{QxQ} & \dots & 0_{QxQ} \end{bmatrix} \tilde{S}[n-1] + DX[n] \qquad \text{(EQ 9)}$$

$$Y[n+1] = \begin{bmatrix} 0_{QxR} & 0_{QxQ} & I_{QxQ} & \dots & 0_{QxQ} \end{bmatrix} \tilde{S}[n-1] + CBX[n] + DX[n+1]$$

$$\dots$$

$$Y[n+i] = \begin{bmatrix} 0_{QxR} & 0_{QxQ} & 0_{QxQ} & \dots & I_{QxQ} \end{bmatrix} \tilde{S}[n-1] + CA^{i-1}BX[n] + \dots + CBX[n+i-1] + DX[n+i]$$

Since we are interested in finding the limits to which the sample period and latency can be improved simultaneously, our task is essentially one of scheduling the above computation such that we get minimum latency for a given sample period. Such a schedule, while requiring much hardware, will be the fastest. If T_S be the sample period (an integer >= 1), then the arrival time of input samples $X[n]\dots X[n+i]$ are $nT_S\dots(n+i)T_S$. This computation is not a one shot computation in the sense that computation for one set of $i+1$ samples is followed by the computation of the next set of $i+1$ samples, so that one can overlap these computations under the constraint that the state value needed by one set of computations is produced in the preceding set. To put it another way, the time at which $\tilde{S}[n-1]$ is available is also a parameter. Let the arrival of $\tilde{S}[n-1]$ be skewed by T_j with respect to the arrival of $X[n]$, or equivalently, let $\tilde{S}[n-1]$ be available at $nT_S + T_j$. Since a similar computation needs to be done for the next block of $i+1$ samples, it follows that $\tilde{S}[n+i]$ has to be available by $(n+i+1)T_S + T_j$. This is pictorially depicted in Figure 4. Note that T_j may even be negative, although since $\tilde{S}[n-1]$ depends on $X[n-1]$, T_j obviously cannot be less than $-T_S$; in fact as later results in the paper show, this is not a sufficient condition for the implementation to be feasible.

From the preceding discussion it is clear that for a given LTI system with sample period T_S, these two parameters - the amount of unfolding i, and the skew T_j in the computation of state vector - together span the range of possible realizations obtained as a result of unfolding. However, the constraint that $\tilde{S}[n+i]$ has to be computed by $(n+i+1)T_S + T_j$ may not be met for all values of these two parameters, even if there is no constraint on the amount of hardware resources. Further, it is intuitively obvious that if T_j is too large, then latency will get hurt because the computation of output values will wait for the state values to be available. We analyze the effect of these two parameters and the sample period on the feasibility of an LTI system (i.e. existence of a schedule), and in case the system is feasible, we calculate the best latency that can be obtained. These results enable one to answer whether a given pair of constraints on T_S and T_L can be met, and also construct a transformed algorithm or CDFG that will meet the constraints.

Feasibility Theorem: *It is feasible to implement a LTI system with non-trivial coefficient matrices, which has been unfolded i times, has a sample period T_S, and a skew in the arrival of previous state value of T_j, if and only if*

$$P \le \left(\frac{2^{T_j}(2^{T_S}-1)}{2^m}\right)\left(\frac{2^{(i+1)T_S}-2^m R}{2^{(i+1)T_S}-1}\right) \qquad \textbf{(EQ 10)}$$

where P is the number of inputs, R is the number of states in the initial CDFG before unfolding and the minimum latency transformation. The adder delay is 1 and the multiplier delay is $m \ge 1$.

Proof: The problem of feasibility occurs because one has to ensure that $\tilde{S}[n+i]$ can be computed by time $(n+i+1)T_S + T_j$ from $X[n] \ldots X[n+i]$ that arrive at time instances $nT_S \ldots (n+i)T_S$, and $\tilde{S}[n-1]$ that arrives at time $nT_S + T_j$. To show that $\tilde{S}[n+i]$ can be computed by this deadline, it is sufficient to show that a maximally fast schedule, where one starts making use of the data values as soon as they arrive, will succeed.

The update equation for each of the $R + iQ$ elements in the state vector $\tilde{S}$ is a linear combination of the R original elements of the state vector, and P input elements for each of the $i+1$ samples. Each of these linear expressions can be evaluated independently in parallel so that it suffices to consider the computation time of just one of them. The fastest way to evaluate such linear combinations is to multiply each of the variables with the corresponding coefficient as soon as the variable arrives, and simultaneously do maximally parallel pair-wise addition of all terms whose variable-coefficient multiplication is finished. This process will continue until all the input vectors $X[n]\ldots X[n+i]$ and the previous state vector $\tilde{S}[n-1]$ have arrived, multiplied with the corresponding coefficient vectors, and the resulting terms all added. At the beginning there are no terms available to add. At every sample period, $kT_S\ \forall k \in \{n\ldots n+i\}$, an input sample vector with P elements is received, and m time units later the P elements would have been multiplied with coefficients and ready for addition. Also, at time nT_S+T_j an additional R elements will be available (only R out of $R+iQ$ elements of $\tilde{S}[n-1]$ contribute to the state update equations), and become ready for addition m time units later. Let $\alpha(T, T_S, T_j, P, R, k_1, k_2, m)$ denote the number of terms available to be added at time T in the maximally fast schedule described above for evaluating the linear expression corresponding to the case where P elements arrive at each of the time instances $kT_S\ \forall k \in \{k_1\ldots k_2\}, k_1 \le k_2$, and R elements arrive at time T_j. Then:

$$\text{Feasibility that } \tilde{S}[n+i] \text{ be computed by } (n+i+1)T_S + T_j \quad \Leftrightarrow \quad \alpha((i+1)T_S + T_j, T_S, T_j, P, R, 0, i, m) = 1$$

It can be shown that:

$$\alpha(T, T_S, T_j, P, R, k_1, k_2, m) = \left\lceil \frac{(T-m \ge T_j)\,(2^{T_j}R) + (T-m \ge k_1 T_S)\left(\dfrac{2^{\left(1+\min\left(k_2, \left\lfloor \frac{T-m}{T_S} \right\rfloor\right)\right)T_S} - 2^{k_1 T_S}}{2^{T_S}-1}P\right)}{2^{T-m}} \right\rceil \qquad \textbf{(EQ 11)}$$

Using the above expression the feasibility requirement $\alpha((i+1)T_S + T_j, T_S, T_j, P, R, 0, i, m) = 1$ can be reduced to:

$$\left\lceil \frac{((i+1)T_S \ge m)\,(2^{T_j}R) + ((i+1)T_S + T_j \ge m)\left(\dfrac{2^{\left(1+\min\left(i, \left\lfloor \frac{((i+1)T_S+T_j)-m}{T_S} \right\rfloor\right)\right)T_S} - 1}{2^{T_S}-1}P\right)}{2^{((i+1)T_S+T_j)-m}} \right\rceil = 1$$

Note that the logical relations $(i+1)T_S \ge m$ and $(i+1)T_S + T_j \ge m$ are always true because one cannot avoid multiplying with coefficients, and that $\min(i, \lfloor(((i+1)T_S+T_j)-m)/T_S\rfloor) = i$ because $\tilde{S}[n+i]$ cannot be computed until at least a multiplier delay m after the arrival of $X[n+i]$ at time iT_S. The above equality can then be simplified to the desired form of the feasibility condition in the theorem statement.

Observations:

1. If no schedule is feasible for the given set of parameters, one can adopt any of the following remedies:

increase the skew T_j in the arrival of previous state value, increase the sample period T_S, make the multiplier faster i.e. reduce m, or increase the amount of unfolding i. The first two strategies will always work although, as shown later, increasing T_j can result in increased latency, and increasing T_S may not be desirable. The latter two strategies may not always work. Reducing m may not be possible if it is already equal to 1. Increasing the amount of unfolding may be futile because although the right-hand side of the feasibility condition increases with i, it converges to a finite value. In particular, the second term on the right-hand side converges to 1 as $i \to \infty$. Also, it is worth noting that unfolding is an expensive operation in terms of hardware resource requirements.

2. The LTI system is feasible for some finite unfolding factor i if and only if:

$$P < \left(\frac{2^{T_j}(2^{T_S}-1)}{2^m}\right) \qquad \text{or, equivalently} \qquad T_j > m + \left\lfloor \log_2\left(\frac{P}{2^{T_S}-1}\right)\right\rfloor \tag{EQ 12}$$

The equivalence between the statement "LTI system is feasible for some finite unfolding factor i" and the inequality $P < (2^{T_j}(2^{T_S}-1)/2^m)$ follows from the fact that the second factor on the right hand side of (EQ 10) is less than 1 for any finite i, takes its minimum value at $i = 0$, and monotonically converges to 1 as $i \to \infty$. The equivalence between the two inequalities in (EQ 12) follows from simple algebraic manipulation, and the facts that T_j is an integer, and that $x < n \Leftrightarrow \lfloor x \rfloor < n$ for all real x and integer n.

3. If (EQ 12) holds, then for given T_S and T_j the unfolding factor i must satisfy the following for the system to be feasible:

$$i \geq \left\lceil \frac{1}{T_S}\left(\log_2\left(\frac{2^m R 2^{T_j}(2^{T_S}-1) - 2^m P}{2^{T_j}(2^{T_S}-1) - 2^m P}\right)\right)\right\rceil - 1 \tag{EQ 13}$$

(EQ 13) follows directly from (EQ 10) in the statement of the Feasibility Theorem by solving (EQ 10) for i, and making use of the fact that i is an integer.

4. For given sample period T_S, and unfolding factor i, the skew in state arrival T_j must satisfy:

$$T_j \geq \left\lceil \log_2\left(\frac{2^m P(2^{(i+1)T_S}-1)}{(2^{(i+1)T_S}-2^m R)(2^{T_S}-1)}\right)\right\rceil \tag{EQ 14}$$

(EQ 14) follows from (EQ 10) by recasting the inequality in (EQ 10) in terms of T_j, and making use of the fact that T_j is an integer.

Latency Theorem: *If an LTI system with non-trivial coefficient matrices, which has been unfolded i times, has a sample period T_S, and a skew in the arrival of previous state value of T_j, is feasible, then it can achieve a latency of:*

$$T_L = m + \left\lceil \log_2(2^{T_j - m} + P) \right\rceil = T_j + \left\lceil \log_2\left(1 + \frac{P}{2^{T_j - m}}\right) \right\rceil \qquad \text{(EQ 15)}$$

Note that the latency T_L is independent of unfolding i.

Proof: Latency varies from sample to sample, i.e. the time taken for output samples $Y[n] \dots Y[n+i]$ to be calculated since the arrival of the corresponding input sample $X[n] \dots X[n+i]$ is in general different for each of the $i+1$ samples. Let $T_L[k]$ be the latency for the $Y[n+k]$ sample $\forall k = 0 \dots i$. We define the system latency T_L to be $\max(T_L[0], T_L[1], \dots, T_L[i])$.

From (EQ 9) it is clear that $Y[n+k]$ is a linear combination of the $R + iQ$ elements of the previous state vector $\tilde{S}[n-1]$ that arrive at time $nT + T_j$, and P elements in each of the vectors $X[n] \dots X[n+k]$ that arrive at time instants $nT_S \dots (n+k) T_S$. The fastest way to evaluate such a linear combination, using the same strategy as we adopted in the proof of the feasibility theorem, is to multiply each of the variable elements by its corresponding coefficient as soon as it arrives, and simultaneously do maximally parallel pair-wise additions of all terms whose variable-coefficient multiplication is finished. This works because addition is associative and commutative. The process continues until all elements have arrived, multiplied, and been added so that a single term remains. One thing to note from the state-space output equations obtained after unfolding and minimum latency transformation is that of the $R + iQ$ elements of $\tilde{S}[n-1]$, one element has a coefficient of 1, and the others have coefficients 0. So in effect $\tilde{S}[n-1]$ contributes only one variable to each output, and that too with a coefficient of 1 so that no multiplication is needed.

Using such a maximally fast schedule, one can express $T_L[k]$ as below using the function $\alpha(T, T_S, T_j, P, R, k_1, k_2, m)$ that we defined earlier:

$$T_L[k] = \max(kT_S + m, T_j) + \lceil \log_2(\alpha(\max(kT_S + m, T_j), T_S, T_j - m, P, 1, 0, k, m)) \rceil - kT_S$$

and using the expression for $\alpha(T, T_S, T_j, P, R, k_1, k_2, m)$ from (EQ 11), one can simplify the above expression to:

$$T_L[k] = m + \left\lceil \log_2\left(2^{T_j - kT_S - m} + P\left(\frac{2^{(k+1)T_S} - 1}{2^{(k+1)T_S} - 2^{kT_S}}\right)\right) \right\rceil \qquad \text{(EQ 16)}$$

From (EQ 16) together with the feasibility theorem one can show that

$$T_L[k] \geq T_L[k+1] \qquad \forall k \in \{0 \dots i-1\}$$

which means that the latency associated with $Y[n]$ is at least as high as latencies for

$Y[n+1]\ldots Y[n+i]$. From this one immediately gets the following which proves the theorem:

$$T_L = \max(T_L[0], T_L[1], \ldots, T_L[i]) = T_L[0] = m + \lceil \log_2(2^{T_j - m} + P) \rceil = T_j + \left\lceil \log_2\left(1 + \frac{P}{2^{T_j - m}}\right) \right\rceil$$

Observations:

1. Intuitively, from (EQ 15) it follows that one should have small values of T_j in order to have low latency. However, recall from (EQ 14) that a certain minimum value of T_j is needed in order for the system to be feasible. This suggests that one cannot arbitrarily improve the sample period T_S in presence of a constraint on latency T_L.

2. A necessary and sufficient condition for a feasible system to achieve latency $\leq T_L$ is:

$$T_j \leq m + \lfloor \log_2(2^{T_L - m} - P) \rfloor \qquad \text{or, equivalently} \qquad 2^{T_j} \leq 2^{T_L} - 2^m P \tag{EQ 17}$$

The observation follows from (EQ 15).

3. A necessary and sufficient condition for a feasible system to achieve the minimum latency, i.e. $T_L = m + \lceil \log_2(1+P) \rceil$, is:

$$T_j \leq m + \lfloor \log_2(2^{\lceil \log_2(1+P) \rceil} - P) \rfloor \tag{EQ 18}$$

(EQ 18) is obtained by plugging $T_L = m + \lceil \log_2(1+P) \rceil$ into (EQ 17).

4. For a given sample period T_S, the best latency that can be achieved by a feasible system is:

$$T_L = m + \left\lceil \log_2\left(2^{1 + \left\lfloor \log_2\left(\frac{P}{2^{T_S} - 1}\right) \right\rfloor} + P \right) \right\rceil \tag{EQ 19}$$

From (EQ 15) it follows that the best latency is obtained for the smallest T_j such that the system is still feasible for the specified T_S at some finite unfolding factor i. Making use of the fact that T_j is an integer, from (EQ 12) it is obvious that this smallest T_j is given by $T_j = 1 + m + \lfloor \log_2(P/(2^{T_S} - 1)) \rfloor$. (EQ 19) then follows trivially by plugging this value of T_j back into (EQ 15).

5. The minimum sample period for a feasible system that achieves the minimum latency, i.e. $T_L = m + \lceil \log_2(1+P) \rceil$, is:

$$T_S = 1 + \left\lfloor \log_2\left(\frac{2^{\lceil \log_2(1+P) \rceil}}{2^{\lceil \log_2(1+P) \rceil} - P} \right) \right\rfloor \tag{EQ 20}$$

To derive (EQ 20) we first note that the feasibility condition for LTI systems at some finite unfolding factor, as implied by (EQ 12), is equivalent to $T_S > \log_2(1 + 2^{m - T_j} P)$. Since T_S is an integer, it follows that the minimum sample period at a given T_j for a system that is feasible for some finite unfolding factor *is* $T_S = 1 + \lfloor \log_2 (1 + 2^{m - T_j} P) \rfloor$. *Note that this minimum sample period will be the smallest for the largest* T_j. *According to (EQ 18) the largest Tj for a feasible system to achieve* $T_L = m + \lceil \log_2 (1 + P) \rceil$ *is* $T_j = m + \lfloor \log_2 (2^{\lceil \log_2 (1 + P) \rceil} - P) \rfloor$. *(EQ 20) is obtained by plugging this expression for* T_j *into* $T_S = 1 + \lfloor \log_2 (1 + 2^{m - T_j} P) \rfloor$ *and simplifying.*

6. From (EQ 19) and (EQ 20) it follows that for any single input LTI system, such as a large fraction of DSP systems, our method can achieve a latency of $T_L = m + 1$ *for all* $T_S \geq 2$, *and a latency of* $T_L = m + 2$ *at* $T_S = 1$. *These numbers are far superior to the fastest reported results in literature even in the specific case of the popular 5-th order elliptical filter benchmark (* $T_L = m + 3$ *and* $T_S = m + 3$ *reported in [34] for* $m = 2$ *).*

Similarly, for any 2-input LTI system, such as many servo controllers, our method guarantees that a latency of $T_L = m + 2$ *can be achieved for all* $T_S \geq 2$, *and that a latency of* $T_L = m + 3$ *can be achieved at* $T_S = 1$.

7. We would like to emphasize that the results in observations 4 and 5, are independent of R, *the number of states in the CDFG, and therefore the latency and sample period implied by (EQ 19) and (EQ 20) are guaranteed for all LTI CDFGs with* P *inputs. Of course different amount of hardware will be required for different cases, and in the next subsection we present a heuristic algorithm to choose the unfolding factor* i *and the skew in state arrival* T_j *so as to minimize hardware.*

8. Figure 5 summarizes in a graphical form the various latency and sample period combinations that are guaranteed to be achieved by the application of our techniques - the graphs are parameterized by the number of inputs P, because as mentioned in Observation 7, our results depend only on P. It should be noted that our technique ignores the coefficient values so that specific examples may in fact achieve better latency and throughput combinations than are guaranteed by our approach.

6.3.3 Algorithm for Simultaneous Optimization of Latency and Sample Period

We have used the analytic results presented above to construct an algorithm which, if feasible, transforms an arbitrary LTI CDFG to satisfy user specified constraints on latency T_L, and sample period T_S.

Step 1: Transform the given LTI CDFG into state space equations characterized by P, Q, R, A, B, C, and D.

Step 2: From (EQ 4), if

$$T_S \geq \max_{r \in \{1 \ldots R\}} (m + \lceil \log_2 (R + P - \mathrm{nr0}(A, r) - \mathrm{nr0}(B, r) - (\mathrm{nr1}(A, r) + \mathrm{nr1}(B, r)) (1 - 1/2^m)) \rceil) ,$$

and,

$$T_L \geq \max_{r \in \{1 \ldots Q\}} (m + \lceil \log_2 (R + P - \mathrm{nr0}(C, r) - \mathrm{nr0}(D, r) - (\mathrm{nr1}(C, r) + \mathrm{nr1}(D, r)) (1 - 1/2^m)) \rceil)$$

then just the maximally fast computation of the linear expressions will suffice - *STOP*.

Step 3: If, in accordance with (EQ 6), $T_S \geq m + \lceil \log_2 (R + P) \rceil$ and, $T_L \geq m + \lceil \log_2 (1 + P) \rceil$, then apply the minimum latency transformation from (EQ 5) to the state space equations - *STOP*.

Step 4: We need to apply techniques from Section 6.3.2. If $T_L < m + \lceil \log_2 (1 + P) \rceil$, then our

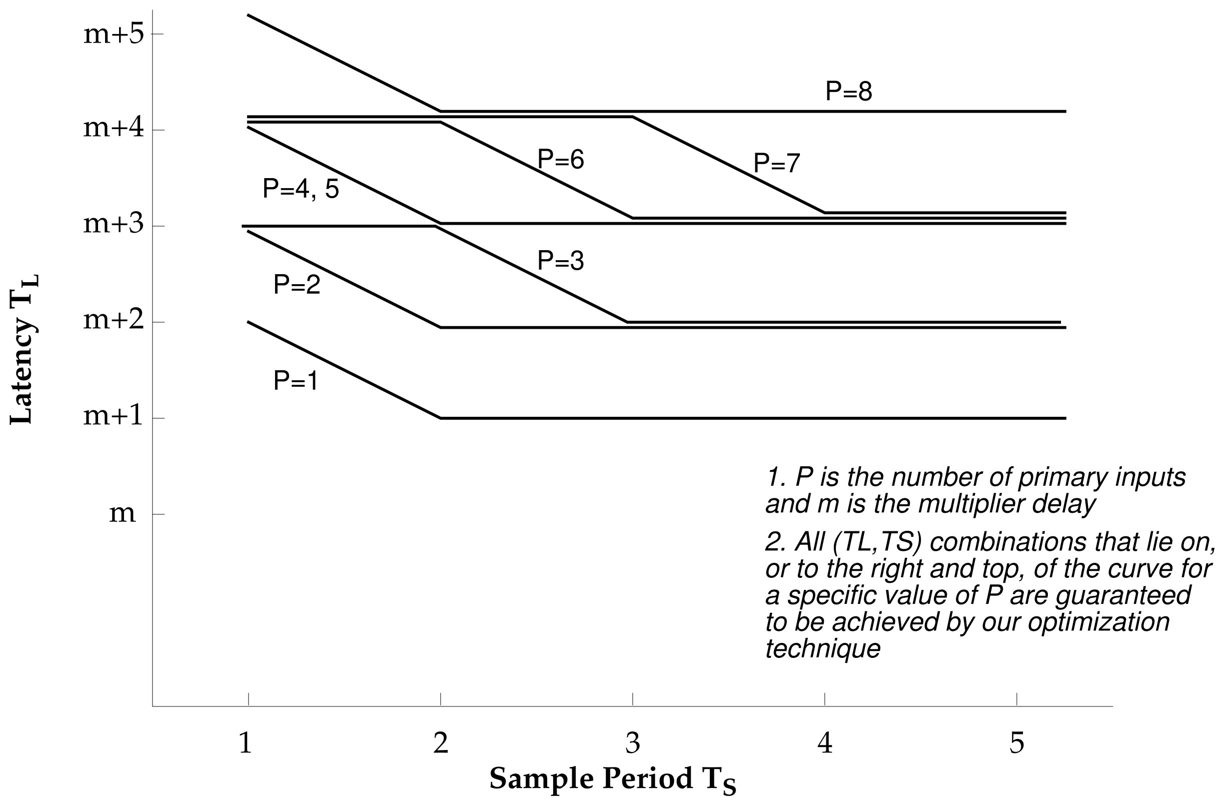


Figure 5: Latency and sample period combinations guaranteed to be achieved by the approach of Section 6.3.

method can never find a feasible system because $m + \lceil \log_2 (1 + P) \rceil$ is the minimum latency that can be achieved for a P input LTI system with arbitrary coefficients - *STOP*.

Step 5: Use (EQ 17) to find the upper bound $T_{j,\,UB}$ on the parameter T_j so that latency is $\le T_L$.

$$T_{j,\,UB} = m + \lfloor \log_2(2^{T_L - m} - P) \rfloor \quad \text{(EQ 21)}$$

Step 6: Use (EQ 12) to find lower bound $T_{j,\,LB}$ on T_j such that the system is feasible for finite i.

$$T_{j,\,LB} = m + 1 + \left\lfloor \log_2\left(\frac{P}{2^{T_S} - 1}\right) \right\rfloor \quad \text{(EQ 22)}$$

Step 7: If $T_{j,\,LB} > T_{j,\,UB}$, we cannot find a feasible system which satisfies the constraints on sample period and latency - *STOP*.

Step 8: Since unfolding is an expensive operation, we would like to keep i as small as possible. Therefore we initially pick the largest allowable T_j, i.e. $T_j = T_{j,\,UB}$, and use (EQ 13) to calculate the minimum unfolding factor i for which a feasible system satisfies the constraints on latency and sample period.

$$i = \left\lceil \frac{1}{T_S}\left(\log_2\left(\frac{2^m R 2^{T_{j,UB}}(2^{T_S}-1) - 2^m P}{2^{T_{j,UB}}(2^{T_S}-1) - 2^m P} \right)\right) \right\rceil - 1 \quad \text{(EQ 23)}$$

As a digression, we would like to note that unfolding is not an expensive operation from the point of view of number of operations - in fact, it can be shown that for LTI systems the average number of operations needed to process a sample is actually smaller with unfolding than without unfolding, unless the unfolding factor is very large. The reason unfolding is an expensive operation is that the number of constant coefficients that are needed grows with increased unfolding, and the resulting coefficient storage requirements often turn out to be very expensive - this, however, is in part depends on the target architecture model that one is using.

Step 9: Use (EQ 14) to find a new lower bound on T_j such that the system is feasible for the above i.

$$T_{j,LB1} = \left\lceil \log_2\left(\frac{2^m P (2^{(i+1)T_S} - 1)}{(2^{(i+1)T_S} - 2^m R)(2^{T_S} - 1)} \right) \right\rceil \quad \text{(EQ 24)}$$

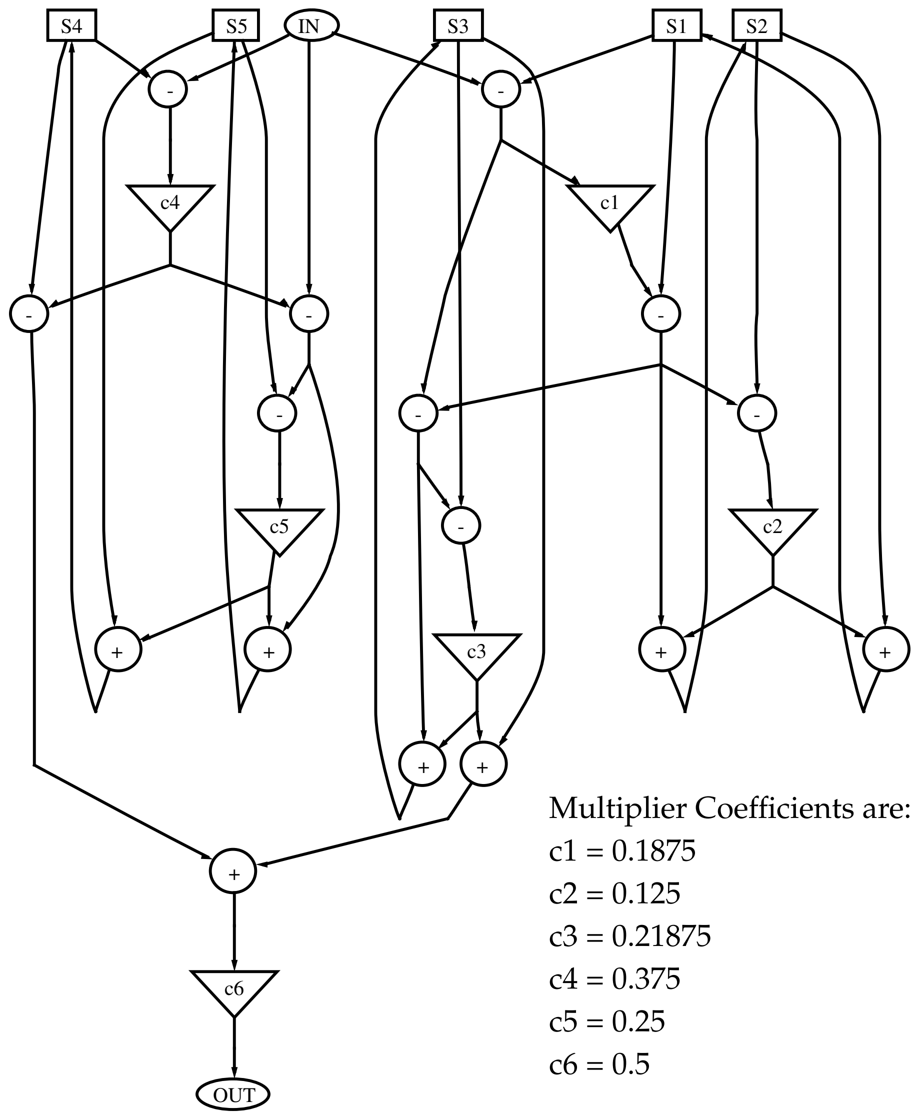
Step 10: Using i from (EQ 23), and a $T_j \in [T_{j,LB1}, T_{j,UB}]$ from (EQ 21) and (EQ 24), generate the CDFG corresponding to the maximally fast schedule as described in Section 6.3.2. This CDFG corresponds to a new algorithm that satisfies the constraints on latency and sample period.

6.4 Latency-Throughput Optimization Technique in Action - An Example

While we will present the results achieved by our techniques on various benchmarks later in the paper, here we illustrate the technique described in the previous section by discussing in detail the application of the technique to a specific example - a 5-th order low pass elliptical wave digital IIR filter. Figure 6 shows the initial CDFG for the example, which has P=1, Q=1, and R=5. Using the definitions in Section 3.1 it can be shown that if hardware was no constraint, this CDFG (without applying any transformations) can achieve the following combinations of latency and sample period: $T_L = 2m + 5$ and $T_S = 3m + 6$.

As mentioned previously, for P=1 the technique for this section guarantees that one can achieve $T_L = m + 1$ for all $T_S \geq 2$, and a latency of $T_L = m + 2$ at $T_S = 1$ - several factors of improvement over latency and throughput achieved by the initial CDFG. We demonstrate the use of the algorithm Section 6.3.3 to transform the CDFG in Figure 6 to achieve $T_L = m + 1$ and $T_S = 2$. All the steps of the algorithm have been completely automated using the software platform described in Section 8.0.

Step 1: The CDFG is converted to the state-space equations:



Multiplier Coefficients are:
c1 = 0.1875
c2 = 0.125
c3 = 0.21875
c4 = 0.375
c5 = 0.25
c6 = 0.5

Figure 6: Initial CDFG for a 5-th Order Low Pass Elliptical Wave Digital Filter with $T_L = 2m + 5$ and $T_S = 3m + 6$

$$S[n] = \begin{bmatrix} \frac{13}{128} & \frac{9}{8} & 0 & 0 & 0 \\ -\frac{91}{128} & \frac{1}{8} & 0 & 0 & 0 \\ -\frac{725}{512} & 0 & \frac{7}{32} & 0 & 0 \\ 0 & 0 & 0 & \frac{3}{32} & \frac{5}{4} \\ 0 & 0 & 0 & -\frac{9}{32} & \frac{1}{4} \end{bmatrix} S[n-1] + \begin{bmatrix} \frac{3}{128} \\ -\frac{21}{128} \\ \frac{325}{512} \\ \frac{5}{32} \\ -\frac{15}{32} \end{bmatrix} X[n]$$

$$Y[n] = \begin{bmatrix} \frac{203}{1024} & 0 & \frac{39}{64} & -\frac{11}{16} & 0 \end{bmatrix} S[n-1] + \begin{bmatrix} \frac{101}{1024} \end{bmatrix} X[n]$$

Step 2: Use (EQ 4) to calculate the latency and sample period that can be achieved by the above state space equations taking into account coefficients with magnitude 0 and 1. We obtain $T_L = m+2$ and $T_S = m+2$ - these values do not meet the requirements, therefore we continue.

Step 3: Next we apply the Minimum Latency transformation of Section 6.2 and check whether the resulting system, shown below, meets the latency and sample period requirements.

$$\tilde{S}[n] = \begin{bmatrix} \frac{13}{128} & \frac{9}{8} & 0 & 0 & 0 & 0 \\ -\frac{91}{128} & \frac{1}{8} & 0 & 0 & 0 & 0 \\ -\frac{725}{512} & 0 & \frac{7}{32} & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{3}{32} & \frac{5}{4} & 0 \\ 0 & 0 & 0 & -\frac{9}{32} & \frac{1}{4} & 0 \\ -\frac{110461}{131072} & \frac{1827}{8192} & \frac{273}{2048} & -\frac{33}{512} & -\frac{55}{64} & 0 \end{bmatrix} \tilde{S}[n-1] + \begin{bmatrix} \frac{3}{128} \\ -\frac{21}{128} \\ \frac{325}{512} \\ \frac{5}{32} \\ -\frac{15}{32} \end{bmatrix} X[n]$$

$$Y[n] = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tilde{S}[n-1] + \begin{bmatrix} \frac{101}{1024} \end{bmatrix} X[n]$$

The above system has $T_L = m+1$ and $T_S = m+3$ - these values do not meet our requirements.

Step 4: Check that the required latency $T_L = m+1$ is greater than or equal to the best latency that the algorithm guarantees $T_L = m + \lceil \log_2(1+P) \rceil$ - since this is indeed the case, we continue.

Step 5: Calculate the upper bound $T_{j,UB}$ on T_j :

$$T_{j,UB} = m + \left\lfloor \log_2(2^{T_L - m} - P) \right\rfloor = m \qquad \textbf{(EQ 25)}$$

Step 6: Calculate the lower bound $T_{j,LB}$ on T_j :

$$T_{j,LB} = m + 1 + \left\lfloor \log_2\left(\frac{P}{2^{T_S} - 1}\right) \right\rfloor = m - 1 \qquad \textbf{(EQ 26)}$$

Step 7: Check that $T_{j,LB} \le T_{j,UB}$ - the condition is satisfied in this example. We continue because the algorithm can find a solution.
Step 8: Calculate the unfolding factor:

$$i = \left\lceil \frac{1}{T_S}\left(\log_2\left(\frac{2^m R 2^{T_{j,UB}}(2^{T_S}-1) - 2^m P}{2^{T_{j,UB}}(2^{T_S}-1) - 2^m P} \right)\right) \right\rceil - 1 = \left\lceil \frac{1}{2}\left(\log_2\left(\frac{2^m 15 - 1}{8} \right)\right) \right\rceil = 1 \qquad \textbf{(EQ 27)}$$

where we had to pick a value for the multiplier delay m, and we assumed $m = 1$. This implies that the final system will achieve $T_L = 2$ and $T_S = 2$.

Step 9: Calculate the new lower bound $T_{j,LB1}$ on T_j:

$$T_{j,LB1} = \left\lceil \log_2\left(\frac{2^m P (2^{(i+1)T_S} - 1)}{(2^{(i+1)T_S} - 2^m R)(2^{T_S} - 1)} \right) \right\rceil = 1 \qquad \textbf{(EQ 28)}$$

Step 10: Unfold the system $i = 1$ times, and then apply the Minimum Latency Transformation of Section 6.2 - the resulting system can achieve the required $T_L = 2$ and $T_S = 2$ (for $m = 1$). Schedule the resulting state space equations, that are show below, using maximally fast computation of the linear expressions and on-arrival processing with $T_j \in [T_{j,LB1}, T_{j,UB}] = [1,1] = 1$, and noting that $X[n]$ arrives at $2n$, $X[n+1]$ arrives at $2n+2$, $\tilde{S}[n-1]$ arrives at $2n+1$, $Y[n]$ is needed by $2n+2$, $Y[n+1]$ is needed by $2n+4$, and $\tilde{S}[n+1]$ is needed by $2n+5$. The results of Section 6.3 guarantee that such a schedule is always feasible.

It is worth noting that the improvement in latency and throughput is at the expense of the number of coefficients increasing from 6 in the original CDFG to 38 in the final equations, and the number of state elements increasing from 5 to 7. In addition, of course, more adders and multipliers will be needed. This extra hardware results in the original latency and sample period improving from $T_L = 2m + 5 = 7$ and $T_S = 3m + 6 = 9$ respectively to $T_L = 2$ and $T_S = 2$.

$$\tilde{S}[n] = \begin{bmatrix} -\frac{12935}{16384} & \frac{261}{1024} & 0 & 0 & 0 & 0 & 0 \\ -\frac{2639}{16384} & -\frac{803}{1024} & 0 & 0 & 0 & 0 & 0 \\ -\frac{29725}{65536} & -\frac{6525}{4096} & \frac{49}{1024} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -\frac{351}{1024} & \frac{55}{128} & 0 & 0 \\ 0 & 0 & 0 & -\frac{99}{1024} & -\frac{37}{128} & 0 & 0 \\ -\frac{7262905}{16777216} & -\frac{964917}{1048576} & \frac{1911}{65536} & \frac{3861}{16384} & -\frac{605}{2048} & 0 & 0 \\ \frac{1221830987}{2147483648} & -\frac{80804817}{134217728} & \frac{13377}{2097152} & \frac{55143}{524288} & \frac{14465}{65536} & 0 & 0 \end{bmatrix} \tilde{S}[n-1] + \begin{bmatrix} -\frac{2985}{16384} \\ -\frac{609}{16384} \\ \frac{6925}{65536} \\ -\frac{585}{1024} \\ -\frac{165}{1024} \\ \frac{7063785}{16777216} \\ \frac{718615077}{2147483648} \end{bmatrix} X[n] + \begin{bmatrix} \frac{3}{128} \\ -\frac{21}{128} \\ \frac{325}{512} \\ \frac{5}{32} \\ -\frac{15}{32} \\ \frac{37229}{131072} \\ \frac{7063785}{16777216} \end{bmatrix} X[n+1]$$

$$Y[n] = \begin{bmatrix}0 & 0 & 0 & 0 & 0 & 1 & 0\end{bmatrix} S[n-1] + \left[\frac{101}{1024}\right] X[n]$$

$$Y[n+1] = \begin{bmatrix}0 & 0 & 0 & 0 & 0 & 0 & 1\end{bmatrix} S[n-1] + \left[\frac{37229}{131072}\right] X[n] + \left[\frac{101}{1024}\right] X[n+1]$$

7.0 Optimizing Latency and Sample Period for the Special Case of Single-Input LTI Systems with Zero Initial State

In the previous section we described a technique to transform a LTI CDFG to satisfy, if feasible, simultaneous constraints on latency and throughput. While on the one hand the technique is completely general in that it produces guaranteed results and is applicable to any LTI CDFG, on the other hand the technique, and the theory behind it, do not make use of the values of the coefficients in the initial state-space equation - the four coefficient matrices A, B, C, and D in (EQ 2) were assumed to be absolutely arbitrary. It is obvious that better results are possible if one were to take advantage of the coefficient values. In particular, coefficients that are 0 need not be considered, and coefficients that have magnitude 1 need not be multiplied. Consequently, it may indeed be possible to achieve combinations of T_S and T_L that the technique in the previous section fails to achieve. More importantly, the knowledge about 0 and 1 coefficients may also be used to obtain more cost efficient implementations - for example, it may be possible to achieve a combination of T_S and T_L with a smaller unfolding factor than is predicted by the algorithm in Section 6.3.3, or it may be possible to use simpler transformations.

Taking advantage of coefficients with values 0 and 1 is, unfortunately, extremely difficult - the mathematical analysis becomes intractable - unless there is some regularity and mathematical structure to the location of these coefficients in the matrices A, B, C, and D. Of course there is no such regularity or structure in the general case of a LTI CDFG. However it is well known in DSP and Linear System Theory that a special case of these LTI CDFGs, namely LTI CDFGs with single-input and zero initial state can always be transformed to certain standard (canonical) CDFG structures [44]. Since throughput and implementation cost (area, number of operations) have been the more popular metrics in traditional DSP, these standard CDFG structures have been developed and analyzed with those two metrics in mind - latency has been overlooked. We noticed that some of these standard forms either have good latency and throughput characteristics at low cost, or are good starting points to apply some of the techniques of the previous section, such as algebraic transformation, unfolding, on-arrival processing, maximally fast computation, and skew between the arrival of the input and the state, to obtain solutions with low latency and high throughput at a low cost.

In this section we describe some techniques that are based on the approach of first converting a single-input single-output LTI CDFG with zero initial state (a single-input multiple-output

CDFG can be treated as a collection of multiple single-input single-output CDFGs) into one of the standard forms, and then applying a fixed sequence of transformations to yield cost efficient solutions with good (often optimal) throughput and latency characteristics. Since a large fraction of applications (e.g. many filters and controllers) are single-input, these techniques can indeed be useful in a variety of designs.

Before describing them, we would like to describe some differences between the special case techniques of this section and the general technique of the previous section. The first crucial difference is that some of the standard forms used by the techniques in this section often do not have good numerical accuracy which may lead to a larger number of bits being required. In contrast, the technique in the previous section by and large preserves the numerical accuracy of the original CDFG. A second important difference is that these special case techniques yield point solutions - a single combination of T_S and T_L - whereas, the technique of the previous section yielded a whole range of optimal combinations of T_S and T_L.

Technique #1: Modified Direct Form II

We begin by describing a technique that is based on the well known standard form: *Direct Form II*. Any single-input single-output LTI CDFG with zero initial state can be transformed to the Direct Form II (shown in Figure 7(a)) using the following steps:

Step 1: Transform the given single-input single-output zero initial state LTI CDFG into state space equations, as in (EQ 2), characterized by P=1, Q=1, R, A, B, C, and D.

Step 2: Taking the Z-Transform of the two state space equations, calculate the *Transfer Function* H(z) = Y(z)/X(z). The zero initial state condition is required for this step.

$$H(z) = \frac{Y(z)}{X(z)} = C(zI-A)^{-1}B + D = \frac{C\,adj(zI-A)B}{det(zI-A)} + D \qquad \textbf{(EQ 29)}$$

It is clear that H(z) will be a ratio of two polynomials in z, and can be expressed as:

$$H(z) = \frac{Y(z)}{X(z)} = \frac{\sum_{i=0}^{N} \alpha_i z^i}{\sum_{i=0}^{N} \beta_i z^i} \qquad \beta_N \neq 0,\ N \leq R \qquad \textbf{(EQ 30)}$$

Step 3: Taking Inverse Z-Transform, the time-domain equations corresponding to (EQ 30) are:

$$Y[n] = \sum_{i=0}^{N} \left(\frac{\alpha_{N-i}}{\beta_N}\right) X[n-i] + \sum_{i=1}^{N} \left(-\frac{\beta_{N-i}}{\beta_N}\right) Y[n-i] \qquad \textbf{(EQ 31)}$$

Comparing (EQ 31) to the CDFG for the Direct Form II structure in Figure 7(a) it follows that the coefficients in the Direct Form II structure can be expressed in terms of the coefficients of the numerator and denominator polynomials of H(z) as below:

$$a_i = -\frac{\beta_{N-i}}{\beta_N} \qquad b_j = \frac{\alpha_{N-j}}{\beta_N} \qquad i \in 1..\mathrm{N}, j \in 0..\mathrm{N} \tag{EQ 32}$$

The Direct Form II structure is widely recognized as a low-throughput, high-latency, high-cost structure, and does not appear to be particularly useful - in the general case of arbitrary coefficients this structure has latency $T_L = 2m + N + 1$ and sample period $T_S = m + N$. However, we found that a simple and fixed sequence of transformation steps can yield a modified structure that has latency $T_L = m + 1$ and sample period $T_S = m + 2$. By way of comparison, the general technique of Section 6.3 can achieve $T_L = m + 1$ for all $T_S \geq 2$, and $T_L = m + 2$ at $T_S = 1$ for the single-input case. The algorithm to modify the Direct Form II structure is as below.

Step 1: If the coefficient b0 in Figure 7(a) is not equal to 0, we first apply a coefficient scaling

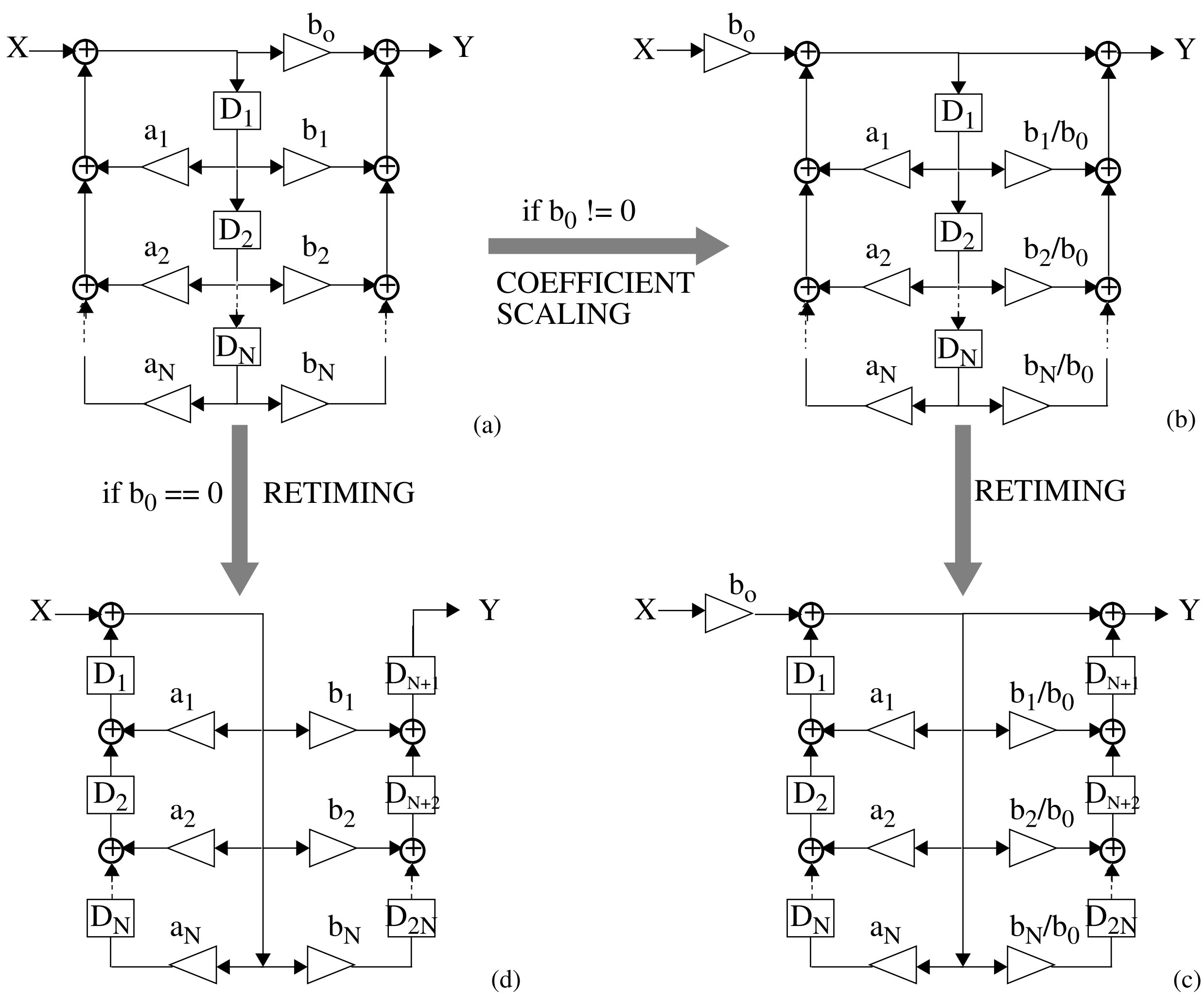


Figure 7: Transforming *Direct Form II* to a Low Latency and High Throughput Structure

transformation to obtain the CDFG in Figure 7(b), and then apply a sequence of retiming steps that move the delay nodes in the middle branch of the CDFG to two sides as in Figure 7(c).

If the coefficient b0 in Figure 7(a) is equal to 0, we remove the corresponding multiplication node and directly apply a sequence of retiming steps to move the delay nodes in the middle branch of the CDFG to the two sides as shown in Figure 7(d).

Step 2: We convert the CDFG obtained in Step 1 into the state space representation. The state space equations for the case $b_0 \neq 0$ are:

$$S[n] = \begin{bmatrix} a_1 & 1 & 0 & \dots & 0 & 0 & 0 & 0 & \dots & 0 \\ a_2 & 0 & 1 & \dots & 0 & 0 & 0 & 0 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ a_N & 0 & 0 & \dots & 0 & 0 & 0 & 0 & \dots & 0 \\ (b_1/b_0) & 0 & 0 & \dots & 0 & 0 & 1 & 0 & \dots & 0 \\ (b_2/b_0) & 0 & 0 & \dots & 0 & 0 & 0 & 1 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ (b_N/b_0) & 0 & 0 & \dots & 0 & 0 & 0 & 0 & \dots & 0 \end{bmatrix} S[n-1] + \begin{bmatrix} b_0 a_1 \\ b_0 a_2 \\ \dots \\ b_0 a_N \\ b_1 \\ b_2 \\ \dots \\ b_N \end{bmatrix} X[n] \tag{EQ 33}$$

$$Y[n] = \begin{bmatrix} 1 & 0 & \dots & 0 & 1 & 0 & \dots & 0 \end{bmatrix} S[n-1] + \begin{bmatrix} b_0 \end{bmatrix} X[n]$$

and, the state space equations for the case $b_0 = 0$ are:

$$S[n] = \begin{bmatrix} a_1 & 1 & 0 & \dots & 0 & 0 & 0 & 0 & \dots & 0 \\ a_2 & 0 & 1 & \dots & 0 & 0 & 0 & 0 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ a_N & 0 & 0 & \dots & 0 & 0 & 0 & 0 & \dots & 0 \\ b_1 & 0 & 0 & \dots & 0 & 0 & 1 & 0 & \dots & 0 \\ b_2 & 0 & 0 & \dots & 0 & 0 & 0 & 1 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ b_N & 0 & 0 & \dots & 0 & 0 & 0 & 0 & \dots & 0 \end{bmatrix} S[n-1] + \begin{bmatrix} a_1 \\ a_2 \\ \dots \\ a_N \\ b_1 \\ b_2 \\ \dots \\ b_N \end{bmatrix} X[n] \tag{EQ 34}$$

$$Y[n] = \begin{bmatrix} 0 & 0 & \dots & 0 & 1 & 0 & \dots & 0 \end{bmatrix} S[n-1] + \begin{bmatrix} 0 \end{bmatrix} X[n]$$

Step 3: Apply maximally fast linear computation transformation [34] - according to (EQ 4) the resulting latency and sample period will be $T_L = m + 1$ and $T_S = m + 2$ for the case $b_0 \neq 0$, and $T_L = 0$ and $T_S = m + 2$ for the case $b_0 = 0$.

Note that most multiplications in the structures in Figure 7(c) (for $b_0 \neq 0$) and in Figure 7(d) (for $b_0 = 0$) are of different constant coefficients with the same variable. By transforming those multiplications to the bit-level, and assuming that word length is W, the multiplications with the various coefficients can all be done using only W shifts regardless of the values of the coefficients. Therefore this structure provides an effective answer to the important problem of minimizing the number of shifts in LTI systems (the number of shifts dominates the implementation cost in bit-serial systems [45]) which has recently received significant attention [45]. Even when the number of states is arbitrarily high, the LTI system can be transformed by our approach such that the number of shifts is a small constant which depends only on the word length requirements. This provides results that are far superior to all previously published approaches. Notice that unlike earlier approaches, this structure does not incur any penalty on either latency or throughput, which, on the contrary, are actually much improved. While simulated annealing based optimization takes between several seconds and few minutes for optimization, optimizing the direct form takes a negligible time. The following table shows the improvement over the initial and the best previously published results on four examples (*mat* - 1 input 3-state controller; *ellip* - 1-input 4-state controller; *lin4* and *lin5* - 1-input 5-state controllers) for 8, 16 and 32 bit designs. The average improvement compared to the initial design is x9.92, and compared to designs optimized using simulated annealing is x7.41, thus clearly demonstrating the importance of coordinated consideration of retiming, algebraic transformations, common sub-expression and multiplication substitution by shift-and-adds.

Design Name	# of Muls		# of Shifts								
			8 bit			16 bit			32 bit		
	I	N	I	SA	N	I	SA	N	I	SA	N
mat	12	6	39	38	6	83	74	14	152	130	25
ellip	20	8	77	56	8	136	87	16	240	198	32
lin4	30	10	92	57	8	212	128	16	383	279	26
lin5	28	10	81	66	8	191	117	16	348	313	26

Table 1: Comparison of Our Modified Direct Form II Structure with Published Results [45]
(I - Initial Design; SA - design optimized using simulated annealing as reported in [45];
N - design based on our new structure)

Technique #2: Modified Direct Form II with One Level of Unfolding + On-Arrival Processing

This is an extension to technique #1 where we unfold once the system obtained in technique #1, and then use maximally fast computation, on-arrival processing, and state arrival skew T_j=0 to get $T_L = m + 1$ and $T_S = m + 1$. Following are the steps corresponding to this technique:

Step 1: Unfold once the modified Direct Form II structure that was obtained in technique 1. We consider only the case $b_0 \neq 0$ to illustrate this technique, and obtain the following state space representation of the once unfolded system using (EQ 31):

$$S[n+1] = \begin{bmatrix} a_1^2 + a_2 & a_1 & 1 & \dots & 0 & 0 & 0 & 0 & \dots & 0 \\ a_1 a_2 + a_3 & a_2 & 0 & \dots & 0 & 0 & 0 & 0 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ a_1 a_N & a_N & 0 & \dots & 0 & 0 & 0 & 0 & \dots & 0 \\ (a_1 b_1 + b_2)/b_0 & b_1/b_0 & 0 & \dots & 0 & 0 & 0 & 1 & \dots & 0 \\ (a_1 b_2 + b_3)/b_0 & b_2/b_0 & 0 & \dots & 0 & 0 & 0 & 0 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ (a_1 b_N)/b_0 & b_N/b_0 & 0 & \dots & 0 & 0 & 0 & 0 & \dots & 0 \end{bmatrix} S[n-1] + \begin{bmatrix} b_0\left(a_1^2 + a_2\right) \\ b_0(a_1 a_2 + a_3) \\ \dots \\ b_0 a_1 a_N \\ a_1 b_1 + b_2 \\ a_2 b_2 + b_3 \\ \dots \\ a_N b_N \end{bmatrix} X[n] + \begin{bmatrix} b_0 a_1 \\ b_0 a_2 \\ \dots \\ b_0 a_N \\ b_1 \\ b_2 \\ \dots \\ b_N \end{bmatrix} X[n+1] \quad \text{(EQ 35)}$$

$$Y[n] = \begin{bmatrix} 1 & 0 & \dots & 0 & 1 & 0 & \dots & 0 \end{bmatrix} S[n-1] + \begin{bmatrix} b_0 \end{bmatrix} X[n]$$

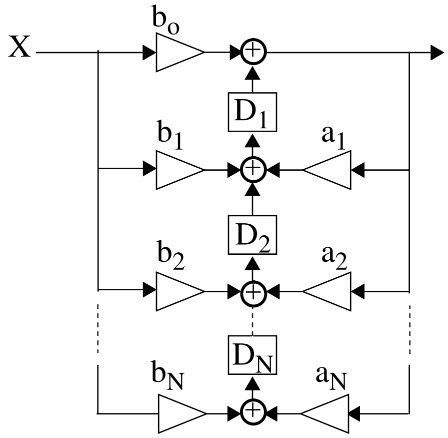
$$Y[n+1] = \begin{bmatrix} (a_1 + b_1/b_0) & 1 & \dots & 0 & 0 & 1 & \dots & 0 \end{bmatrix} S[n-1] + \begin{bmatrix} b_0 a_1 + b_1 \end{bmatrix} X[n] + \begin{bmatrix} b_0 \end{bmatrix} X[n+1]$$

Step 2: Now use a combination of maximally fast linear expression computation and on-arrival processing in a fashion similar to that we employed in Section 6.3. Note that $X[n]$ and $S[n-1]$ arrive at nT_S, $X[n+1]$ arrives at $(n+1)T_S$, $S[n+1]$ must be calculated by $(n+2)T_S$, $Y[n]$ must be calculated by $nT_S + T_L$, and $Y[n+1]$ must be calculated by $(n+1)T_S + T_L$. Using (EQ 31) it is easy to show that the smallest sample period at which this system is feasible are $T_S = m+1$, and that the corresponding latency is $T_L = m+1$ (which, according to Section 6.2, is the minimum latency for a general 1-input system). As a comparison, recall that the general technique of Section 6.3 can achieve $T_L = m+1$ for all $T_S \geq 2$, and $T_L = m+2$ at $T_S = 1$ for the single-input case.

Compared to technique #1, this technique reduces the sample period by 1, achieves the same latency, and has (8N+4) coefficients as opposed to (4N+1) for approximately x2 increase in coefficient memory.

Technique #3: Transposed Direct Form II

This technique is based on the observation that another standard form known as *Transposed Direct Form II* (also known as the *Companion Form*) has good latency throughput characteristics. This form is shown in Figure 8, and has sample period $T_S = m+2$ and latency $T_L = m+1$ (which is equal to the minimum latency for a general single-input system). The coefficient matrices for the corresponding state space representation are also shown in the figure. The coefficients in Figure 8 for *Transposed Direct Form II* are equal to the corresponding coefficients in Figure 7(a) for *Direct Form II*, and can therefore be calculated for any single-input single-output zero initial state LTI CDFG using (EQ 31) under technique #1. The advantage of this structure over that obtained in technique #1 is that there are only N state nodes, as opposed to 2N. Same throughput and latency is obtained as in technique #1 with x2 savings in number of registers



$$A = \begin{bmatrix} a_1 & 1 & 0 & \ldots & 0 \\ a_2 & 0 & 1 & \ldots & 0 \\ \ldots & \ldots & \ldots & \ldots & \ldots \\ a_{N-1} & 0 & 0 & \ldots & 1 \\ a_N & 0 & 0 & \ldots & 0 \end{bmatrix} \qquad B = \begin{bmatrix} b_1 + a_1 b_0 \\ b_2 + a_2 b_0 \\ . \\ b_{N-1} + a_{N-1} b_0 \\ b_N + a_N b_0 \end{bmatrix} \qquad C = \begin{bmatrix} 1 & 0 & \ldots & 0 \end{bmatrix} \quad D = \begin{bmatrix} b_0 \end{bmatrix}$$

Figure 8: The *Transposed Direct Form II* (also called *Companion Form*) with $T_L = m+1$ and $T_S = m+2$

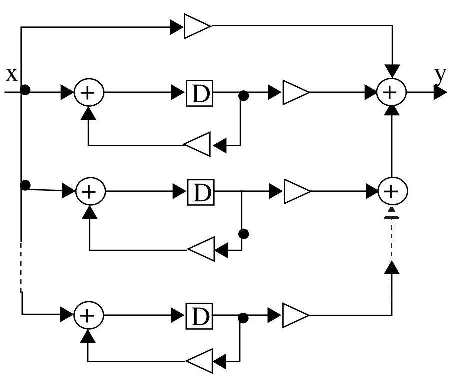
used to store state variables, and only 2N+1 coefficients being required as opposed to 4N+1 resulting in an almost x2 savings in coefficient memory as well.

Technique #4: Transposed Direct Form II with One Level of Unfolding + On-Arrival Processing

Similar to the approach used in technique #2, a latency of $T_L = m+1$ and a sample period of $T_S = m+1$ are obtained if the *Transposed Direct Form II* of Figure 8 is unfolded by 1, and then implemented using on-arrival processing and maximally fast linear computation. We omit the details since they are similar to those of technique #2. However, this technique has only N state nodes and (4N+2) coefficients, for a x2 advantage in both the number of states and the size of coefficient memory over technique #2 for the same latency and sample period.

Other Special Case Techniques for Single-Input Single-Output LTI CDFGs with Zero Initial State

Many single-input single-output LTI systems with zero initial states can be transformed into what is known as the *Diagonal Form*, shown in Figure 9. This form has $T_L = m + \lceil \log_2(R+1) \rceil$ and $T_S = m+1$. More importantly though, the matrix A in the state-space representation is diagonal so that the number of multiplications and additions is substantially reduced in the state update equation. This can be of benefit in implementations where the cost is dominated by



$$A = \begin{bmatrix} s_1 & 0 & \ldots & 0 \\ 0 & s_2 & \ldots & 0 \\ \ldots & \ldots & \ldots & \ldots \\ 0 & 0 & \ldots & s_k \end{bmatrix} \qquad B = \begin{bmatrix} 1 \\ 1 \\ \ldots \\ 1 \end{bmatrix} \qquad C = \begin{bmatrix} r_1 & r_2 & \ldots & r_k \end{bmatrix} \quad D = \begin{bmatrix} b_0 \end{bmatrix}$$

Figure 9: The *Diagonal Form* with $T_L = m + \lceil \log_2(R+1) \rceil$ and $T_S = m+1$

multiplication cost, as is often the case in programmable DSPs and microcontrollers. An even larger class of LTI systems with zero initial state can be transformed into the so called *Jordan Form* where the resulting matrix A has a non-zero diagonal together with some elements that are 1 in the super-diagonal that runs parallel to and just above the main diagonal. This form retains the nice attributes of the diagonal form except that the achievable sample period degrades slightly to $T_S = m + 2$. Unfortunately, the process of finding the diagonal and Jordan forms is often numerically unstable, and caution needs to be exercised in using them.

8.0 Software Platform - HYPER 2.1 + Maple V

The approach presented here imposes a number of demanding requirements of very distinct and different nature on the synthesis process. The requirements include simulation which addresses word-length trade-offs after transformations, manipulation of computations in both the CDFG and the state-space domains using both numeric and symbolic techniques, rapid estimation of quantitative performances of proposed solutions, and a software platform which supports a large number of transformations.

Both modern algebra manipulation systems and state-of-the-art high level synthesis systems are implemented using several hundred thousand lines of code (for example, just the kernel of Mathematica has more than 330,000 lines of code) and still fail to fulfill all the requirements. Therefore developing an integrated system which will provide all the outlined capabilities is not a cost and time effective solution. However, by interfacing computer algebra and high level synthesis tools which are readily available, a simple and effective solution was created by us which satisfies our requirements, required limited software effort, and allows different tasks to be done using tools best suited for them.

We coupled Maple V, a computer algebra system originally from University of Waterloo [12], with HYPER 2.1 [24], a high level synthesis system from University of California, Berkeley. Maple allows symbolic as well as numerical manipulations in a mixed functional and procedural paradigm. Various packages in Maple enable the representation and manipulation of a variety of mathematical structures such as those found in linear algebra, group theory etc. HYPER on the other hand provides the capability to input a design described in the applicative language SILAGE, translate it into a CDFG, and perform high-level synthesis tasks such as module selection, various transformations, scheduling, and allocation. The final result is implemented using a variety of cell libraries and physical design tools.

Figure 10 shows a typical flow of synthesis for joint latency/throughput optimization using the techniques described in this paper through HYPER and Maple. The user describes the algorithm in SILAGE, and HYPER translates it into the CDFG format and does the initial simulation. An interface program then translates the CDFG into Maple equations, and uses a Maple script to

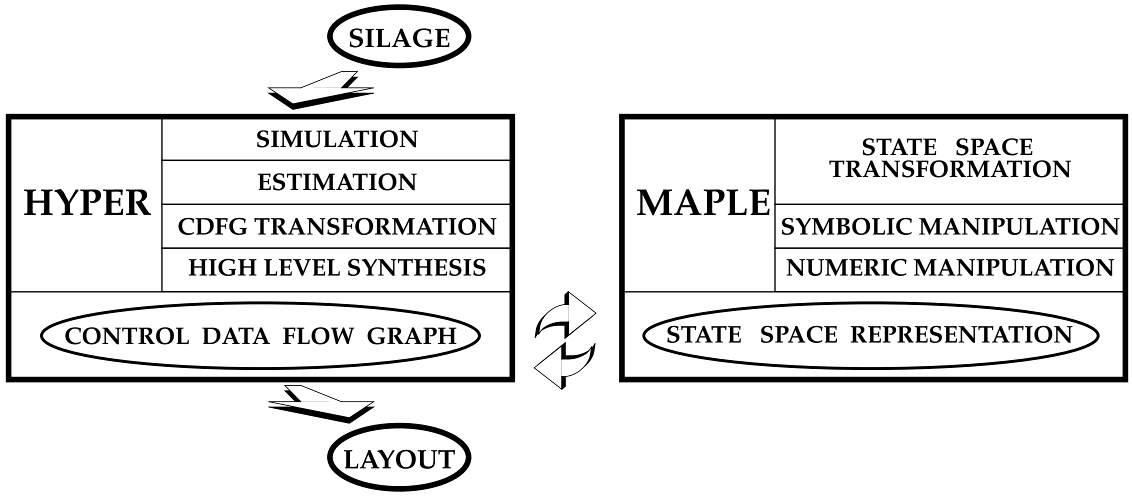


Figure 10: Software Platform Based on Coupling of HYPER 2.1 and Maple V

perform various transformations (such as unfolding, minimum latency transformation, conversion to standard forms) in the state space and to derive a structure which satisfies the constraints on both latency and throughput. The result is then fed back to HYPER, which then estimates the implementation cost and performs bit level simulation. HYPER can also be used for further optimizations. For example multiplications with constants can be replaced by shift-and-adds which often significantly reduces cost of implementation, particularly in the case of linear computations.

9.0 Experimental Results

Using the HYPER+Maple based software platform described in the previous section, we tested the effectiveness of the optimal transformation technique of Section 6.0 and the special case heuristic techniques (for single-input systems) of Section 7.0 on a number of examples. As mentioned in Section 1.3, the latency-throughput transformation techniques can be used for two distinct purposes: to transform a CDFG to meet joint constraints on latency and throughput, and to transform a CDFG so as to improve the cost of implementation at the same latency and throughput as that of the initial CDFG. We tested our techniques in both these modes, and the results are reported later in this section. Since we are not aware of any previous work on algorithm transformations that simultaneously addresses latency and throughput, we are unable to compare our results when using our transformations in the first mode, i.e. to meet constraints on latency and throughput. When using the transformations in the second mode - to improve the cost of implementation for unchanged latency and throughput requirements - we compare the cost of implementing the initial design and the final design using the HYPER synthesis system.

The characteristics of the examples that we tested our techniques on are shown in Table 2. All

Design Name	Description	Characteristics of the Initial CDFG				
		P	Q	R	T_L [cycles]	T_S [cycles]
mat	3-state 1-input linear controller	1	1	3	4	4
ellip	4-state 1-input linear controller	1	1	4	5	5
lin4	5-state 1-input linear controller	1	1	5	6	6
lin5	5-state 1-input linear controller	1	1	5	6	6
iir5 (wdf5)	5th order low pass elliptic wave digital IIR filter	1	1	5	7	9
iir6	6th order lowpass elliptic cascade IIR filter	1	1	6	8	7
iir8	8th order bandpass direct form IIR filter	1	1	8	11	10
iir10	10th order bandstop Butterworth cascade IIR filter	1	1	10	17	15
iir11	11th order low pass Chebyshev cascade IIR filter	1	1	11	19	19
iir12	12th order bandpass Chebyshev cascade IIR filter	1	1	12	20	18
steam	power plant controller	1	1	5	6	6
dist	distillation plant linear controller	2	1	5	5	6
chemical	chemical plant controller	2	1	4	4	6
aircraft	6 knot linear plant controller for VSTOL aircraft	3	3	14	10	10

Table 2: Characteristics of the benchmark examples

the results in this section were obtained using $m = 1$. The area numbers were obtained using the HYPER [24] synthesis system which implements the designs as a custom chip made using a set of communicating word-parallel dedicated datapaths that are controlled by a central finite-state machine controller. HYPER generates the physical layout of the chips by using the LAGER silicon compilation system [46] at the backend - datapaths are generated using the datapath compiler in LAGER, the FSM controller using the logic synthesis and tiling tools in LAGER, the datapath control logic using standard cells, and the overall chip using the macro-cell place-and-route and pad ring generator tools that are part of LAGER. We used a 1.2 micron feature size technology for the examples presented in this paper.

Table 3 presents the results obtained when the transformations described in this paper were

Design Name	Initial Design			Optimum Technique[a]			Heuristic Techniques[b]		
	T_L [cycles]	T_S [cycles]	Area [mm^2]	T_L [cycles]	T_S [cycles]	Area [mm^2]	T_L [cycles]	T_S [cycles]	Area [mm^2]
mat	4	4	14.07	2	2	15.51	2	3	13.93
							2	2	15.59
				3	1	35.29	2	3	11.26
							2	2	10.52
ellip	5	5	52.58	2	2	90.02	2	3	36.33
							2	2	21.65
				3	1	159.77	2	3	65.41
							2	2	36.83

Table 3: Improvements in latency and throughput of the initial design using the heuristic techniques of Section 7.0, and the optimum technique of Section 6.3

Design Name	Initial Design			Optimum Technique[a]			Heuristic Techniques[b]		
	T_L [cycles]	T_S [cycles]	Area [mm^2]	T_L [cycles]	T_S [cycles]	Area [mm^2]	T_L [cycles]	T_S [cycles]	Area [mm^2]
lin4	6	6	67.59	2 3	2 1	183.85 242.97	2 2 2 2	3 2 3 2	50.00 73.14 27.74 44.11
lin5	6	6	223.31	2 3	2 1	614.47 805.25	2 2 2 2	3 2 3 2	162.77 234.42 91.64 141.10
iir5 (wdf5)	7	9	28.92	2 3	2 1	138.16 215.89	2 2 2 2	3 2 3 2	73.34 115.54 40.25 63.11
iir6	8	7	10.90	2 3	2 1	71.83 218.79	2 2 2 2	3 2 3 2	32.33 47.27 22.06 33.23
iir8	11	10	29.25	2 3	2 1	253.82 424.01	2 2 2 2	3 2 3 2	36.74 43.02 24.47 28.86
iir10	17	15	22.69	2 3	2 1	259.80 566.60	2 2 2 2	3 2 3 2	78.21 117.63 49.66 81.09
iir11	19	19	20.58	2 3	2 1	166.94 466.36	2 2 2 2	3 2 3 2	70.15 106.63 45.57 71.15
iir12	20	18	25.73	2 3	2 1	317.02 886.13	2 2 2 2	3 2 3 2	104.13 175.51 69.96 106.28
steam	6	6	82.34	2 3	2 1	184.00 370.30	2 2 2 2	3 2 3 2	35.98 88.10 34.00 54.28
dist	5	6	36.18	3 4	2 1	111.61 225.64	Not Applicable	Not Applicable	Not Applicable
chemical	4	6	30.09	3 4	2 1	101.56 143.36	Not Applicable	Not Applicable	Not Applicable

Table 3: Improvements in latency and throughput of the initial design using the heuristic techniques of Section 7.0, and the optimum technique of Section 6.3

<table>
<tr><th rowspan="2">Design Name</th><th colspan="3">Initial Design</th><th colspan="3">Optimum Technique[a]</th><th colspan="3">Heuristic Techniques[b]</th></tr>
<tr><th>T_L [cycles]</th><th>T_S [cycles]</th><th>Area [mm^2]</th><th>T_L [cycles]</th><th>T_S [cycles]</th><th>Area [mm^2]</th><th>T_L [cycles]</th><th>T_S [cycles]</th><th>Area [mm^2]</th></tr>
<tr><td>aircraft</td><td>10</td><td>10</td><td>25.62</td><td>3
4</td><td>3
1</td><td>61.31
155.40</td><td>Not Applicable</td><td>Not Applicable</td><td>Not Applicable</td></tr>
</table>

Table 3: Improvements in latency and throughput of the initial design using the heuristic techniques of Section 7.0, and the optimum technique of Section 6.3

a. For each example the first set of numbers corresponds to the case when the optimum technique of Section 6.3 is used to achieve a minimum latency system; the second set of number corresponds to the case when it is used to achieve a maximum throughput system.

b. The for sets of numbers for each example correspond to the four special-case heuristic techniques #1, #2, #3, and #4 respectively of Section 7.0

used in the first mode mentioned above - to simultaneously reduce latency and sample period. The table contains the latency, sample period, and chip area for the initial CDFG as given by the user, and the corresponding numbers obtained by the four heuristic techniques of Section 7.0 (only in the case of single-input examples) and the optimum technique of Section 6.3. The optimum technique can give a range of latency and sample period values - the table contains the numbers corresponding to the following two cases: at the minimum latency that can be guaranteed by the algorithm ($m + \lceil \log_2 (1 + P) \rceil$ for a P-input system), and at the minimum sample period that can be guaranteed by the algorithm (1 for all systems). The results show that the techniques are successful at achieving many factors of improvement in latency and throughput, although often at an increased implementation cost. The optimum technique always surpasses the heuristic technique in the latency and throughput that is achieved - in fact the difference would be even more pronounced if the multiplier delay was $m > 1$. Unfortunately, these better latency and throughput characteristics that are achieved by the optimum technique come at a higher implementation cost due to the algorithm unfolding used by the optimum technique. Also, note that the heuristic techniques are not applicable to the three multi-input examples in our benchmark set (*dist*, *chemical*, and *aircraft*).

Table 4 presents the data obtained when the transformation techniques were used in the second mode: to improve the cost of implementation for the same latency and throughput requirements as for the original CDFG. Again, the data is presented for the four special-case heuristic techniques, and for two extreme cases of the optimum technique. The data shows that substantial reduction in the area is obtained in many cases. For example when we compare the initial implementation and the implementations under the same initial timing constraints using technique H3, the area of all benchmark design was reduced. The average and the median reduction in area were by factors 2.93 and 2.26 respectively. When the approach H4 is used, then the average and median improvements were by factors 2.15 and 1.91 respectively. With the optimum technique the results are not consistent - some examples show improvement in area

Design Name	T_L [cycles]	T_S [cycles]	Initial Design Area [mm^2]	Optimum Technique[a] Area [mm^2]		Heuristic Techniques[b] Area [mm^2]			
				O1	O2	H1	H2	H3	H4
mat	4	4	14.07	9.29	11.15	9.62	9.84	5.31	6.47
ellip	5	5	52.58	38.69	37.40	23.79	34.07	15.20	18.37
lin4	6	6	67.59	41.14	42.39	31.55	33.24	16.03	19.53
lin5	6	6	223.31	126.34	118.39	77.35	108.3	28.70	56.68
iir5 (wdf5)	7	9	28.92	28.81	44.62	27.77	34.63	18.16	34.63
iir6	8	7	10.90	21.53	26.29	16.66	20.41	10.90	13.90
iir8	11	10	29.25	21.53	26.29	14.18	20.22	12.95	15.34
iir10	17	15	22.69	39.10	47.75	23.19	32.22	19.36	26.06
iir11	19	19	20.58	43.86	44.44	20.65	28.81	15.12	22.25
iir12	20	18	25.73	57.64	59.41	25.04	46.83	22.39	29.99
steam	6	6	82.34	41.14	46.79	23.79	25.29	14.87	16.37
dist	5	6	36.18	35.22	48.82	-	-	-	-
chemical	4	6	30.09	25.57	28.18	-	-	-	-
aircraft	10	10	25.62	61.31	105.88	-	-	-	-

Table 4: Improvements in area over the initial design using the heuristic techniques of Section 7.0, and the optimum technique of Section 6.3 when the transformed design is scheduled for the same latency and throughput as the original design (using $m = 1$)

a. O1 corresponds to the case when the optimum technique of Section 6.3 is used to achieve a minimum latency system; O2 corresponds to the case when it is used to achieve a maximum throughput system.

b. H1, H2, H3, and H4 correspond to techniques #1, #2, #3, and #4 respectively of Section 7.0.

whereas the others show a degradation. To a large part this is also due to the fact that implementations in HYPER use regular register files to store the constant coefficients too - this makes the unfolding operation used by the optimum technique very expensive.

As the data in Table 3 (where the transformed designs are scheduled under the new, and significantly stricter, latency and throughput constraints) suggests, a large reduction in the latency and sample period of designs is often achieved simultaneously with reduction, or a very limited increase, in the number of operations and the size of final implementation. An intuitive generalization of the results of [28] to the case where both throughput and latency are constrained suggests that the preceding set of conditions is sufficient for power reduction. Table 5 shows the reduction in power between the initial designs and designs obtained using the H4 technique. The power was reduced in all the examples. The highest reduction was for the *lin5* linear controller by factor of 16.8 times, while the smallest improvement was for the *iir6* filter, by 86%. The average and the median reduction in power were by factors 6.15 and 3.58 times. Although the average increase in area was by 88.8%, this number is biased by the sharp increase in areas for a few examples - in more than half the examples both area and power were simultaneously reduced, resulting in an overall median decrease in area by 1.3%.

Design Name	Initial Energy [nJ / sample]	H4 Energy [nJ / sample]	Improvement Factor
mat	59.2	16.8	3.52
ellip	222	45.7	4.86
lin4	288	30.2	9.54
lin5	756	45.0	16.8
iir5 (wdf5)	118	33.0	3.58
iir6	40.2	21.6	1.86
iir8	119	14.1	8.44
iir10	96.7	28.8	3.35
iir11	81.3	27.1	3.00
iir12	89.2	28.3	3.15
steam	377	39.2	9.62

Table 5: Improvements in energy per sample over the initial design using H4, the technique #4 of Section 7.0

For the three multiple input examples the technique H4 is not applicable. However, even in those three cases power can be reduced by using the optimum technique O2. In that case the power reduction for dist, chemical and aircraft controller were by factors 7.36 (from 157 nJ to 21.3 nJ), 9.32 (from 123 nJ to 13.2 nJ), and 3.95 (117 nJ to 29.6 nJ) respectively.

10.0 Conclusion

Meeting simultaneous constraints on throughput and latency while synthesizing from a high-level description is an important unsolved problem. Based on our conviction that algorithm transformations are effective in solving this problem, we first presented a generic technique to optimize latency by using retiming, pipelining, and all algebraic and redundancy manipulation transformations. Building on insights conveniently offered by both state-space and CDFG representation of linear time-invariant systems, we combined this latency minimization technique with unfolding and on-arrival processing to optimally address the latency-sample period product bottleneck for LTI systems. We also presented techniques that transform a large class of linear computations such that not only are the latency and throughput competitive, though sub-optimal, but at the same time a dramatic reduction in power consumption, area of implementation, and the number of operations is obtained. Furthermore, we presented a set of sufficient and necessary conditions for pipelining without degrading latency. On all benchmarked examples the new approaches yielded improvements in latency and throughput when the goal was to improve the latency and throughput characteristics, and in many cases resulted in substantial reductions in chip area when the goal was to reduce the implementation cost for unchanged latency and throughput requirements.

Besides its direct impact in providing superior design solutions for a variety of important applications, our work for the first time explores and exploits the synergy between computer algebra manipulation systems and high level synthesis tools - this connection between the two appears potentially important.

11.0 References

[1] H. D. Man. "Design Technology Research for the Nineties: More of the Same?" In *Euro-DAC '92, European Design Automation Conference*, pages 592–596, 1992.

[2] T. Parks. "Prototyping Real-Time Systems in Ptolemy." Handout at The 1993 ILP Conference at the University of California at Berkeley, 1993.

[3] J. A. Stankovic. "Real-Time Computing Systems: The Next Generation." In J. A. Stankovic and K. Ramamritham, editors, *IEEE Tutorial on Hard Real-Time Systems*, pages 14–38. Computer Society Press of the IEEE, 1988.

[4] J. H. Reif, editor. *Synthesis of Parallel Algorithms*. Morgan Kaufmann Publishers, 1993.

[5] A. G. Akritas. *Elements of Computer Algebra*. John Wiley, 1989.

[6] J. H. Davenport, Y. Siret, and E. Tournier. *Computer Algebra - Systems and Algorithms for Algebraic Computation*. Academic Press, London, UK, 1988.

[7] J. H. Davenport and B. M. Trager. "Scratchpad's View of Algebra I: Basic Commutative Algebra." In A. Miula, editor, *International Symposium on Design and Implementation of Symbolic Computation Systems, Lecture Notes in Computer Science*, pages 40–54. Springer-Verlag, 1990. Lecture Notes in Computer Science, Vol. 429.

[8] B. Simmon. "Four Computer Mathematical Environments." *Notices of the American Mathematical Society*, 37(7):861–868, 1990.

[9] J. K. Foderaro and R. J. Fateman. "Characterization of VAX MACSYMA." In P. S. Wang, editor, *SYMSAC'81: Proceedings of the 1981 ACM Symposium on Symbolic and Algebraic Computation*. ACM, August 1981.

[10] A. C. Hern. *REDUCE User's Manual, Version 3.0*. RAND, Santa Monica, CA, 1983. Publication CP78(4/83).

[11] S. Wolfram. *Mathematica - A System for Doing Mathematics by Computer*. Addison-Wesley Publishing Company, 1988.

[12] B. W. Char, K. O. Geddes, G. H. Gonnet, B. L. Leong, M. B. Managan, and S. M. Watt. *Maple V: The Future of Mathematics*. Springer-Verlag, 1991.

[13] R. D. Jenks and R. S. Sutor. *Axiom: The Scientific Computation System*. Springer-Verlag, 1992.

[14] A. A. Aho, R. Sethi, and J. D. Ullman. *Compilers: Principles, Techniques, and Tools*, chapter 10, pages 585–722. Addison-Wesley Publishing Company, 1986.

[15] C. N. Fischer and J. R. J. LeBlanc. *Crafting a Compiler*. The Benjamin/Cummings Publishing Co. Inc., 1991.

[16] R. E. Crochiere and A. V. Oppenheim. "Analysis of Linear Networks." *Proceedings of the IEEE*, 63(4):581–595, 1975.

[17] R. E. Blahut. *Fast Algorithms for Digital Signal Processing*. Addison-Wesley Publishing Company, 1985.

[18] A. Fetweis, H. Meyr, and L. Thiele. "Algorithm Transformations for Unlimited Parallelism." In *Proceedings of IEEE International Symposium on Circuits and Systems*, pages 1756–1759, 1990.

[19] K. K. Parhi and D. Messerschmitt. "Pipeline Interleaving and Parallelism in Recursive Digital Filters - I: Pipelining using Scattered Look-ahead and Decomposition." *IEEE Transactions on Audio, Speech, and Signal Processing*, 37(7):1099–1117, 1989.

[20] K. K. Parhi and D. G. Messerschmitt. "Pipeline Interleaving and Parallelism in Recursive Digital Filters - II: Pipelined Incremental Block Filtering." *IEEE Transactions on Audio, Speech, and Signal Processing*, 37(7):1118–1134, 1989.

[21] K. K. Parhi. "Algorithm Transformation Technique for Concurrent Processors." *Proceedings of the IEEE*, 77(12):1879–1895, 1989.

[22] D. C. Ku and G. D. Micheli. *High Level Synthesis of ASICs under Timing and Synchronization Constraints*. Kluwer Academic Publishers, 1992.

[23] B. S. Haroun and M. I. Elmasry. "Architectural Synthesis for DSP Silicon Compilers." *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 8(4):431–447, 1989.

[24] J. M. Rabaey, C.-M. Chu, P. D. Hoang, and M. Potkonjak. "Fast Prototyping of Datapath-Intensive Architectures." *IEEE Design & Test of Computers*, 8(2):40–51, June 1991.

[25] H. Trickey. "Flamel: A High-Level Hardware Compiler." *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 6(2):259–269, 1987.

[26] R. A. Walker and D. E. Thomas. "Behavioral Transformation for Algorithmic Level IC Design." *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 8(10):1115–1127, 1989.

[27] R. A. Walker and R. Camposano, editors. *A Survey of High-Level Synthesis Systems*. Kluwer Academic Publishers, 1991.

[28] A. Chandrakasan, M. Potkonjak, J. Rabaey, and R. Brodersen. "HYPER-LP: A System for Power Minimization Using Architectural Transformations." In *Proceedings of IEEE International Conference on Computer-Aided Design*, pages 300–303, 1992.

[29] R. Karri and A. Orailoglu. "Transformation-Based High-Level Synthesis of Fault-Tolerant ASICs." In *Proceedings of ACM/IEEE Design Automation Conference*, pages 662–665, 1992.

[30] N. Park and A. C. Parker. "Sehwa: A Software Package for Synthesis of Pipelines from Behavioral Specifications." *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 7(3):356–370, 1988.

[31] G. Goossens, J. Wandewalle, and H. D. Man. "Loop optimization in register-transfer scheduling for DSP Systems." In *Proceedings of ACM/IEEE Design Automation Conference*,

pages 826–831, 1989.

[32] M. Potkonjak and J.Rabaey. “Optimizing for Resource Utilization Using Transformations.” In *Proceedings of IEEE International Conference on Computer-Aided Design*, pages 88–91, 1991.

[33] P. M. Kogge. *The Architecture of Pipelined Computers*. Hemisphere Publishing Corporation, 1981.

[34] M. Potkonjak and J. Rabaey. “Maximally Fast and Arbitrarily Fast Implementation of Linear Computations.” In *Proceedings of IEEE International Conference on Computer-Aided Design*, pages 304–308, November 1992.

[35] E. A. Lee and D. G. Messerschmitt. “Static Scheduling of Synchronous Data Flow Programs for Digital Signal Processing.” *IEEE Transactions on Computers*, 36(1):24–35, January 1987.

[36] G. Kahn. “The Semantics of a Simple Language for Parallel Programming.” In *Proceedings of the IFIP Information Processing Congress*, pages 471–475. North-Holland, Amsterdam, 1974.

[37] G. Kahn and D. MacQueen. “Coroutines and Networks of Parallel Processes.” In B. Gilchrist, editor, *Proceedings of the IFIP Information Processing Congress*, pages 993–998. North-Holland, Amsterdam, 1977.

[38] D. F. Delchamps. *State-Space and Input-Output Linear Systems*. Springer-Verlag, 1988.

[39] P. R. Gelabert and T. P. Barnwell. “Optimal Automatic Periodic Multiprocessor Scheduler for Fully Specified Flow Graphs.” *IEEE Transactions on Signal Processing*, 41(2):858–888, February 1993.

[40] D. A. Patterson and J. L. Hennessy. *Computer Architecture: A Quantitative Approach*. Morgan Kaufman Publishers, 1989.

[41] M. Potkonjak and J. Rabaey. “Pipelining: Just Another Transformation.” In *Proceedings of the IEEE ASAP-92*, pages 163–175, 1992.

[42] S. Y. Kung. *VLSI Array Processors*. Prentice-Hall, 1988.

[43] R. A. Roberts and C. T. Mullis. *Digital Signal Processing*, chapter 8-9. Addison-Wesley Publishing Company, 1987.

[44] B. Friedland. *Control System Design: An Introduction to State-Space Methods*. McGraw-Hill, Inc., 1986.

[45] A. Chatterjee and R. Roy. “ARTIST: An Architectural Transformation Program for Optimization.” In *Proceedings of ACM/IEEE Design Automation Conference*, pages 343–348, 1993.

[46] R. W. Brodersen, editor. *Anatomy of a Silicon Compiler*. Kluwer Academic Publishers, 1992.