Transforming Linear Systems for Joint Latency and Throughput Optimization

Mani B. Srivastava
AT&T Bell Laboratories
Murray Hill, NJ 07974, USA
mbs@research.att.com

Miodrag Potkonjak
NEC C&C Research Laboratories
Princeton, NJ 08540, USA
miodrag@ccrl.nj.nec.com

Abstract

We present algorithm transformations to simultaneously optimize for throughput and latency for the important case of linear time-invariant DSP systems. Although throughput alone can be arbitrarily improved using previously published techniques, none of them is effective when latency constraints are considered. We have used a state-space based approach which treats various algorithm 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 optimum and constructive in nature, and produces a complete implementation when feasibility conditions are fulfilled. We also present a sub-optimal but hardware efficient heuristic approach. On all benchmarks the new approaches show much superior results than published ones.

1 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, such as lookahead, unfolding, retiming, and algebraic manipulations, have been developed to obtain high-throughput VLSI and software implementations of such computation - in some cases even arbitrarily high throughput, although often at the cost of increased implementation cost and latency.

However, increasingly DSP computation is being used in systems where both throughput and latency are of importance. One example is that of DSP subsystems used in hard real-time embedded systems where the reactive nature of the system imposes hard constraints on throughput as well as latency. 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 the clients.

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.

1.2 Previous Work, and What is New?

The motivation for this work has been provided by the considerable history of work in effectively using algorithm transformations for problems like code optimization in software compilers [Fis91], generation of functionally equivalent but structurally different computation schemes in control systems and DSP filtering [Cro75], improvement in the throughput of DSP algorithms [Goo89, Par89], and for optimization of area, speed, power etc. in VLSI synthesis [Par88,Wal91, Ku92].

We have developed a novel transformation technique that restructures the initial computation algorithm of a *Linear Time-Invariant* (LTI) system to one that has provably optimal latency and throughput. 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 [Par89]. However, this comes at the expense of a proportional degradation in latency. This long-standing *Latency-Sample Period Product 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. In addition, for cost sensitive systems, we present a highly competitive heuristic transformation techniques.

2 CDFG and State-Space Representations

2.1 System Representation by CDFG

We represent an algorithm for a system by a directed control-dataflow graph (CDFG) where nodes represent data operators, data edges represent the flow of data, and control edges represent sequencing and timing constraints. We restrict ourselves to synchronous operators [Lee87], and to single-rate systems. The system state is represented in a CDFG by special delay nodes which are initialized to user specified values, and delay by one sample the stream of data on their input ports. A system is completely represented by a CDFG and the initial values for all the delay operator nodes in the CDFG. Figure 1 shows an example CDFG.

A system is LTI 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 time-invariant linear combinations of input samples and previous state values. Equivalently, all the operators in a LTI CDFG are either addition of two variables, or addition of a

1066-1409/94 $3.00 © 1994 IEEE

variable and a constant, or multiplication of a variable and a constant.

2.2 The Two Metrics of Speed - Throughput and Latency

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.

The definitions of T_S and T_L can be re-cast in terms of CDFG, as shown in Figure 1. Latency T_L is the length of the longest computation path from any primary input or state to the primary output, while sample period T_S is the length of the longest computation path from any primary input or state to any state. Note that throughput and latency are two independent metrics of speed, and a user may specify constraints on both of them as part of the system specification.

2.3 System Representation in State-Space

Any P-input, Q-output, R-state real-valued LTI CDFG with real-valued data can be equivalently expressed by the following analytically powerful discrete-time finite-dimensional state-space system where X is the input vector, S is the state vector, Y is the output vector, and n is the time index. $S[-1] \in \Re^{R\times 1}$, the initial state at time 0, is known.

$$\begin{aligned} S[n] &= AS[n-1] + BX[n] \\ Y[n] &= CS[n-1] + DX[n] \\ n &\in \{0, 1, \ldots\} \qquad X[n] \in \Re^{P\times 1} \\ S[n] &\in \Re^{R\times 1} \qquad Y[n] \in \Re^{Q\times 1} \end{aligned} \qquad \textbf{(EQ 1)}$$

$A \in \Re^{R\times R}, B \in \Re^{R\times P}, C \in \Re^{Q\times R}, D \in \Re^{Q\times P}, S[-1] \in \Re^{R\times 1}$ are constant matrices

3 Latency and Sample Period for LTI Systems

3.1 Latency and Sample Period of a LTI CDFG

From the state-space representation of a LTI CDFG in (EQ 1), it follows that the sample period T_S is decided by the time taken to compute $S[n] = AS[n-1] + BX[n]$, and latency T_L is decided by the time taken to compute $Y[n] = CS[n-1] + DX[n]$. Therefore:

$$T_S = m + \lceil \log_2 (R+P) \rceil, \text{ and } T_L = m + \lceil \log_2 (R+P) \rceil \quad \textbf{(EQ 2)}$$

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

3.2 Transforming a LTI CDFG for Minimum Latency

From (EQ 1) it becomes obvious that no amount of retiming and algebraic transformations can ever change the matrix D. Therefore, 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 remaining with value 0, then the value of T_L is the minimum possible because each output depends on exactly one state variable through a coefficient of 1. Any LTI CDFG can indeed be transformed to such a minimum latency realization by introducing Q redundant state variables denoted by $\tilde{S}$.

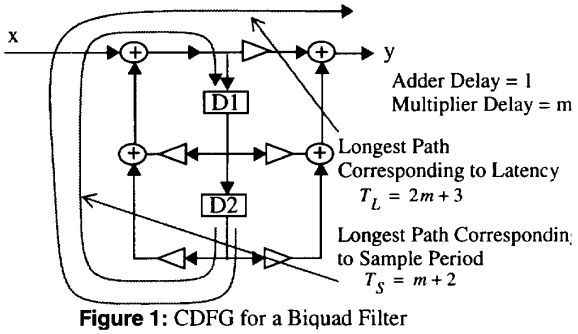

Figure 1: CDFG for a Biquad Filter

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

$$Y[n] = \begin{bmatrix} 0_{QxR} & I_{QxQ} \end{bmatrix} \begin{bmatrix} S[n-1] \\ \tilde{S}[n-1] \end{bmatrix} + DX[n] \qquad \textbf{(EQ 3)}$$

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

This is equivalent to the original system, and achieves the following sample period and latency:

$$T_S = m + \lceil \log_2 (R+P) \rceil, \text{ and } T_L = m + \lceil \log_2 (1+P) \rceil \quad \textbf{(EQ 4)}$$

3.3 Transforming a LTI CDFG to Jointly Improve Latency & Sample Period

Previous research [Par89] 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 latency there is a limit on the best sample period that can be achieved, and this limit depends on the number of primary inputs.

3.3.1 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 amortize the computation cost over several samples. Block processing [Rob87] is one such technique. 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.

$$S[n+i] = A^{i+1}S[n-1] + A^iBX[n] + \ldots + BX[n+i]$$
$$Y[n] = CS[n-1] + DX[n] \qquad \textbf{(EQ 5)}$$
$$\ldots$$
$$Y[n+i] = CA^iS[n-1] + CA^{i-1}BX[n] + \ldots + DX[n+i]$$

Taking buffering delays into account, one obtains:

$$T_S = \frac{m + \lceil \log_2 (R + (i+1)P) \rceil}{i+1}$$
$$T_L = \left(\frac{2i+1}{i+1}\right)(m + \lceil \log_2 (R + (i+1)P) \rceil) \qquad \textbf{(EQ 6)}$$

Note that $\lim_{i \to \infty} T_S = 0$, 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. As a side note, unfolding does not hurt numerical properties - in fact, as shown in [Rob87] there is actually an improvement.

3.3.2 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 3.2 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. The strategy for bringing the unfolded system into a minimum latency form is the same as that 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. One can accomplish this by adding the linear combinations CS, CAS, ..., CA^iS of the original state vector S as redundant states:

[please see lower right corner of this page] **(EQ 7)**

Since we are interested in finding out the limits to which the sample period and latency can be improved simultaneously, the 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 $X[n] \ldots X[n+i]$ are $nT_S \ldots (n+i)T_S$. Computation of 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. 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 2. 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 $-T_S$. For a given T_S, these two parameters - the amount of unfolding i, and the skew T_j in the computation of state vector - together let one 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 with unlimited hardware resources. Further, if T_j is too large, then latency will be hurt because the computation of outputs will wait for the state values to be available. We state the following without proofs:

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

$$P \le \left(\frac{2^{T_j}(2^{T_S} - 1)}{2^m}\right)\left(\frac{2^{(i+1)T_S} - 2^m R}{2^{(i+1)T_S} - 1}\right) \qquad \textbf{(EQ 8)}$$

P = # of inputs, R = original # of states, the adder delay is 1, and the multiplier delay is $m \ge 1$.

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

$$T_L = m + \left\lceil \log_2 (2^{T_j - m} + P) \right\rceil = T_j + \left\lceil \log_2 \left(1 + \frac{P}{2^{T_j - m}}\right) \right\rceil \qquad \textbf{(EQ 9)}$$

Note that T_L is independent of unfolding factor i.

Observations:

1. A LTI system is feasible at finite unfolding factor i iff:

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

or, equivalently $\quad T_j > m + \left\lfloor \log_2 (P/(2^{T_S} - 1)) \right\rfloor$

2. If (EQ 10) holds, then for given T_S and T_j the unfolding factor

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

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

$$\ldots$$

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

i must satisfy the following:

$$i \geq \left\lceil \frac{1}{T_S}\left(\log_2\left(\frac{2^m R 2^{T_j}(2^{T_S}-1)-2^m P}{2^{T_j}(2^{T_S}-1)-2^m P}\right)\right)\right\rceil - 1 \qquad \textbf{(EQ 11)}$$

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

$$T_j \geq \left\lceil \log_2\left(\frac{2^m P(2^{(i+1)T_S}-1)}{2^{(i+1)T_S}-2^m R}\right)\right\rceil \qquad \textbf{(EQ 12)}$$

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

$$T_j \leq m + \left\lfloor \log_2(2^{T_L - m} - P)\right\rfloor \qquad \textbf{(EQ 13)}$$

or, equivalently $\quad 2^{T_j} \leq 2^{T_L} - 2^m P$

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

$$T_L = m + \left\lceil \log_2\left(2^{\left\lfloor \log_2\left(\frac{P}{2^{T_S}-1}\right)\right\rfloor + 1} + P\right)\right\rceil \qquad \textbf{(EQ 14)}$$

Note that (EQ 14) is independent of R, the number of states.

6. Any single-input LTI system can achieve $T_L = m+1$ for all $T_S \geq 2$, and $T_L = m+2$ at $T_S = 1$. These numbers are far superior to the best published results.

3.3.3 Algorithm for Simultaneous Optimization of Latency and Sample Period

The above analytic results are used to construct an algorithm which, if feasible, transforms an arbitrary LTI CDFG to meet constraints on maximum latency T_L, and sample period T_S.

Step 1: If $T_S \geq m + \lceil \log_2(R+P) \rceil$ and, $T_L \geq m + \lceil \log_2(R+P) \rceil$, then by (EQ 2), no transformation is needed - *STOP.*

Step 2: If, from (EQ 4), $T_S \geq m + \lceil \log_2(R+P) \rceil$ and, $T_L \geq m + \lceil \log_2(1+P) \rceil$, then apply the minimum latency transformation in (EQ 3) to the state space equations - *STOP.*

Step 3: If $T_L < m + \lceil \log_2(1+P) \rceil$, then our method can never find a feasible system - *STOP.*

Step 4: Use (EQ 13) to find upper bound $T_{j,UB} = m + \left\lfloor \log_2(2^{T_L - m} - P)\right\rfloor$ on T_j so that latency is $\leq T_L$.

Step 5: Use (EQ 10) to find lower bound $T_{j,LB} = m + 1 + \left\lfloor \log_2(P/(2^{T_S}-1))\right\rfloor$ on T_j such that the system is feasible for finite i .

Step 6: If $T_{j,LB} > T_{j,UB}$, we cannot find a feasible system - *STOP.*

Step 7: Since unfolding is an expensive operation, we initially pick the largest allowable $T_j = T_{j,UB}$, and use (EQ 11) to calculate the minimum unfolding factor i for which a feasible system exists.

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

Step 8: Use (EQ 12) to find a new lower bound on T_j such that the system is feasible for the above i.

$$T_{j,LB1} = \left\lceil \log_2\left(\frac{2^m P(2^{(i+1)T_S}-1)}{2^{(i+1)T_S}-2^m R}\right)\right\rceil \qquad \textbf{(EQ 16)}$$

Step 9: Generate a CDFG using a maximally fast schedule for i and $T_j \in [T_{j,LB1}, T_{j,UB}]$.

3.4 High Throughput, Low Latency, and Low Cost CDFG Structures

For the special case of initially relaxed (zero initial state) LTI systems we have found that a heuristic transformation approach results in structures that are simultaneously competitive, though not optimal, in throughput, latency and implementation cost. The heuristic approach is based on using symbolic manipulation to first transform the zero-initial-state LTI CDFG to one of the well known standard CDFG structures, and then using a transformation script based on retiming and algebraic transformations to transform the standard structure to a structure with high throughput, low latency, and low implementation cost.

As an example of our approach, any single-input single-output LTI CDFG with zero initial state can be transformed to the well known *Direct Form II* structure using symbolic algebra manipulations of the corresponding state-space or Z-transform representations. This structure can be transformed in a surprisingly simple way to a low cost structure with almost optimal latency-throughput trade-off: latency $T_L = m+1$, and sample period $T_S = m+2$. The transformation steps are shown

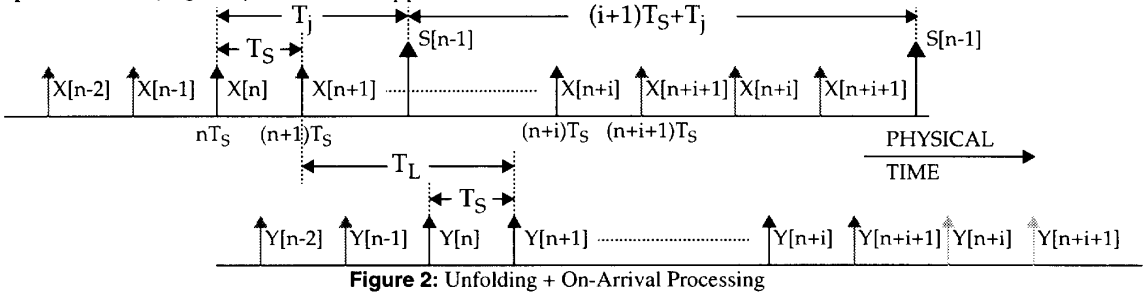

Figure 2: Unfolding + On-Arrival Processing

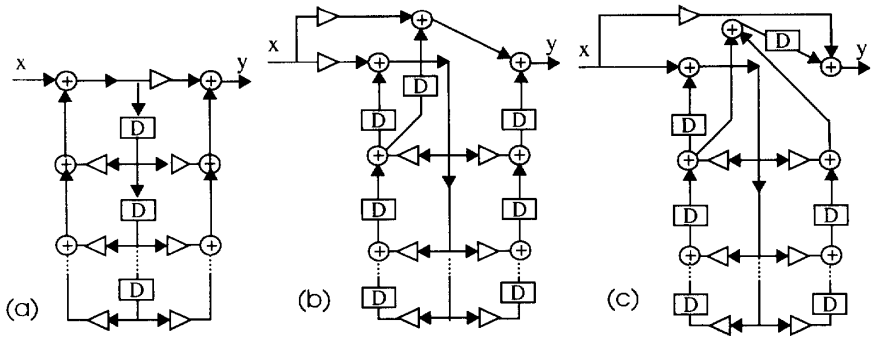

Figure 3: Transforming *Direct Form II* to a Low Latency and High Throughput Structure. The output in (c) depends only on one delay and the input, while each state (delay) depends on at most two states and the input. Therefore, using the method from [Pot92], $T_L = m+1$ and $T_S = m+2$

Design Name	Throughput T_S [cycles]			Latency T_L [cycles]			Area [mm^2]		Power [nJ/sample]	
	Initial	Heuristic	Optimum	Initial	Heuristic	Optimum	Initial	Heuristic	Initial	Heuristic
mat1	4	3	1, ≥2	4	2	3, 2	19.48	3.31	15.9	2.3
ellip	5	3	1, ≥2	5	2	3, 2	18.93	3.75	26.2	3.3
lin5	6	3	1, ≥2	6	2	3, 2	29.58	4.40	53.6	4.5
5WDF[a]	17	4	1, ≥2	16	3	4, 3	10.02	5.81	21.1	4.1
7FWDF	14	3	1, ≥2	12	2	3, 2	16.45	12.89	49.2	6.2

Table 1: Improvements in throughput, latency, area, and power for a set of benchmark examples using the heuristic technique of Section 3.4, and the optimum technique of Section 3.3 [using $m = 1$]

a. Results for 5WDF were obtained using $m = 2$, the value used in the literature for this popular benchmark.

in Figure 3 (a-c), and are successive application of distributivity, constant propagation enabled by associativity, common sub-expression replication, and retiming. The final step is the application of the method presented in [Pot92[on structure shown in Fig 3.c . See [Pot94] for details of this technique.

4 Experimental Results and Conclusion

We tested the effectiveness of our techniques on several benchmark examples. Table 1 compares the key design parameters in the initial and the final designs, obtained using the approach presented in Section 3.4 in terms of throughput, latency, area and power. The area and power values were obtained from the HYPER synthesis system [Rab91] while assuming equal throughput specification for both initial and final designs. The average reductions in sample period and latency were by factors of 2.78 and 3.76 respectively. The area was reduced by a factor of 4.13, while the power was reduced by a factor of 7.97. Table 1 also compares the improvements in latency and throughput obtained over the initial design using the heuristic technique of Section 3.4, and the optimal technique of Section 3.3. The optimum technique always surpasses the heuristic technique by combing an algorithm transformation with unfolding and on-arrival processing to optimally addresses the problem of simultaneous optimization of latency and throughput of a LTI system.The heuristic technique, while being competitive in latency and throughput, has lower implementation costs.

5 References

[Cro75] R.E. Crochiere, A. V. Oppenheim: "Analysis of Linear Networks", Proc. of the IEEE, Vol. 63, No. 4, pp. 581-595, 1975.

[Fis91] C.N. Fischer, R.J. LeBlanc, Jr.: "Crafting a Compiler", The Benjamin/ Cummings Pub. Co. Inc., 1991.

[Goo89] G. Goossens, J. Wandewalle, H. De Man: "Loop optimization in register-transfer scheduling for DSP-systems", DAC, pp. 826-831, 1989.

[Ku92] D. Ku, G.D. Micheli: "Constrained Synthesis and Optimization of Digital Circuits from Behavioral Specifications", Kluwer Academic Publishers, 1992.

[Lee87] E.A. Lee, D.G. Messerschmitt: "Static Scheduling of Synchronous Dataflow Programs for Digital Signal Processing", IEEE Transactions on Computers, Vol. 36, No. 1, pp. 24-35, 1987.

[Par88] N. Park, A.C. Parker: "Sehwa: A Software Package for Synthesis of Pipelines from Behavioral Specifications", IEEE Transaction on CAD for IC, Vol 7, No. 3, pp. 356-370, 1988.

[Par89] K.K. Parhi: "Algorithm transformation technique for concurrent processors", Proceedings of the IEEE. Vol. 77, No. 12, pp. 1879-1895, 1989.

[Pot92] M. Potkonjak, J. Rabaey: "Maximally Fast and Arbitrarily Fast Implementation of Linear Computations, ICCAD, pp. 304-308.

[Pot94] M. Potkonjak, M. B. Srivastava: "Design of High Throughput, Low Latency, and Low Cost Structures for Linear Systems", ICASSP, April 1994.

[Rab91] J. Rabaey, C. Chu, P. Hoang, M. Potkonjak: "Fast Prototyping of Data Path Intensive Architecture", *IEEE Design and Test*, Vol. 8, No. 2, pp. 40-51, 1991.

[Rob87] R.A. Roberts, C.T. Mullis: "Digital Signal Processing", Chapters 8 & 9, Addison-Wesley Pub. Co., 1987.

[Wal91] R.A. Walker, R. Camposano: "A Survey of High-Level Synthesis Systems", Kluwer Academic Pub., 1991.