

# EFFICIENT THROUGHPUT OPTIMIZATION OF FEEDBACK LINEAR COMPUTATIONS USING GENERALIZED HORNER'S SCHEME

*Jan M. Rabaey*

Department of EECS, University of California, Berkeley, CA

*Miodrag Potkonjak*

C&C Research Laboratories, NEC USA, Princeton, NJ

*Kazutoshi Wakabayashi*

C&C Research Laboratories, NEC Corporation, Tokyo, Japan

**ABSTRACT:** We present a generalized Horner's scheme-based approach which enables that a large and important subclass of nonlinear computations, named feedback linear computations, is efficiently, maximally, and arbitrarily sped-up. The new class includes popular nonlinear polynomial Volterra filters and widely used LMS and RLS adaptive filters. The effectiveness and low overhead of the proposed techniques is illustrated on several designs.

## 1. Feedback Linear Computations

Throughput is widely recognized as the single, most important parameter of state-of-the art designs [Bla85, Mit93]. Iteration bound, control and data dependencies impose fundamental limits on achievable performance. Algorithmic transformations are universally accepted as the most efficient and effective way in overcoming these limitations during throughput optimization [Par89, Pot92, Sri94].

It has been shown that an arbitrary linear computation can be sped-up to an arbitrary high throughput using unfolding along time loop, pipelining and a set of algebraic transformations [Pot92]. It has been also shown that there are classes nonlinear computations which can not be transformed so that the throughput is increased [Kun76]. Classification of computation into only two classes, linear and nonlinear, is however a coarse one. We introduce a new class of computations, feedback linear. The motivation behind the introduction of the new class of computations is directly related to our main goal to develop a transformation-based approach which will for as broad as possible a class of computation

enable provably maximally fast and arbitrarily fast implementations.

Informally and intuitively, the class of feedback linear computations encompasses all those computations which do not have multiplications (or divisions) between variables which belong to feedback cycles. More formally, feedback linear computations can be defined as those who can be described using the following set of equations.

$$X[n+1] = A * X[n] + B * (U[n])$$

$$Y[n] = C * (X[n]) + D * (U[n])$$

$X[n]$  is a vector which denotes feedback states (algorithmic delays),  $U[n]$  is a vector of primary inputs, and  $Y[n]$  is a vector representation of primary outputs.  $A$  is a matrix which has as entries functions which do not depend on feedback states  $x[n]$ . There is no restriction of functions which forms the entries of matrices  $B$ ,  $C$  and  $D$ .

The importance of feedback linear computations is emphasized by the fact that many of modern VLSI applications belong to this class. The feedback linear class of computations includes widely used Kalman, LMS adaptive, block LMS, direct RLS adaptive, Cholesky RLS adaptive, QR-RLS adaptive, and nonlinear Volterra polynomial filters. The adaptive filters are regularly used in other high volume applications, such as data and voice cancelers, for equalization of magnetic disk channels, and for equalization of digital wideband packet radio networks [Mit93].

Identification of feedback linear computation is simple and can be done rapidly in polynomial, worst case quadratic, time. The first step is the identification of cycles using the standard depth first search algorithm for labeling of strongly connected components. All nontrivial strongly connected components with more than one node, belong to cycles. All other nodes do not belong to cycles. The second step is a checking that all operations in cycles are additions, subtractions and multiplications with either constants or with variables which do not have transitive fanout from some of feedback delays. During the second step we assume that all inputs are constants. Standard constant propagation algorithms and software can be used for constant propagation [Rab91].

## 2. Horner 's Scheme based Throughput Optimization

We now present the procedure which transforms a feedback linear computation in its maximally fast and arbitrarily fast implementation. The maximally fast implementation refers to the fastest possible implementation when alternation of latency is not allowed, while the arbitrarily fast implementation refers to the implementation where the only goal is throughput optimization with no constraints on the latency.

$$\begin{aligned}
 X_{n+1} &= A^n X_1 + A^{n-1} B U_1 + A^{n-2} B U_2 + \dots + A B U_{n-1} + B U_n \\
 Y_1 &= C X_1 + D U_1 \\
 Y_2 &= C A X_1 + C B U_1 + D U_2 \\
 &\vdots \\
 Y_n &= C A^{n-1} X_1 + C A^{n-2} B U_1 + C A^{n-2} B U_2 + \dots + C B U_{n-1} + D U_n
 \end{aligned}$$

**FIGURE 1.** Arbitrarily Fast Implementation of Linear Computation: Necessary and Sufficient Set of Computations.

We will introduce the transformation technique, for the sake of clarity, on special case of constant feedback linear computation. This subclass of feedback linear computations has matrix A where all entries are constants. The technique is based on the use of maximally fast and arbitrarily fast algorithms for linear computations [Pot92]. The key technical novelty is use of the new generalized Horner's scheme.

**Theorem 1:** *The ratio of the initial and the final AT product of the designs produced using the modified algorithm for arbitrarily fast implementation [Pot92] of constant feedback linear program is constant for an arbitrary high throughput improvement.*

**Proof:**

The proof is constructive. It is based on the novel use of Horner's rule for polynomial evaluation. The rule rearranges the computation of an nth degree polynomial

$$u(x) = u_n x^n + u_{n-1} x^{n-1} + \dots + u_1 x + u_0, \quad u_n$$

to the following form:

$$u(x) = (\dots (u_n x + u_{n-1}) x + \dots) x + u_0$$

Therefore, an arbitrary polynomial can be computed using at most n additions and n multiplications. The excellent exposure of Horner's rule and the number of its generalization are presented by

$$\begin{aligned}
 Y_1 &= C X_1 + D U_1 \\
 Y_2 &= C A X_1 + C (B U_1) + D U_2 \\
 Y_2 &= C A^2 X_1 + C (A (B U_1) + B U_2) + D U_3 \\
 &\vdots \\
 Y_n &= C A^{n-1} X_1 + C (A ( \quad )) + D U_{n-1}
 \end{aligned}$$

**FIGURE 2.** Application of generalized Horner's scheme on the arbitrarily fast implementation of an arbitrary linear computation.

Knuth [Knu81]. Note that in the rest of the paper we treat B, C, and D as matrix operators.

A simple analysis shows that direct implementation of the targeted linear computation after the application of the algorithm presented in [Pot92] causes the resources to grow at a quadratic pace, as the function of the number of unfoldings. We apply the key idea from Horner's scheme, on the part of the computation used to compute the influence of primary inputs shown in Figure 1, so that this overhead is reduced to linear increase. The resulting structure is shown in Figure 2 using the functional dependency form. The part of the pipelined computation which depends only on the primary inputs is shown graphically in Figure 3. Note that we can add to this computational structure as many pipeline delays as requested. Also note that with any additional level of unfolding only the constant amount of computation is added: what is within a constant factor equal to how much more computation is needed if the new speed-up algorithm is not applied. Therefore, the AT product is constant. Q.E.D.

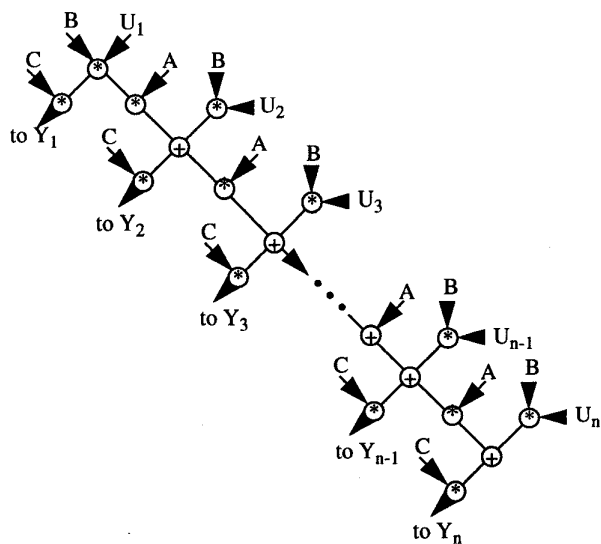


FIGURE 3. The part of feedback linear CDFG on which generalized Horner's scheme is applied. The structure can be pipelined to an arbitrary level.

All theoretical and algorithmic results from the previous theorem can be easily adapted for the general feedback linear system. The only difference is that while initially we treated A as a matrix with constant entries, now they represent matrix operators. It is important to note that the new semantic meaning associated with the A matrix has a limited, but significant impact on hardware overhead. The entries of operator A are now symbolic expressions, and therefore constant propagation cannot be applied in general case. Therefore, each additional application of operator A, induces in the worst case quadratic overhead.

It can be easily shown that the Horner's scheme-based algorithm for arbitrarily speed-up of linear computation results in the following two theorems. The proofs are presented in [Pot94].

**Theorem 2:** *Given an arbitrary feedback linear computation, the fastest implementation of this computation (obtained using only algebraic transformations, common subexpression elimination and replication, constant propagation, and pipelining of primary inputs) uses at least  $\log N + 1$  computational levels and is provided by the generalized Horner's -scheme based algorithm.*

**Theorem 3:** *The ratio of the initial and the final  $AT^2$  product of the designs produced using the algorithm arbitrarily fast implementation of feedback linear program is constant for an arbitrary throughput improvement.*

An interesting observation is that if the level of unfolding is small, when the number of delays in feedback loops is large and the number of inputs and outputs is relatively small in several initial unfoldings, the direct maximally fast computation using functional dependencies shown in Figure 1 may be the more economical alternative. However, when the number of unfoldings is large, the new scheme is superior to the initial, regardless of the number of inputs and states.

### 3. Experimental Results

The Horner's scheme-based approach for optimization of feedback linear DSP computation was applied on four DSP benchmark examples: two nonlinear fixed coefficients Volterra filters and two LMS adaptive filters for achieving maximally fast implementations. The results are shown in Table 1.

Design	ICP	FCP	IFCP	IA	FA	FIA	IFAT
2 Volterra	12	2	6.00	28.86	94.01	3.26	1.84
Volterra	8	2	4.00	27.86	55.21	1.98	2.02
4 LMS	12	4	3.00	55.75	134.97	2.42	1.24
8 LMS	20	5	4.00	111.15	298.03	2.68	1.49

**Table 1:** Fast Implementation of Feedback Linear Programs: ICP - initial critical path; FCP - critical path after the application of the algorithm for fast implementation of linear programs (FLP), IFCP - ratio of critical path before and after application of FLP, IA - initial area; FA - Final area; FIA - ratio of the final and the initial area; IFAT - ratio of the initial and the final AT product. The description of examples is provided in Section 3.

Parameter	In	A1	A2	A3
CP	6	3	1.5	1
Area	26.28	56.27	103.97	317.56
Energy	98.0	207	401	1239
EnergyS	98.0	62.6	54.3	128
CPR	1.00	2.00	4.00	6.00
AreaR	1.00	2.14	3.96	12.1
EnergyR	1.00	2.11	4.09	12.6
EnergySR	1.00	0.64	0.55	1.31

**Table 2:** Application of Arbitrarily Fast Procedure on Volterra Filter: CP - effective critical path for one iteration; Energy - energy at 5 V per sample; Energy - energy at scaled voltage per sample; CPR, AreaR, EnergyR, EnergySR - ratio between initial and final critical path, area, energy at 5 V, and energy at the scaled voltage per sample respectively. The columns represent initial design and designs after application of n (n=1,2,3) times unfolded arbitrarily fast Horner's scheme-based procedure

For this group of examples, the average and median improvement in throughput are 4.25 and 4.00 times,

while the average and median increase in area are 2.58 and 2.50 times, respectively. Table 2 shows experimental results after the application of the Horner's -scheme based algorithm for arbitrarily fast implementation on the Volterra filter for a limited number of unfoldings. As indicated by Theorem 2, if the unfolding process is continued an arbitrary high speed-up is achievable.

### 4. Conclusion

The procedures for transforming feedback linear computations to their maximally (when a limit on latency is imposed) or arbitrarily fast form are presented. Due to use of the generalized Horner's scheme, the hardware efficient implementations are provided.

### 5. References

- [Cha92] A.P. Chandrakasan, et. al.: "Hyper-LP: A Design System for Power Minimization using Architectural Transformations", IEEE ICCAD, pp. 300-303, 1992.
- [Cro75] R.E. Crochiere, A. V. Oppenheim: "Analysis of Linear Networks", Proceeding of the IEEE, Vol. 63, No. 4, pp. 581-595, 1975.
- [Bla85] R.E. Blahut: "Fast Algorithms for Digital Signal Processing", Addison-Wesley, 1985.
- [Kun76] H.T. Kung: "New Algorithms and Lower Bounds for the Parallel Evaluation of Certain Rational Expressions and Recurrences", Journal of the ACM, Vol. 23, No. 2, pp. 252-261, 1976.
- [Lee87] E. A. Lee and D. G. Messerschmitt: "Static Scheduling of Synchronous Dataflow Programs for Digital Signal Processing", IEEE Trans. on Computers, Vol. 36, No. 1, pp. 24-35, 1987.
- [Mit93] S. K. Mitra, J.F. Kaiser: "Handbook for Digital Signal Processing", John Wiley, New York, NY, 1993.
- [Par89] K.K. Parhi: "Algorithm transformation technique for concurrent processors", Proc. of the IEEE. Vol. 77, No. 12, pp. 1879-1895, 1989.
- [Pot94] M. Potkonjak, J. Rabaey, "Maximally Fast and Arbitrarily fast Hardware Efficient Implementation of Linear and Feedback Linear Computations", NEC USA Technical Report, 1994.
- [Sri94] M. B. Srivastava, M. Potkonjak: "Transforming Linear Systems for Joint Latency and Throughput Optimization", EDAC-94 European Design Automation Conference, pp. 267-271, 1994
- [Rab91] J. Rabaey, et. al.: "Fast Prototyping of Data Path Intensive Architecture", IEEE Design and Test, Vol. 8, No. 2, pp. 40-51, 1991.