Synthesis of Configurable Architectures for DSP Algorithms

S. Ramanathan, V. Visvanathan and S. K. Nandy
Supercomputer Education and Research Centre
Indian Institute of Science, Bangalore-560012, INDIA.
{raman, vish, nandy}@serc.iisc.ernet.in

Abstract

ASICs offer the best realization of DSP algorithms in terms of performance, but the cost is prohibitive, especially when the volumes involved are low. However, if the architecture synthesis trajectory for such algorithms is such that the target architecture can be identified as an interconnection of elementary parameterized computational structures, then it is possible to attain a close match, both in terms of performance and power with respect to an ASIC, for any algorithmic parameters of the given algorithm. Such an architecture is weakly programmable (configurable) and can be viewed as an application specific instruction-set processor (ASIP). In this work, we present a methodology to synthesize ASIPs for DSP algorithms.

1 Introduction

Phenomenal advances in silicon micro-chip technology have brought about a paradigm shift in the realization of future systems-on-a-chip. Embedded systems are now being increasingly considered to form part of such systems. Re-targettable, domain specific computation engines are being sought as application specific solutions for a variety of embedded applications that must meet the current day design constraints of ever decreasing time-to-market.

There exist a variety of architectural synthesis techniques [1]-[14], most of which target DSP algorithms onto a predefined hardware configuration in terms of multipliers, adders, MAC, etc. This however results in inferior implementation of DSP algorithms, since the regularity feature of these algorithms are not exploited. Other techniques reported in [15]-[17] rely on software approach to derive the optimal hardware-software combine for a given application. The hardware component in such systems is often conceived as an ASIP for which code is generated using advanced compiling techniques. Very often the solutions offered by such methods do not scale beyond a certain problem size. This is because any software solution is based on a processor view of the hardware, that abstracts the hardware in terms of an instruction set and this imposes an artificial restriction that precludes exploiting all the features of the hardware. By hardware we mean a micro-architecture of a processing element in terms of processing cores, DSP cores, and special functional units. On the other hand if the micro-architecture is defined in terms of elementary parameterized computation structures (subgraphs of the algorithm SFG), it will be possible to fully utilize the capabilities of such a hardware configuration.

Contrary to already cited approaches for synthesis of architectures using hardware-software co-design techniques, there is no partition between hardware and software in the synthesis approach presented in this work. The software counter-part here corresponds to deriving the appropriate control sequence for the computational structures. This approach is a radical shift from conventional schemes, in that, processing elements (computational structures) are re-targetted to realize a given algorithm/application by defining appropriate control sequences. These control sequences can however be determined by developing and using appropriate procedures. The control sequences so determined can be used to reconfigure the hardware.

The advantages of architecture synthesis based on such an approach are:

- processing elements can be defined and synthesized to meet the specific performance requirements of the application; since the processing elements are defined at sufficiently low-level of abstraction, synthesis for low latency and high throughput can be easily achieved,
- processing elements are modular since the synthesis methodology exploits the regularity of the algorithm signal-flow-graph (SFG),
- the synthesized architecture is scalable since control is distributed and implemented in hardware,

- the synthesized architecture is configurable and can be viewed as an ASIP for which the control sequences are set,
- the synthesis methodology can be easily driven to synthesize architectures that meet a pre-specified power budget [18].

It may be appropriate to mention at this point that the architecture synthesis approach proposed in this paper aims to offer domain specific multiprocessor implementations (ASIPs) for DSP algorithms/applications, exploiting both fine-grain and coarse-grain parallelism in the application at the level of a micro-architecture without the aid of a compiler. The advantage of such an approach is that, these ASIPs are comparable to their ASIC counter-part in terms of performance and power, and has the potential to be reconfigured for various algorithmic parameters over the algorithm space.

The rest of the paper is organized as follows. In Section 2, we describe the architecture synthesis methodology for DSP algorithms. The synthesis methodology uses a library of computational structures to synthesize a base architecture for the given algorithm and its specifications. The implementation of this base architecture is an ASIC. In order to support configurability within the domain of the specified algorithm (in otherwords, to derive a computational engine for the given algorithm), the synthesis methodology generates the necessary modifications to the base architecture thus realizing an ASIP for the given algorithm. In Section 3, we exemplify the architecture synthesis methodology through a variety of case studies, which include ASIPs for finite-impulse-response (FIR) filters, FIR lattice filters and fast-fourier-transform (FFT). In Section 4, we conclude this work.

2 Synthesis Methodology

The architecture synthesis methodology proposed here is based on a library of computational structures that forms the set of core computations in the application. These computational structures are parameterized hardware functional units, generic in terms of their functionality. As a result, these computational structures must be fully characterized based on performance constraints imposed by the application. A set of such interconnected computational structures synthesise a base architecture for the application at hand.

Conceptually, it is easy to comprehend that much like a butterfly computational structure is a key computation entity for applications involving FFT, DCT

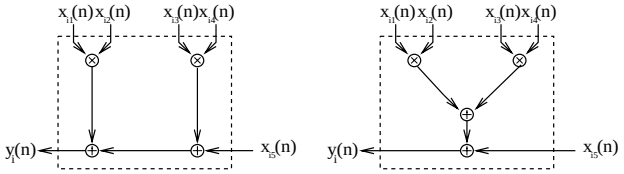

Figure 1: **Equivalent Computational Structures**

etc., other computational structures based on MAC, etc., are essential computational cores necessary to synthesize a wide variety of filter architectures. The choice of a particular structure to others of the same functionality is driven by performance constraints imposed by the overall application. Fig. 1 shows how two computational structures that are functionally equivalent can be derived from one another by applying function preserving transformations (namely, associativity of addition).

The overall synthesis methodology therefore involves the following steps.

1. **Identification of computation structures**: This step is usually algorithm driven. The computation structures chosen correspond to subgraphs of the SFG that match the desired functionalities and are amenable to meet the performance constraints. In case the identified computation structures fails to meet the performance constraints, equivalent structures that meet the performance constraints are derived by applying function preserving transformations to the original structures.
2. **Synthesis of base architecture in terms of chosen computational structures**: Based on instances of computation structures derived earlier, an overall base architecture is synthesized. This architecture is evaluated for the overall algorithm for a chosen technology. This step is iterative, and could result in reviewing step 1 to arrive at a synthesizable architecture.
3. **Synthesis of control and control structures**: In this step the control specifics of the algorithm are re-targetted for the synthesized base architecture. Certain control sequences may translate to additional hardware in this step which renders the base architecture as an ASIP for the given algorithm.

Fig. 2 captures the flow chart of the synthesis methodology to derive an ASIP or computational engine for a given algorithm starting from the algorithm SFG.

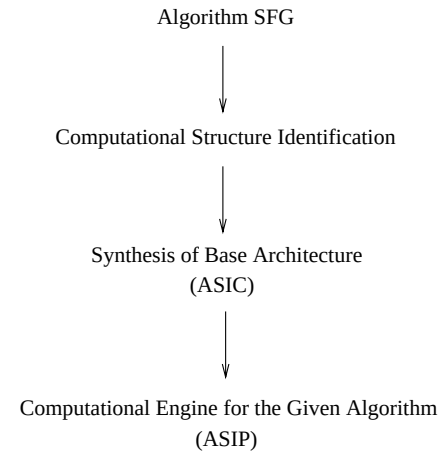

Figure 2: **Flow-Chart of Synthesis Methodology**

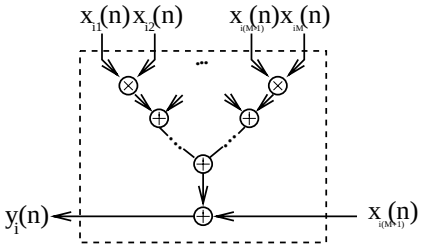

Figure 3: **Type 1 Computation Structure**

We will demonstrate the synthesis technique in the next section with the set of computation structures described below. Exhaustive identification of key computation structures for important DSP algorithms will be the focus of our future work.

Type 1: The computation structure shown in Fig. 3 implements the following expression:

$$y_i(n) = (x_{i1}(n).x_{i2}(n) + x_{i3}(n).x_{i4}(n) + ... + x_{i(M-1)}(n).x_{iM}(n) + x_{i(M+1)}(n)) \quad (1)$$

where, $M \geq 2$ and is a multiple of 2. Note that, some of the multiplier inputs could be a constant, which would be the case if this computation structure is used for fixed filtering implementations. Further, this structure is ideally suited for fixed and adaptive filters, and in its reduced form is the traditional MAC. The following example demonstrates the usefulness of this computation structure.

Example: Consider the SFG of the delayed least-mean-square (DLMS) adaptive filtering algorithm [19]-[21] shown in Fig. 4, wherein, D_A is the adaptation delay necessary for deriving a systolic architecture. This DLMS algorithm has degraded convergence performance and the degradation is progressive with respect to D_A [19]-[21]. Therefore, every effort should be taken to keep the adaptation delay as small as possible. Fig. 5 depicts a systolic array for the DLMS adaptive filter arrived by using associativity of addition followed by retiming on the SFG shown in Fig. 4. On the otherhand, Fig. 6 depicts another systolic array for the DLMS adaptive filter arrived by using two levels of associativity of addition followed by retiming on the SFG shown in Fig. 4. It is clear from Fig. 6 that, Type 1 computation structure with $M = 4$ provides a superior solution since it uses only $\frac{N}{2}$ adaptation delays.This is in contrast to the Fig. 5 implemenation, which uses Type 1 computation structure with $M = 2$, wherein N adaptation delays are used. This clearly illustrates the basic theme, that any restriction on the computation structure to a select few (eg., multiplier, adder, MAC, etc.) can have an adverse effect on the performance. Note that, the coefficient update block is implemented using parallel MAC units (ie., Type 1 computation structure with $M = 2$) □

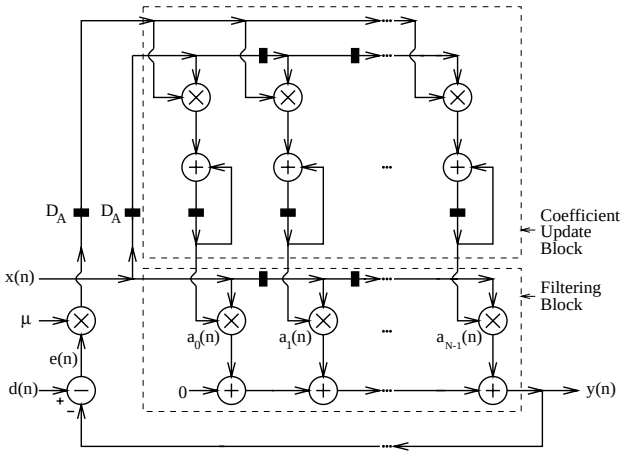

Figure 4: **SFG of the DLMS Algorithm**

Type 2: The computation structure shown in Fig. 7 implements the following expressions:

$$y_{i1}(n) = x_{i1}(n) - K_i.x_{i2}(n) \quad (2)$$
$$y_{i2}(n) = x_{i2}(n) - K_i.x_{i1}(n) \quad (3)$$

This is the key computation structure for the FIR lattice algorithm [22].

Type 3: The computation structure shown in Fig. 8 implements the following expressions:

$$X(j) = K_i.x(j) + x(j) \quad (4)$$
$$X(k) = -K_i.x(k) + x(k) \quad (5)$$

This is the butterfly computation structure employed in the in-place FFT algorithm [22].

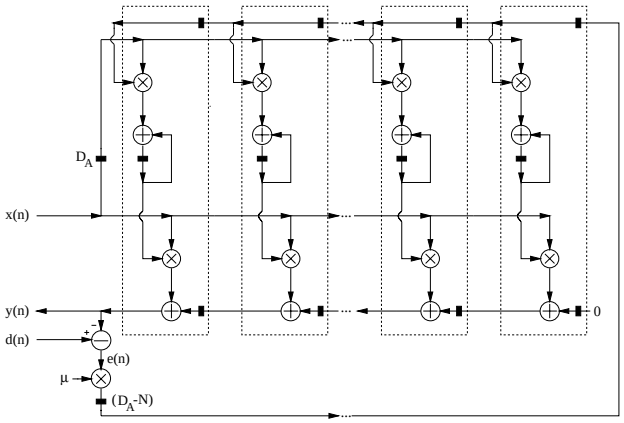

Figure 5: **Systolic array for DLMS adaptive filter ($D_A = N$)**

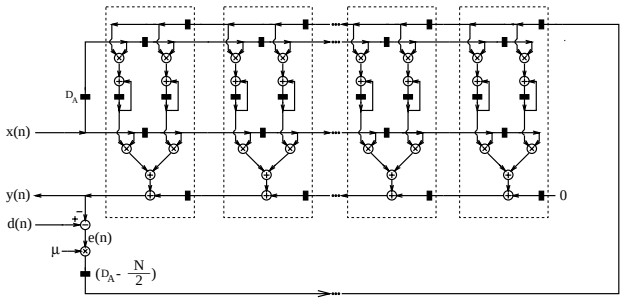

Figure 6: **Systolic array for DLMS adaptive filter ($D_A = \frac{N}{2}$)**

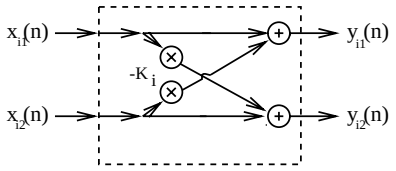

Figure 7: **Type 2 Computation Structure**

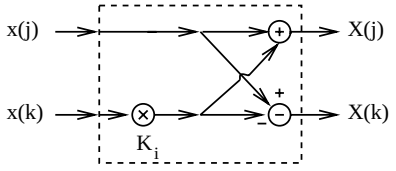

Figure 8: **Type 3 Computation Structure**

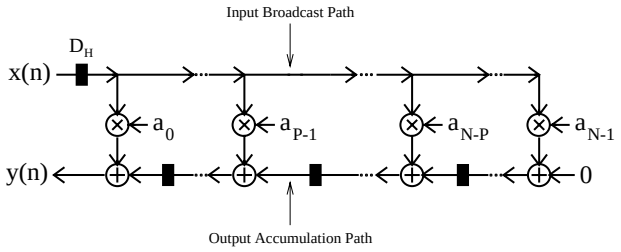

Figure 9: **SFG of N-th order FIR filter**

The use of the computation structures Type 2 and Type 3 for the FIR lattice and FFT implementation respectively will provide a superior area/power/performance solution compared with the use of a multiplier, adder or MAC. In otherwords, subgraphs (corresponding to the computation structures) of an algorithm SFG should provide an ideal platform for the algorithm's implementation. This is primarily due to the fact that, efficient implementation of unique computation structures are possible. Further, regularity of algorithm SFG is not distorted, thus resulting in minimal control overhead.

In the following section, we illustrate the architecture synthesis methodology for a variety of important DSP algorithms like finite-impulse-response (FIR) filters [22], FIR lattice filters and fast-fourier-transform (FFT) [22].

3 Case Study

A) Finite-Impulse-Response Filters

The classical finite-impulse-response (FIR) filter is represented by [22]:

$$y(n) = \sum_{i=0}^{N-1} a_i \; x(n-i) \tag{6}$$

Fig. 9 depicts the SFG representation of an N-th order transpose-form FIR filter with D_H holdup delays at the input. These holdup delays would be defined and used later for pipelining the feedforward path of the final architecture. It is clear from Fig. 9 that, the computation structure Type 1 with $M = 2$ (or in otherwords MAC) would be a good starting choice for realizing a coefficient programmable FIR filter architecture using the proposed synthesis methodology. The task now is to design a coefficient programmable FIR filter architecture (based on the MAC computation structure) with minimal area and power dissipation for an input sampling rate f_s and filter order N.

Let the critical path delay of the final synthesized architecture be denoted by T_{crit}. We now exploit the

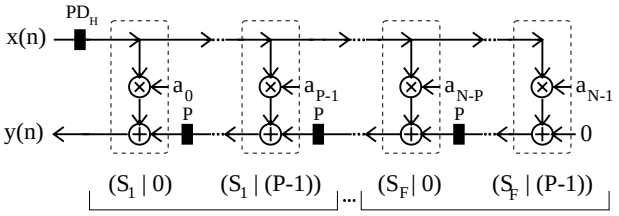

Figure 10: **Slowed-Down FIR Filter**

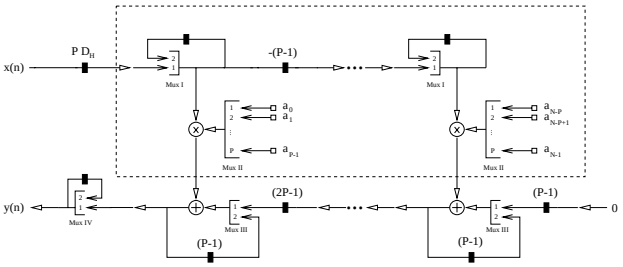

Figure 11: **Folded FIR Filter Architecture**

slowness of the input sampling rate ($f_s = \frac{1}{T_s}$) with respect to the hardware speed ($\frac{1}{T_{crit}}$) to minimize area (as described below using slowdown and folding transformations [2]) by a factor $P = \left\lfloor \frac{T_s}{T_{crit}} \right\rfloor$. The SFG shown in Fig. 9 is slowed-down [2] by a factor P resulting in the structure shown in Fig. 10. The internal clock frequency f_c is then given by $(P * f_s)$. This slowed-down structure is $\frac{100}{P}\%$ hardware efficient for a single-channel implementation. However, 100% hardware efficiency can be achieved by using the folding transformation [2]. Using locally-sequential-globally-parallel (LSGP) mapping [8], P identical computation structures (in this case it is MAC) are mapped onto one physcial computation structure and scheduled in the order indicated in Fig. 10. Note that, any other mapping and scheduling strategy would result in a larger control overhead. Also note that, each sample period consists of P internal clock cycles numbered $0, \cdots, (P-1)$ in that order, and there are F $(= \frac{N}{P}$, for ease of exposition we assume N is a multiple of $P)$ folding sets denoted by $S_1, \cdots, S_F$.

The resulting folded architecture is shown in Fig. 11. The control circuitry consists of 2-to-1 and P-to-1 P-periodic multiplexers and folded arcs with synchronization registers [2]. The negative delays present in the input path of the folded architecture can be neutralized by retiming [23]-[24] appropriate delays from the corresponding output accumulation path. Further, using m $(= \left\lceil \frac{T_m + T_{P-to-1}}{T_a + T_{2-to-1}} \right\rceil$, where T_{P-to-1} and T_{2-to-1} are the delays associated with P-to-1 and 2-to-1 P-periodic multiplexers and T_m and T_a are the delays associated with a multiplier and an adder respectively) holdup registers, the multiplier and the P-to-1 P-periodic multiplexer present in the feedforward paths of the folded architecture are pipelined to obtain a critical path delay (T_{crit}) of $(T_a + T_{2-to-1})$. Note that, the above mentioned retiming and pipelining transformations uses the generalized retiming theorem [24]. With the further use of this theorem and additional holdup registers, register minimization can be achieved.

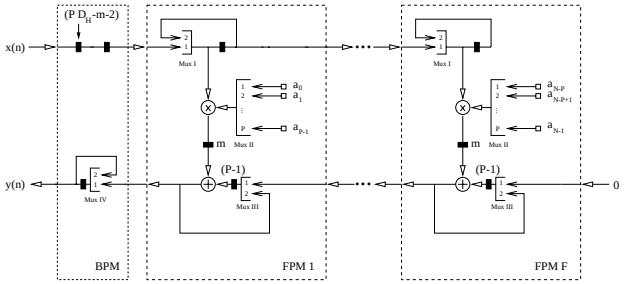

Figure 12: **Final Pipelined FIR Filter Architecture**

Mux #	Selected Input	Clock Cycle # of a Sample Period
I	1	$(-m-1) \bmod P$
	2	Rest
II	i	$(i-m-2) \bmod P$
	$1 \le i \le P$	
III	1	$(P-1)$
	2	Rest
IV	1	$(P-1)$
	2	Rest

Table 1: **Multiplexer Definitions for Figure 12**

The final pipelined architecture shown in Fig. 12 consists of a boundary processor module (BPM) and F identical folded processor modules (FPMs). The multiplexer definitions based on the generalized retiming theorem are given in Table 1. Since, the remaining number of holdup delays present at the input, after pipelining the architecture, should be non-negative $(PD_H - m - 2 \ge 0)$, the holdup delay D_H is given by $\left\lceil \frac{m+2}{P} \right\rceil$. The final base architecture shown in Fig. 12 meets the area and throughput constraints. In order to minimize power dissipation in the base architecture, the supply voltage is chosen such that the critical path delay is $\frac{1}{f_c}$. This can be achieved through an on-chip DC-DC converter. The implementation of this base architecture will provide an ASIC for the given FIR filtering algorithm.

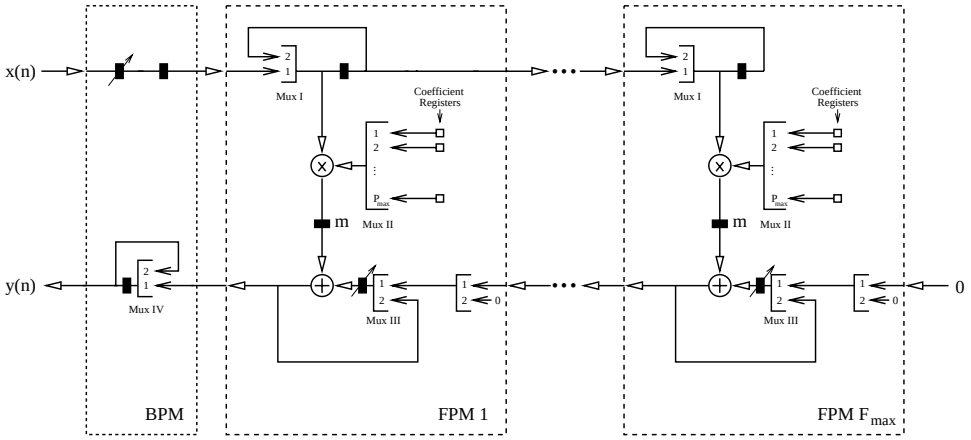

Figure 13: **An ASIP for FIR Filters**

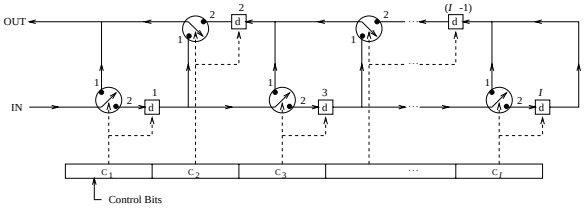

Figure 14: **Programmable Delay Line**

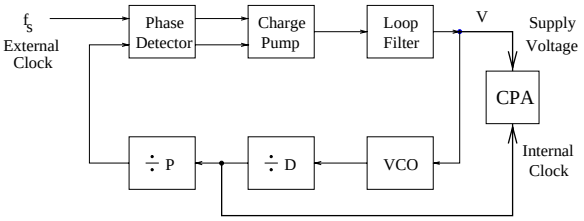

Figure 15: **Programmable clock/supply-voltage**

In order to derive an ASIP for the given algorithm (or in otherwords, to derive a computational engine for the given algorithm), the following components in the base architecture shown in Fig. 12 needs to be made configurable:

- the number of FPMs, the delay lines, the control for multiplexers, and
- the clock and the supply voltage.

Programmability of the number of FPMs can be easily achieved by introducing 2-to-1 multiplexers between adjacent FPMs as indicated in Fig. 13. These multiplexers can be programmed to either connect or disconnect adjacent FPMs. Introduction of these 2-to-1 multiplexers will increase T_{crit} from $(T_a + T_{2-to-1})$ to $(T_a + 2T_{2-to-1})$. Note that F_{max} indicates the maximum number of FPMs, and the number of used FPMs (F) for the given algorithm specifications (N and T_s) is given by $F = \lceil \frac{N}{P} \rceil$ (where, $P = \lfloor \frac{T_s}{T_{crit}} \rfloor$). The unused FPMs in the ASIP are powered down to minimize power dissipation [25]. In each FPM, the maximum number of folded multiply-add computations is P_{max}, resulting in the use of programmable P_{max}-to-1 periodic multiplexer and P_{max} coefficient registers.

The delay lines in the base architecture should be made programmable. A delay of 0 to $\mathcal{I}$ number of registers can be obtained by a programmable delay line (Fig. 14, illustrated for odd $\mathcal{I}$) consisting of a regular pattern of multiplexers, demultiplexers and registers. The multiplexers (demultiplexers) select input 2 (output 2) for a '1' present in the corresponding control-bit C_i. Also, a '1' in the control-bit enables the clock to the corresponding register, while, a '0' disables the clock and tristates the output of these registers. This ensures that the unused registers consume no power and are decoupled from the rest of the circuit.

All the multiplexers in the base architecture, except those used for connecting and disconnecting the FPMs, are P-periodic P-to-1 or 2-to-1 multiplexers. Since P is programmable ($1 \leq P \leq P_{max}$), the control for these multiplexers should also be programmable. This is a straightforward design problem.

Both, the programmable clock $\left(T_c = \frac{T_s}{P}\right)$ and the programmable power supply, can be achieved by a phase-locked-loop (PLL) based design [26]-[27], wherein a clock at sampling rate f_s is the input to the PLL (refer Fig. 15) and the frequency divider $(\div P)$ is programmable. Further, the power supply required to meet the throughput is achieved by using a ring oscil-

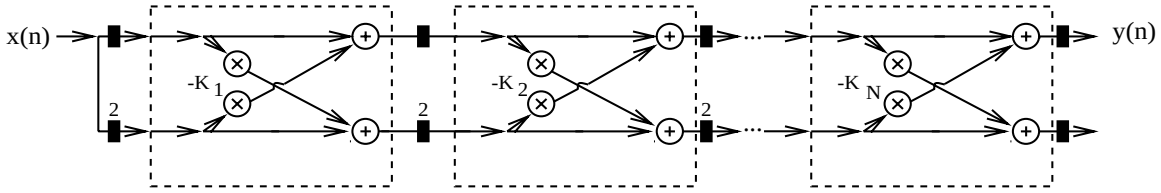

Figure 16: **SFG of an FIR Lattice Filter**

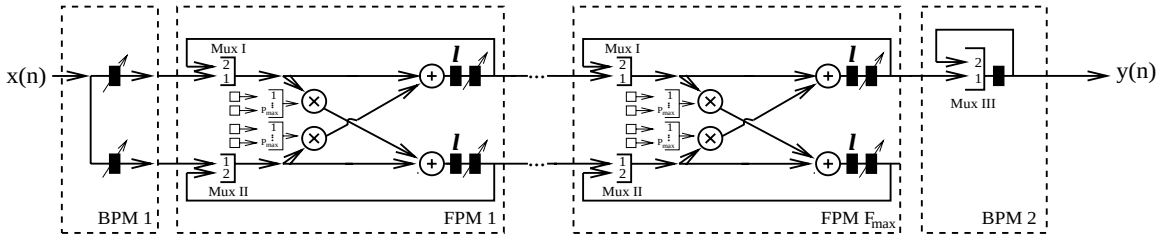

Figure 17: **An ASIP for FIR Lattice Filters**

lator as the voltage controlled oscillator in the design of the PLL. The period of the ring oscillator is such that, the ratio of the delay through the critical path of the ASIP to the period of the ring oscillator is the constant D [26]-[27].

B) FIR Lattice Filters

Fig. 16 shows the SFG of an N-th order FIR lattice filter [22] with a latency of $(N+1)$. Using Type 2 computation structure and following the synthesis procedure described earlier, we arrive at the ASIP for FIR lattice filters as shown in Fig. 17. Note that, l is the number of pipeline registers necessary to pipeline the computation structure. It is interesting to observe that, there is an inherent trade-off between throughput and latency due to uni-directional flow of data from the input to the output. Note that, it is straightforward to obtain the multiplexer defintions for the various multiplexers used in the ASIP.

C) Fast-Fourier-Transform

Fig. 18 shows the SFG of the decimation-in-time 8-point fast-fourier-transform (FFT) algorithm [22] with a latency of 3. Using Type 3 computation structure and following the synthesis procedure described earlier, we arrive at the ASIP for FFT as shown in Fig. 19. Note that, l is the number of pipeline registers necessary to pipeline the computation structure. It is interesting to observe that, there is an inherent trade-off between throughput and latency due to uni-directional flow of data from the input to the outputs. Note that, this ASIP can handle $\lfloor \frac{P}{3} \rfloor$ independent input channels, since only three identical computation structures are mapped onto a physical computation structure.

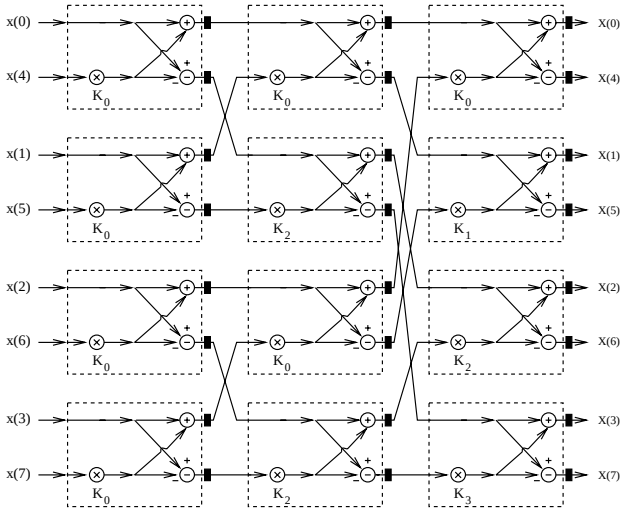

Figure 18: **SFG of an 8-point FFT**

4 Conclusion

We in this work have presented a synthesis methodology to derive configurable architectures (ASIPs) for DSP algorithms. The ASIPs are derived through a synthesis trajectory, wherein the target architecture is identified as an interconnection of elementary parameterized computational structures. These ASIPs are comparable to their ASIC counter-parts in terms of performance and power dissipation, and has the potential to be reconfigured for various algorithmic parameters over the algorithm space. The synthesis methodology was illustrated through a variety of important DSP algorithms.

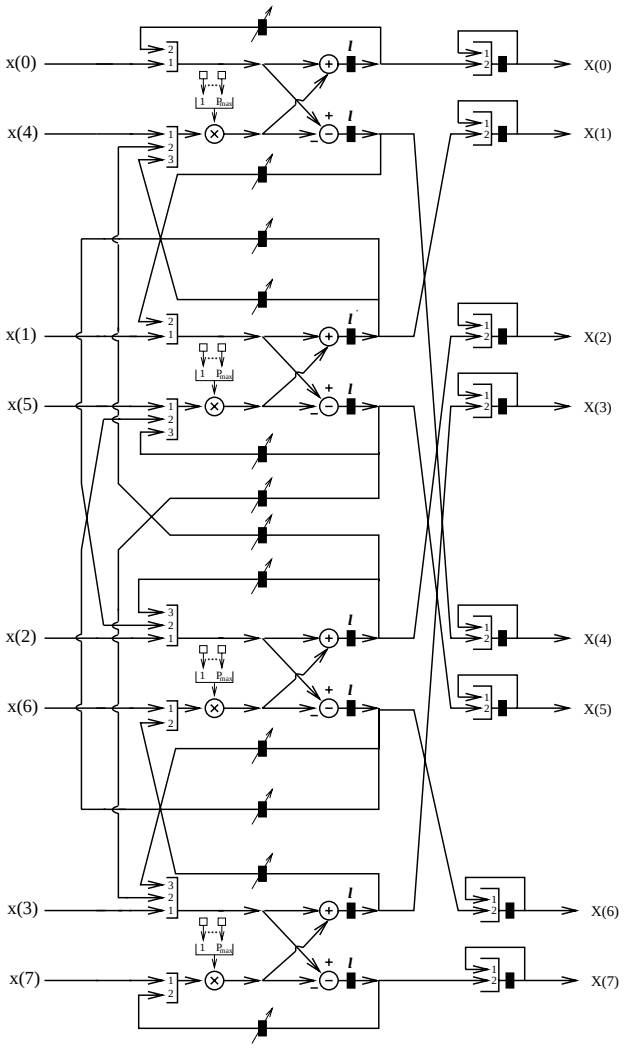

Figure 19: **An ASIP for 8-point FFT**

References

[1] K. K. Parhi, "High-level algorithm and architecture transformations for DSP synthesis," *J. VLSI Signal Processing*, 9(1):121-143, Jan. 1995.

[2] K. K. Parhi, C.-Y. Wang, and A. P. Brown, "Synthesis of control circuits in folded pipelined DSP architectures," *IEEE JSSC*, 27(1):29-43, Jan. 1992.

[3] P. DeWilde et.al., "Parallel and pipelined VLSI implementation of signal processing algorithms," *VLSI and Modern Signal Processing*, Kung, Whitehouse, and Kailath, Eds., Englewood Cliffs, NJ: Prentice-Hall, 1985.

[4] H. J. De Man et.al., "Cathedral II: a silicon compiler for digital signal processing," *IEEE Design and Test*, 3(6):12-25, Dec. 1986.

[5] N. Park and A. C. Parker, "Sehwa: a software package for the synthesis of pipelines from behavioral specifications," *IEEE TCAD*, 7(3):356-370, Mar. 1988.

[6] K. S. Hwang et.al., "Scheduling and hardware sharing in pipelined data paths," *Proc. ICCAD*, 1989.

[7] M. C. McFarland, A. C. Parker, and R. Camposano, "The high-level synthesis of digital systems," *Proc. IEEE*, 78(2):301-318, Feb. 1990.

[8] S. Y. Kung, *VLSI Array Processors*, Englewood Cliffs, NJ: Prentice-Hall, 1988.

[9] B. S. Haroun and M. I. Elmasry, "Architectural synthesis of DSP silicon compiler," *IEEE TCAD*, 8(4):431-447, Jun. 1990.

[10] D. Lanneer et.al., "Architectural synthesis for medium and high throughput signal processing with the new CATHEDRAL environment," *High-Level Synthesis*, R. Camposano and W. Wolf, Eds., Norwell, MA: Kluwer Academic, 1991.

[11] C. H. Gebotys and M. I. Elmasry, "Optimal synthesis of high-performance architectures," *IEEE JSSC*, 27(3):389-397, Mar. 1992.

[12] C.-Y. Wang and K. K. Parhi, "High level DSP synthesis using the MARS design system," *Proc. ISCAS*, 1992.

[13] C. H. Gebotys, "Synthesizing embedded speed-optimized architectures," *IEEE JSSC*, 28(3):242-252, Mar. 1993.

[14] C. H. Gebotys and M. I. Elmasry, "Global optimization approach for architectural synthesis," *IEEE TCAD*, 12(9):1266-1278, Sep. 1993.

[15] S. Vercauteren, B. Lin, and Hugo De Man, "Constructing application-specific heterogenous embedded architectures from custom HW/SW applications," *33rd DAC,* 1996.

[16] P. G. Paulin, et.al., "Embedded software in real-time signal processing systems: application and architecture trends," *Proc. IEEE*, 85(3):419-435, Mar. 1997.

[17] G. Goossens, et.al., "Embedded software in real-time signal processing systems: design technologies," *Proc. IEEE*, 85(3):436-454, Mar. 1997.

[18] V. Visvanathan and S. Ramanathan, "Synthesis of energy-efficient configurable processor array," *Proc. Intl. Workshop on Parallel Processing,* Bangalore, India, 1994.

[19] G. Long, F. Ling, and J. G. Proakis, "The LMS algorithm with delayed coefficient adaptation," *IEEE TASSP*, 37(9):1397-1405, Sep. 1989, and corrections, 40(1):230-232, Jan. 1992.

[20] T. Ernst and A. Kaelin, "Analysis of the LMS algorithm with delayed coefficient update," *Proc. ISCAS*, 1995.

[21] S-S. Ahn and P. J. Voltz, "Convergence of the DLMS algorithm with decreasing step size," *Proc. ICASSP*, 1997.

[22] A. V. Oppenheim and R. W. Schafer, *Discrete-Time Signal Processing*, Englewood Cliffs, NJ: Prentice-Hall, 1989.

[23] C. E. Leiserson and J. B. Saxe, "Optimizing synchronous systems," *J. VLSI and Computer Systems*, 1(1):41-67, 1983.

[24] S. Ramanathan and V. Visvanathan, "Low-power pipelined LMS adaptive filter architecture with minimal adaptation delay," to appear in *INTEGRATION, the VLSI journal.*

[25] M. B. Srivastava, A. P. Chandrakasan, and R. W. Broderson, "Predictive system shutdown and other architectural techniques for energy efficient programmable computation," *IEEE TVLSI*, 4(1):42-55, Mar. 1996.

[26] V. V. Kaenel, P. Macken, and M. G. R. Degrauwe, "A voltage reduction technique for battery-operated systems," *IEEE JSSC*, 25(5):1136-1140, Oct. 1990.

[27] V. Gutnik and A. P. Chandrakasan, "Embedded power supply for low-power DSP," *IEEE TVLSI*, 5(4):425-435, Dec. 1997.