System Architecture of an Adaptive Reconfigurable DSP Computing Engine

An-Yeu Wu, *Member, IEEE*, K. J. Ray Liu, *Senior Member, IEEE*, and Arun Raghupathy

***Abstract*— Modern digital signal processing (DSP) applications call for computationally intensive data processing at very high data rates. In order to meet the high-performance/low-cost constraints, the state-of-the-art video processor should be a programmable design which performs various tasks in video applications without sacrificing the computational power and the manufacturing cost in exchange for such flexibility. Currently, general-purpose programmable DSP processor and application-specific integrated circuit (ASIC) design are the two major approaches for data processing in practical implementations. In order to meet the high-speed/low-cost constraint, it is desirable to have a programmable design that has the flexibility of the general-purpose DSP processor while the computational power is similar to ASIC designs. In this paper, we present the system architecture of an adaptive reconfigurable DSP computing engine for numerically intensive front-end audio/video communications. The proposed system is a massively parallel architecture that is capable of performing most low-level computationally intensive data processing including finite impulse response/infinite impulse response (FIR/IIR) filtering, subband filtering, discrete orthogonal transforms (DT), adaptive filtering, and motion estimation for the host processor in DSP applications. Since the properties of each programmed function such as parallelism and pipelinability have been fully exploited in this design, the computational speed of this computing engine can be as fast as ASIC designs that are optimized for individual specific applications. We also show that the system can be easily configured to perform multirate FIR/IIR/DT operations at negligible hardware overhead. Since the processing elements are operated at half of the input data rate, we are able to double the processing speed on-the-fly based on the same system architecture without using high-speed/full-custom circuits. The programmable/high-speed features of the proposed design make it very suitable for cost-effective video-rate DSP applications.**

***Index Terms*— Coordinate rotation digital computer (CORDIC), discrete cosine transforms, FIR digital filters, IIR digital filters, multirate processing, quadrature mirror filter, reconfigurable architectures.**

I. Introduction

MODERN communication services such as high-definition TV (HDTV), video-on-demand services (VOD), and PC-based multimedia applications call for computationally intensive data processing at a video data rate which includes low-level tasks like discrete cosine transform (DCT) and filtering/convolution operations as well as medium-level tasks like motion estimation (ME), variable length coding (VLC), and vector quantization (VQ). All these tasks require millions-to-billions of additions/multiplications per second to ensure the real-time performance of those applications. As a result, the computational power of the traditional general-purpose programmable digital signal processing (DSP) processor [1] is not sufficient under such a speed constraint. Although the performance of the DSP processor can be improved by using advanced VLSI technology and special arithmetic kernels [2], [3], the manufacturing cost is significantly increased due to the complexity of the DSP processor. On the other hand, dedicated very large scale integration (VLSI) application-specific integrated circuit (ASIC) designs, which are optimally designed for given DSP functions, can handle the speed-demanding tasks. However, since a collection of ASIC chips are required to perform various tasks, it results in higher manufacturing cost and larger system complexity. Therefore, we are motivated to design a high-speed DSP computing engine with the flexibility of a general DSP processor while meeting the stringent speed requirement as in the ASIC designs.

In this paper, we propose a unified approach to integrate the rotation-based finite impulse response/infinite impulse response (FIR/IIR) structure, quadrature mirror filter (QMF) lattice structure [4], discrete transform (DT) architecture [5], [6], and adaptive recursive least square (RLS) lattice filter [7] into one reconfigurable parallel processing architecture. It can serve as a computing engine in the audio/video system to perform those front-end computationally intensive DSP functions for the host processor. We first extract the common inherent features of each function, then design a programmable rotation-based computational module that can serve as a basic processing element (PE) in all desired functions. The overall system of the proposed DSP computing engine consists of an array of P identical programmable computational modules and one reconfigurable interconnection network. Each computational module can be adaptively changed to the basic PE in each desired DSP function by setting suitable parameters and switches. Next, according to the data paths, the interconnection network is configured to connect/combine the appropriate module outputs to perform the programmed function. When the system is in the execution mode, all PE's are operating in parallel, and the output data can be collected by the host processor in a fully pipelined way. Since the properties of each programmed function such as parallelism and pipelinability

Manuscript received July 27, 1995; revised October 15, 1996. This paper was recommended by Associate Editor B. Zeng. This work was supported in part by the ONR Grant N00014-93-10566 and the NSF NYI Award MIP9457397.

A.-Y. Wu is with the Electrical Engineering Department, National Central University, Chung-li, Taiwan, R.O.C.

K. J. R. Liu and A. Raghupathy are with the Electrical Engineering Department and Institute for Systems Research, University of Maryland, College Park, MD 20742 USA.

Publisher Item Identifier S 1051-8215(98)01162-8.

have been fully exploited, the maximum processing speed of the proposed design can be as fast as dedicated ASIC designs. Our approach is similar to the CORDIC-based FIR, IIR, and DCT architectures in the literature [8]–[10], but the functionality is much more general-purpose. It can be classified as the *algorithm-driven* solution [11] since the programmability and ASIC-like processing speed of our design are achieved by fully exploiting the inherent properties of the given DSP algorithms. Besides, the proposed architecture is very suitable for VLSI implementation due to its modularity and regularity.

Next, we will show how to improve the speed performance of the system by using the multirate approach. In video-rate signal processing, the data throughput rate is limited by the maximum processing speed of the PE's. The demanding speed requirement in general calls for the use of expensive high-speed multiplier/adder circuits or full-custom designs. Hence, the cost and the design cycle will increase drastically. Recently, it has been shown that the multirate approach is a powerful tool for high-speed/low-power DSP applications [12]–[14]. By employing the multirate architecture with decimation factor equal to two, we can process data at, for example, 100 MHz rate while using only 50-MHz PE's. Thus, we can speed up the system at the algorithmic/architectural level without using expensive high-speed circuits or advanced VLSI technologies. We will show that we can map the multirate FIR/IIR/DT operations onto our design. As a result, we can double the speed performance of the DSP computing engine on-the-fly by simply reconfigurating the programmable modules and the interconnection network.

In the last part of this paper, we incorporate the feature of adaptive filtering into our design. The RLS filter, which is widely used in channel equalization, system identification, and image restoration, has become another computationally intensive key component in PCS equipment. The reason is that wireless communication requires fast adaptation to highly nonstationary mobile channels. We will show that, with little modification of the programmable module design, the proposed system architecture can also perform the QR-decomposition-based RLS lattice algorithm (QRD-LSL) [7] in a fully pipelined way.

The purpose of this work is to map an important set of signal processing algorithms onto a common parallel core which is of CORDIC nature. Although there already exist some works on CORDIC-based designs of DSP algorithms, none of those works, to our best knowledge, has a unified architecture that can be easily reconfigurated to perform such a large set of DSP functions (FIR, IIR, QMF, eight DT's, and adaptive filtering) as presented in this paper. In addition, the proposed architecture is very different from the traditional multiply-and-accumulate (MAC)-based DSP processor. The latter approach relies on very high-speed arithmetic logic units (ALU's) to perform DSP functions serially. The speed performance will be degraded as the order/block-size of the DSP functions increase. Our design, on the contrary, utilizes P identical programmable modules running in parallel to perform those DSP functions in a fully pipelined way. The system throughput rate remains constant until the order/block-size exceeds the total available programmable modules. The parallel processing nature together with the programmability of the DSP computing engine makes it very suitable for cost-effective, high-speed DSP applications.

The organization of the this paper is as follows: Section II discusses the basic operations of the FIR, IIR, QMF filtering, and discrete transforms. Section III presents the design and operation of the proposed reconfigurable DSP computing engine for the operations of FIR/QMF/IIR/DT. In Section IV, the speed-up of the DSP computing engine based on the multirate approach is discussed. The incorporation of the QRD-LSL algorithm into our design is presented in Section V. The comparison of the proposed computing engine with other existent approaches is discussed in Section VI followed by the conclusions.

II. Basic Computational Module in the FIR/QMF/IIR/DT

In this section, we present a unified treatment of FIR filtering, QMF bank, IIR filtering, and DTs to identify their basic computational modules. Then, based on the common features of those computational modules in each DSP function, we derive a unified programmable module that can integrate those basic computational modules in FIR/QMF/IIR/DT.

A. Basic Module in FIR

The FIR filter is widely used in DSP applications. In addition to the MAC implementation of the filtering operation, an alternative realization of the FIR filter is the lattice structure as shown in Fig. 1 [15]. It consists of N basic lattice sections that are connected in a cascade form. The advantages of the lattice structure over the MAC-based implementation are its robustness to the coefficient quantization effect as well as the smaller dynamic range, which is due to the orthogonal operation used in each lattice.

Given an Nth-order FIR transfer function

$$H(z) = 1 - \sum_{m=0}^{N-1} a_m z^{-(m+1)} \tag{1}$$

the FIR lattice parameters can be computed as follows [16].

1) *Initialization:* $a_m^{(N-1)} = a_m, m = 0, 1, \cdots, N-1.$

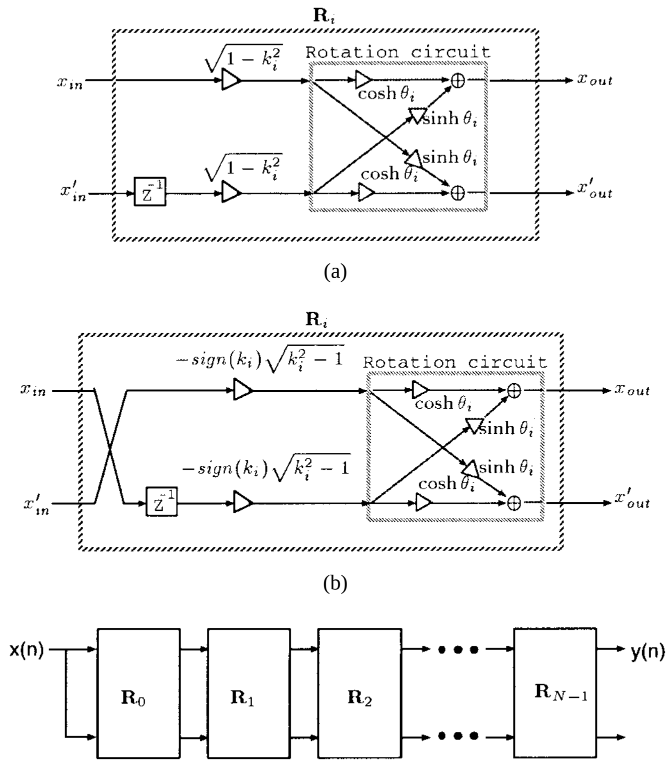
2) For $i = N-1, N-2 \cdots, 0$

$$k_i = a_i^{(i)},$$
$$a_m^{(i-1)} = \frac{a_m^{(i)} + k_i a_{i-m}^{(i)}}{1 - k_i^2}, \quad m = 0, 1, \cdots, (i-1). \tag{2}$$

end

where the parameter k_i's, $i = 0, 1, \cdots, N-1$, are known as the *reflection coefficients*, or the *PARtial CORrelation* (PARCOR) coefficients in the theory of linear prediction [17]. After k_i's are computed, the lattice section of the FIR filter can be described by

$$\begin{bmatrix} x_{\text{out}} \\ x'_{\text{out}} \end{bmatrix} = \begin{bmatrix} 1 & -k_i \\ -k_i & 1 \end{bmatrix} \begin{bmatrix} x_{\text{in}} \\ x'_{\text{in}} \end{bmatrix} \tag{3}$$



(a)

(b)

(c)

Fig. 1. Lattice FIR structure: (a) basic FIR lattice filter section with $|k_i| < 1$, (b) basic FIR lattice filter section with $|k_i| > 1$, and (c) overall filtering structure.

or equivalently

1) For $|k_i| < 1$

$$\begin{bmatrix} x_{\text{out}} \\ x'_{\text{out}} \end{bmatrix} = \begin{bmatrix} \dfrac{1}{\sqrt{1-k_i^2}} & \dfrac{-k_i}{\sqrt{1-k_i^2}} \\ \dfrac{-k_i}{\sqrt{1-k_i^2}} & \dfrac{1}{\sqrt{1-k_i^2}} \end{bmatrix} \cdot \begin{bmatrix} \sqrt{1-k_i^2} & 0 \\ 0 & \sqrt{1-k_i^2} \end{bmatrix} \begin{bmatrix} x_{\text{in}} \\ x'_{\text{in}} \end{bmatrix} = \begin{bmatrix} \cosh\theta_i & \sinh\theta_i \\ \sinh\theta_i & \cosh\theta_i \end{bmatrix} \cdot \begin{bmatrix} \sqrt{1-k_i^2} & 0 \\ 0 & \sqrt{1-k_i^2} \end{bmatrix} \begin{bmatrix} x_{\text{in}} \\ x'_{\text{in}} \end{bmatrix} \tag{4}$$

with

$$\theta_i = \tanh^{-1}(-k_i). \tag{5}$$

2) For $|k_i| > 1$

$$\begin{bmatrix} x_{\text{out}} \\ x'_{\text{out}} \end{bmatrix} = \begin{bmatrix} \dfrac{|k_i|}{\sqrt{k_i^2-1}} & \dfrac{-\text{sign}(k_i)}{\sqrt{k_i^2-1}} \\ \dfrac{-\text{sign}(k_i)}{\sqrt{k_i^2-1}} & \dfrac{|k_i|}{\sqrt{k_i^2-1}} \end{bmatrix} \cdot \begin{bmatrix} -\text{sign}(k_i)\sqrt{k_i^2-1} & 0 \\ 0 & -\text{sign}(k_i)\sqrt{k_i^2-1} \end{bmatrix} \cdot \begin{bmatrix} x'_{\text{in}} \\ x_{\text{in}} \end{bmatrix} = \begin{bmatrix} \cosh\theta_i & \sinh\theta_i \\ \sinh\theta_i & \cosh\theta_i \end{bmatrix} \cdot \begin{bmatrix} -\text{sign}(k_i)\sqrt{k_i^2-1} & 0 \\ 0 & -\text{sign}(k_i)\sqrt{k_i^2-1} \end{bmatrix} \cdot \begin{bmatrix} x'_{\text{in}} \\ x_{\text{in}} \end{bmatrix} \tag{6}$$

where $\text{sign}(k_i)$ denotes the sign of k_i and θ is defined by

$$\theta_i = \tanh^{-1}(-1/k_i). \tag{7}$$

The lattice modules to realize (4) and (6) are depicted in Fig. 1(a) and (b), respectively. Each module is composed of two scaling multipliers and one rotation circuit. In general, the rotation circuit can be implemented by using either general-purpose multipliers/adders or the CORDIC processor in *hyperbolic mode* [9]. Note that we need to swap the two inputs for the case of $|k_i| > 1$ since the input vector is inverted in (6). These two basic modules constitute the FIR lattice structure as shown in Fig. 1(c). The operation of the lattice structure is parallel in nature. It performs FIR filtering in a fully pipelined way. The throughput rate is bounded by the maximum processing speed of the lattice module (scaling plus rotation), which is independent of tap length N.

B. Basic Module in QMF

The QMF plays a key role in image compression and subband coding [18], [19]. In [4], the two-channel paraunitary QMF lattice was proposed. It possesses all the advantages of the lattice structure such as robustness to coefficient quantization, smaller dynamic range, and modularity. Such properties are preferable when the filter bank is implemented in fixed-point arithmetic. Fig. 2 shows the QMF lattice structure, where part (c) is the analysis bank and part (d) is the synthesis bank. We can see that the QMF lattice is very similar to the FIR lattice except that the inputs of the lattice become the decimated sequences of the input signal, and two scaling multipliers are set to one. If CORDIC processor is employed to realize the rotation circuit in the QMF lattice, it works in the *circular mode* to perform the rotation operations. Its speed performance is twice as fast as the FIR lattice since the data rate is halved after the downsampling operation.

The conversion of a two-channel (real-coefficient, FIR) paraunitary QMF bank to the QMF lattice is as follows. Given a predesigned power-symmetric FIR analysis filter $H_0(z)$ with unit sample response $h_0(n), n = 0, 1, \cdots, N$, we can first find the unit sample response of the other analysis filter $H_1(z)$ by

$$h_1(n) = (-1)^n h_0(N-n). \tag{8}$$

Then the rotation angle θ_i in each QMF module can be computed by the following [20, Ch. 6].

1) *Initialization:* $H_0^{(J)}(z) = H_0(z)$ and $H_1^{(J)}(z) = H_1(z)$ with $N = 2J + 1$.
2) For $i = J, J-1, \cdots, 0$

$$\begin{aligned} (1+\alpha_i^2)H_0^{(i-1)}(z) &= H_0^{(i)}(z) - \alpha_i H_1^{(i)}(z), \\ (1+\alpha_i^2)z^{-2}H_1^{(i-1)}(z) &= \alpha_i H_0^{(i)}(z) + H_1^{(i)}(z), \\ \theta_i &= \tan^{-1}\alpha_i, \end{aligned} \tag{9}$$

end

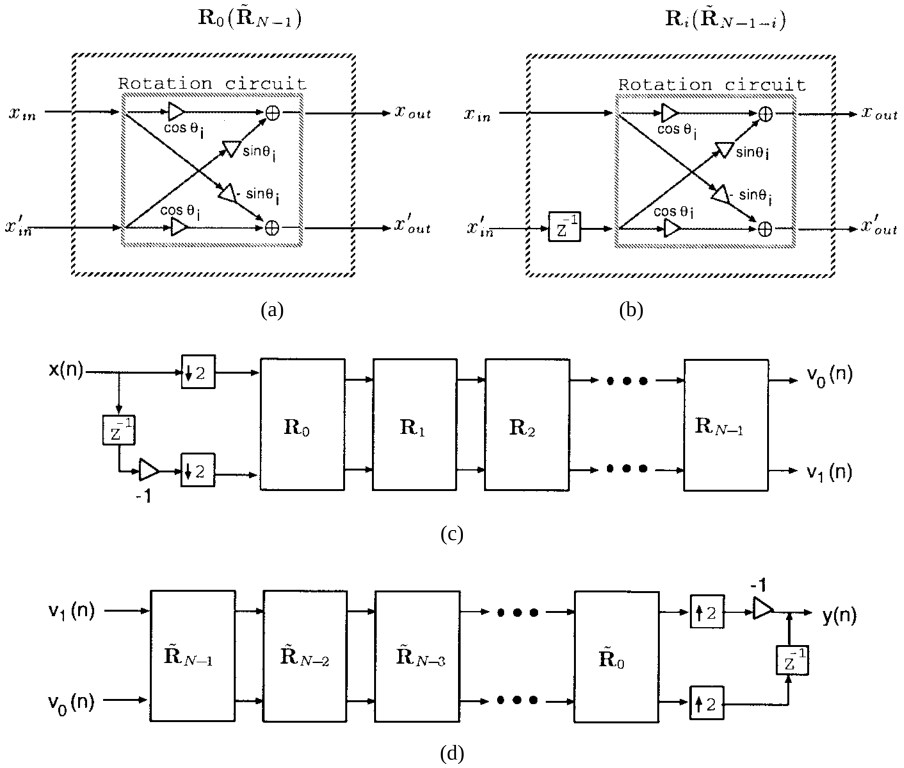


Fig. 2. The two-channel paraunitary QMF lattice: (a), (b) basic lattice section, (c) the analysis bank, and (d) the synthesis bank.

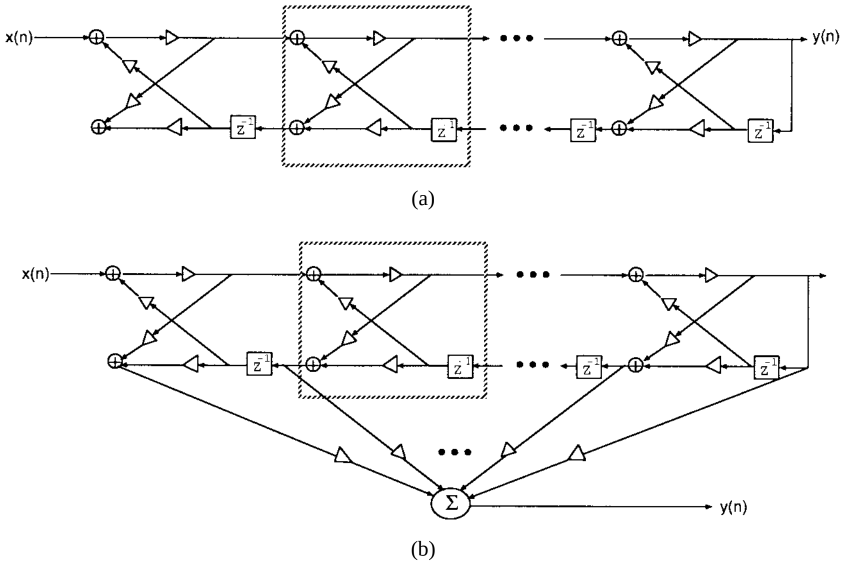


Fig. 3. (a) All-pole IIR lattice structure and (b) general IIR (ARMA) lattice structure.

The coefficient α_i is computed by setting the highest power of z^{-1} in $H_0^{(i)}(z) - \alpha_i H_1^{(i)}(z)$ equal to zero.

C. Basic Module in IIR

Next we want to consider the basic module in IIR filtering. The lattice structure of an IIR system (all-pole and ARMA) [15] is shown in Fig. 3. Although the basic lattice module in IIR filters is similar to the one in FIR lattice, the opposite data flow in the IIR module makes it difficult to be incorporated into our unified design. Besides, the modularity of the lattice structure is no longer maintained if we want to implement a general IIR (ARMA) filter [see Fig. 3(b)]. Therefore, we are motivated to find an IIR lattice structure that has similar data paths as in the FIR/QMF lattice while retaining the property of modularity.

Fig. 4 shows the lattice structure that can be used to realize a second-order IIR filter. It can be shown that the transfer

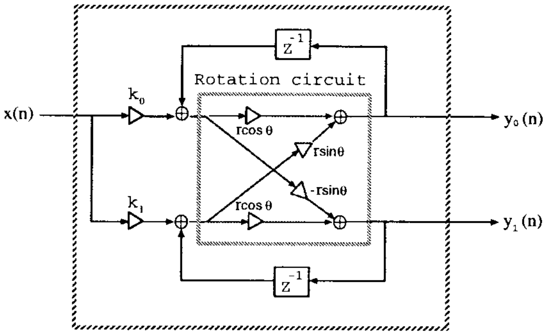


Fig. 4. Rotation-based second-order IIR lattice module.

functions of the two outputs are given by

$$\tilde{H}_0(z) = \frac{Y_0(z)}{X(z)} = \frac{r(k_0\cos\theta + k_1\sin\theta) - r^2k_0z^{-1}}{1 - 2r\cos\theta z^{-1} + r^2z^{-2}} \quad (10)$$

$$\tilde{H}_1(z) = \frac{Y_1(z)}{X(z)} = \frac{r(k_1\cos\theta - k_0\sin\theta) - r^2k_1z^{-1}}{1 - 2r\cos\theta z^{-1} + r^2z^{-2}}. \quad (11)$$

Now given an even-order real-coefficient IIR (ARMA) filter $H(z)$, we can first rewrite it in cascade form

$$H(z) = \prod_{i=0}^{N/2-1} H_i(z) \quad (12)$$

and each subfilter $H_i(z)$ is of the form

$$\begin{aligned} H_i(z) &= \frac{c_i + d_iz^{-1} + e_iz^{-2}}{1 + a_iz^{-1} + b_iz^{-2}} \\ &= c_i + z^{-1}\frac{d_i' + e_i'z^{-1}}{1 + a_iz^{-1} + b_iz^{-2}} \\ &= c_i + z^{-1}A_i(z) \end{aligned} \quad (13)$$

where $d_i' = d_i - a_ic_i$ and $e_i' = e_i - b_ic_i$. Comparing (10) and (11) with (13), we see that $A_i(z)$ can be realized by either $\tilde{H}_0(z)$ or $\tilde{H}_1(z)$ with appropriate settings of k_0, k_1, r, and θ. The conversion of those parameters is derived in Appendix A, where $\tilde{H}_0(z)$ is chosen for the realization.

Now based on (12) and (13), we can realize $H(z)$ using the structure depicted in Fig. 5. Each stage performs the filtering of $H_i(z)$ for $i = 0, 1, \cdots, N/2-1$, where $A_i(z)$ is realized by the second-order IIR module in Fig. 4. Note that the scaling multiplier c_i is also realized by the IIR module by setting $k_0 = c_i, k_1 = 0, r = 1, \theta = 0$, and disconnecting the feedback data paths. We use a dashed box to symbolize this implementation. By doing so, both modules to realize c_i and $A_i(z)$ will have the same latency. Hence, the two inputs of the adders in Fig. 5 can be synchronized. To perform an Nth-order (N is even) $H(z)$, we need a total of N IIR lattice modules running in parallel. The maximum data throughput rate is bounded by the feedback loop within the IIR module.

D. Basic Module in Discrete Transforms

The DT's are the kernel operations in the transform coding applications and signal compression schemes. Recently, we have developed a unified time-recursive IIR-based DT architecture [13] which shows that most DT's can be represented as linear combinations of two functions defined by

$$X_C(k) \triangleq \beta\sum_{n=0}^{L-1}\cos[(2n+1)\omega_k + \eta_k]x(n) \quad (14)$$

$$X_S(k) \triangleq \beta\sum_{n=0}^{L-1}\sin[(2n+1)\omega_k + \eta_k]x(n) \quad (15)$$

for $k = 0, 1, \cdots, N-1$. The only differences among various DT's are the setting of the parameters in (14) and (15) and how to combine $X_C(k)$ and $X_S(k)$ together. As an example, the discrete Hartley transform (DHT) can be represented as

$$X_{\mathrm{DHT}}(k) = X_C(k) + X_S(k) \quad (16)$$

by setting

$$L = N, \quad \beta = \frac{1}{\sqrt{N}}, \quad \omega_k = \frac{-k\pi}{N}, \quad \text{and} \quad \eta_k = -\omega_k. \quad (17)$$

The other example is the modulated lapped transform (MLT) [21]. It can be computed by

$$X_{\mathrm{MLT}}(k) = -s_k[X_C(k+1) + X_S(k)] \quad (18)$$

for $k = 0, 1, \cdots, N-1$, where the scaling factor $s_k = (-1)^{(k+2)/2}$ if k is even, $s_k = (-1)^{(k-1)/2}$ if k is odd, and the parameter settings are

$$L=2N, \quad \beta=\frac{1}{\sqrt{2N}}, \quad \omega_k=\frac{\pi k}{2N}, \quad \text{and} \quad \eta_k=\frac{\pi}{2}\left(k+\frac{1}{2}\right). \quad (19)$$

Equations (16) and (18) are referred to as the *combination functions* which describe how to combine $X_C(k)$ and $X_S(k)$ together to obtain the desired DT coefficients. By the use of (14), (15), and the combination functions, we are able to perform most of the existing DT's. The parameter settings and the corresponding combination functions of most DT's are listed in Table I, and the resulting IIR-based unified DT architecture is described in [13].

In order to incorporate the unified DT operations into our design, we need a rotation-based computational module, instead of the IIR-based one in [13], as the processing element. In [5], [6], a rotation-based module was derived for the dual generation of $X_C(k)$ and $X_S(k)$ [see Fig. 6(a)], where the scaling multipliers and the rotation operation are given by

$$\boldsymbol{f}_k = \begin{bmatrix} f_{0,k} \\ f_{1,k} \end{bmatrix} = \begin{bmatrix} \beta\cos((2L+1)\omega_k + \eta_k) \\ \beta\sin((2L+1)\omega_k + \eta_k) \end{bmatrix} \quad (20)$$

and

$$\boldsymbol{R}_k = \begin{bmatrix} \cos(2\omega_k) & \sin(2\omega_k) \\ -\sin(2\omega_k) & \cos(2\omega_k) \end{bmatrix} \quad (21)$$

respectively. This module works in a serial-input-parallel-output (SIPO) way: the block input data is fed serially into the module. After the updating of the last datum is completed, the values of $X_C(k)$ and $X_S(k)$ in (14) and (15) are available at the module outputs.

The aforementioned module can be used as a basic building block to implement the rotation-based DT architecture.

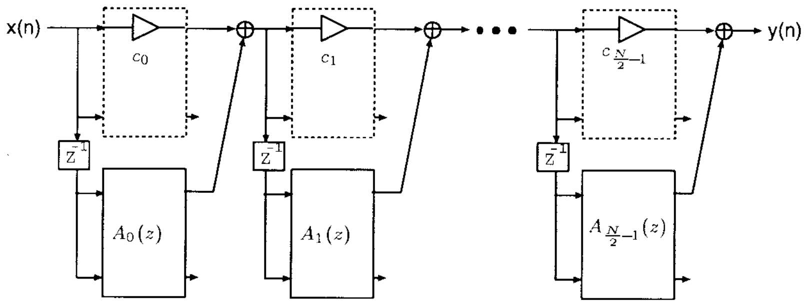


Fig. 5. IIR (ARMA) structure based on the second-order IIR lattice module.

TABLE I
PARAMETER SETTING OF THE UNIFIED DISCRETE TRANSFORMATION ARCHITECTURE, WHERE $\mathrm{Re}\{X_{\mathrm{DFT}}(k)\}$ AND $\mathrm{Im}\{X_{\mathrm{DFT}}(k)\}$ DENOTE THE REAL PART AND THE IMAGINARY PART OF THE DFT, RESPECTIVELY. c_k IS THE SCALING FACTOR USED IN THE DCT/DST AND THEIR TRANSFORMS

	L	β	ω_k	η_k	Combination Function
DCT	N	c_k	$\frac{k\pi}{2N}$	0	$X_{DCT}(k) = X_C(k)$
IDCT	N	c_1	$\frac{\pi}{2N}(k+\frac{1}{2})$	$-\omega_k$	$X_{IDCT}(k) = X_C(k) + (c_0 - c_1)x(0)$
DST-IV [27]	N	c_1	$\frac{\pi}{2N}(k+\frac{1}{2})$	0	$X_{DST}(k) = X_S(k)$
IDST-IV [27]	N	c_1	$\frac{\pi}{2N}(k+\frac{1}{2})$	0	$X_{IDST}(k) = X_S(k)$
MLT [21]	$2N$	$\frac{1}{\sqrt{2N}}$	$\frac{k\pi}{2N}$	$\frac{\pi}{2}(k+\frac{1}{2})$	$X_{MLT}(k) = -s_k[\,X_C(k+1) + X_S(k)\,]$
ELT [28]	$4N$	$\frac{1}{2\sqrt{2N}}$	$\frac{\pi}{2N}(k+\frac{1}{2})$	$\frac{\pi}{2}(k+\frac{1}{2})$	$X_{ELT}(k) = -X_S(k+1) + \sqrt{2}X_C(k) + X_S(k-1)$
DFT	N	$\frac{1}{\sqrt{N}}$	$-\frac{k\pi}{N}$	$-\omega_k$	$Re\{X_{DFT}(k)\} = X_C(k),\ Im\{X_{DFT}(k)\} = X_S(k).$
DHT	N	$\frac{1}{\sqrt{N}}$	$-\frac{k\pi}{N}$	$-\omega_k$	$X_{DHT}(k) = X_C(k) + X_S(k).$

Fig. 6(b) illustrates the overall rotation-based MLT architecture for the case of $N = 8$. It consists of two parts: One is the *programmable module array* which computes $X_C(k)$ and $X_S(k), k = 0, 1, \cdots, N-1$, in parallel. The other is the *interconnection network* which selects and combines the array outputs to generate the desired DT coefficients according to the combination functions defined in Table I. It is similar to the unified DT architecture discussed in [13] except that the IIR-based computational module is replaced with the rotation-based module in Fig. 6(a).

E. Unified Programmable Module Design

From Figs. 1, 2, 4–6, we observe that all the architectures share a common computational module with only some minor differences in the data paths, the module parameters (multiplier coefficients and rotation angle), and the way the modules are connected. With this observation in mind, we first integrate those basic computational modules into one unified programmable module as shown in Fig. 7. It consists of six switches, four scaling multipliers, and one rotation circuit. One pipelining stage (the dash line in Fig. 7) is inserted after the scaling multipliers $f_{0,i}$ and $f_{1,i}$, respectively, to shorten the critical path of the programmable module. By setting the switches, multiplier coefficients, and the rotating angle, the unified programmable module can be programmed to act as the basic PE in FIR/QMF/IIR/DT. The detailed settings are discussed below.

The six switches $[s_0s_1s_2s_3s_4s_5]$ control the data paths inside the module: the switch pair s_0 and s_1 select the input from either in_i or in_i': with $s_0s_1 = 00, in_i$ becomes the common input of the lattice which is required in the first stage of FIR and in the IIR module. Using $s_0s_1 = 10$, we can swap the inputs for the FIR lattice with $|k_i| > 1$. Switches s_2 and s_3 decide if the delay element is used or not. With $s_2s_3 = 01$, the lower-left delay element is included in the data path, which is required in the FIR/QMF lattice (except the first stage in QMF banks). With the setting $s_2s_3 = 11$, the delay element in Fig. 5 can be incorporated into the module $A_i(z)$. Therefore, we do not need to implement the delay elements explicitly

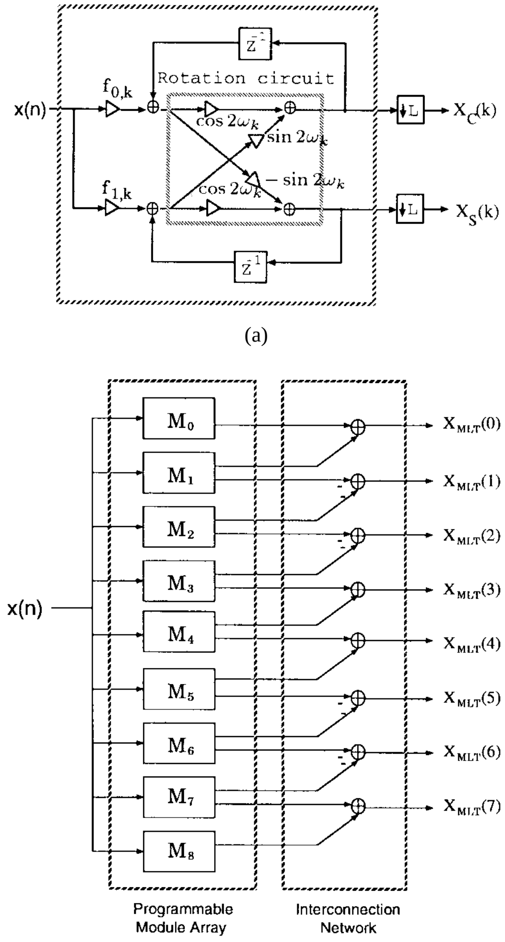


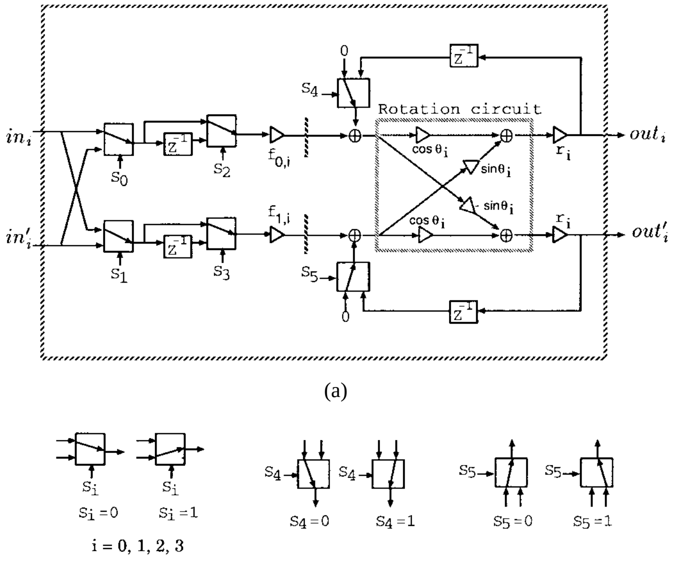
Fig. 6. SIPO MLT architecture. (a) Rotation-based module for the dual generation of $X_C(k)$ and $X_S(k)$, where the downsampling operation $\downarrow L$ denotes that we pick up the values of $X_C(k)$ and $X_S(k)$ at the Lth clock cycle and ignore all the previous intermediate results. (b) Overall MLT transform architecture.

in the IIR filtering operation. The last switch pair is s_4 and s_5. They control the two feedback paths in the module: when $s_4 s_5 = 11$, the delayed module outputs are added with the current inputs as required in the IIR and DT operations. The setting $s_4 s_5 = 00$ will disconnect the feedback paths. The two multipliers r_i's at the outputs of the rotation circuit are required only when we want to incorporate IIR function into this unified module design.

In addition to the data paths, we also need to set the values of the scaling multipliers $f_{0,i}, f_{1,i}$, and r_i, as well as the rotating angle θ_i. Given the DSP function specification, those parameters can be determined from our discussions in Section II-A–D. The complete settings of the programmable module for the FIR/QMF/IIR/DT operations are listed in Table II.

III. DSP Computing Engine Design for the FIR/QMF/IIR/DT

In this section, the design of the proposed DSP computing engine under normal operation (without speed-up) is discussed. We will show that, by appropriately setting the parameters



(b)

Fig. 7. (a) Unified programmable module for the FIR/QMF/IIR/DT. (b) Switches used in the module.

of the unified modules discussed in the previous section, and connecting them via a reconfigurable interconnection network, we are able to perform all functions in the FIR/QMF/IIR/DT in a fully pipelined way.

A. Operation of the DSP Computing Engine

In previous section, we have derived a unified programmable module that can be used as the basic PE in the operations of FIR/QMF/IIR/DT. Since in each function of the FIR/QMF/IIR/DT, the basic PE's are connected in different way, we employ a reconfigurable interconnection network to perform the connection task. Fig. 8 shows the overall architecture of the proposed DSP computing engine under FIR filtering mode. It consists of two parts: one is the *programmable module array* with P identical unified programmable modules. The other is the *reconfigurable interconnection network* which connects those programmable modules according to the data paths. In the FIR/QMF/IIR operations, the data are processed in a serial-input-serial-output (SISO) way. Hence, the programmable modules need to be cascaded for those operations. For example, the FIR modules can be connected by setting the interconnection network as shown in Fig. 8. We can also realize the connections of the IIR modules in Fig. 5 by using the network setting as shown in Fig. 9. On the other hand, the DT architecture in Section II-D performs the block transforms in an SIPO way. The interconnection network is configured according to the combination functions defined in Table I. The detailed settings of the interconnection network used in this paper (Type I-IX) are described in Table III.

The operation of the DSP computing engine is as follows: during the *initialization mode*, the host processor will compute all the necessary parameters $\boldsymbol{f}_i, r_i, \theta_i$ according to the function type (FIR/QMF/IIR/DT) and the function specification as listed in Table II. In general, the DSP functions to be

TABLE II
SETTING OF THE ITH PROGRAMMABLE MODULE, WHERE $N_{\max}$ IS THE MAXIMUM ORDER (OR BLOCK SIZE IN DT) THAT CAN BE REALIZED BY A P-MODULE ARRAY, AND SWITCH s_6 IS ONLY USED IN THE QRD-LSL OPERATION

	$\mathbf{S} = [s_0 s_1 s_2 s_3 s_4 s_5 s_6]$	$\mathbf{f}_i = [f_{0,i}, f_{1,i}]^T$	$\mathbf{R}_i$	Network type	$N_{\max}$
FIR $(\lvert k_i \rvert < 1)$	$\begin{cases} 000100 & (M_0) \\ 010100 & (M_i,\ i \neq 0) \end{cases}$	$\begin{bmatrix} \sqrt{1-k_i^2} \\ \sqrt{1-k_i^2} \end{bmatrix}$ with k_i defined in (2)	$\begin{bmatrix} \cosh\theta_i & \sinh\theta_i \\ \sinh\theta_i & \cosh\theta_i \end{bmatrix}$, with θ_i defined in (5)	Type I	P
FIR $(\lvert k_i \rvert > 1)$	$\begin{cases} 000100 & (M_0) \\ 100100 & (M_i,\ i \neq 0) \end{cases}$	$\begin{bmatrix} -sign(k_i)\sqrt{k_i^2-1} \\ -sign(k_i)\sqrt{k_i^2-1} \end{bmatrix}$ with k_i defined in (2)	$\begin{bmatrix} \cosh\theta_i & \sinh\theta_i \\ \sinh\theta_i & \cosh\theta_i \end{bmatrix}$, with θ_i defined in (7)	Type I	P
QMF (analysis)	$\begin{cases} 010000 & (M_0) \\ 010100 & (M_i,\ i \neq 0) \end{cases}$	$\begin{bmatrix} 1 \\ 1 \end{bmatrix}$	$\begin{bmatrix} \cos\theta_i & \sin\theta_i \\ -\sin\theta_i & \cos\theta_i \end{bmatrix}$, with θ_i defined in (9)	Type I	$2P-1$
QMF (synthesis)	$\begin{cases} 010000 & (\tilde{M}_{N-1}) \\ 010100 & (\tilde{M}_i,\ i \neq N-1) \end{cases}$	$\begin{bmatrix} 1 \\ 1 \end{bmatrix}$	$\begin{bmatrix} \cos\tilde{\theta}_i & \sin\tilde{\theta}_i \\ -\sin\tilde{\theta}_i & \cos\tilde{\theta}_i \end{bmatrix}$ with$\tilde{\theta}_i = \theta_{N-1-i}$, where θ_{N-1-i} is defined in (9)	Type I	$2P-1$
IIR (c_i)	000000	$\begin{bmatrix} c_i \\ 0 \end{bmatrix}$ with c_i defined in (13)	$r_i = 1$ and $\theta_i = 0$	Type III	P
IIR $(A_i(z))$	001111	$\begin{bmatrix} k_0 \\ k_1 \end{bmatrix}$ defined in (43)	$\begin{bmatrix} r_i\cos\theta_i & r_i\sin\theta_i \\ -r_i\sin\theta_i & r_i\cos\theta_i \end{bmatrix}$, with r_i, θ_i defined in $(39) or (41)$	Type III	P
DT	000011	$\begin{bmatrix} \beta\cos((2L+1)\omega_i+\eta_i) \\ \beta\sin((2L+1)\omega_i+\eta_i) \end{bmatrix}$ with L, β, ω_i, η_i defined in Table 1	$\begin{bmatrix} \cos 2\omega_i & \sin 2\omega_i \\ -\sin 2\omega_i & \cos 2\omega_i \end{bmatrix}$, with ω_i defined in Table 1	Type V-VIII	P
Multirate FIR $(\lvert k_i \rvert < 1)$	$\begin{cases} 000100 & (M_0, M_1, M_2) \\ 010100 & (M_i,\ i > 2) \end{cases}$	$\begin{bmatrix} \sqrt{1-k_i^2} \\ \sqrt{1-k_i^2} \end{bmatrix}$ with k_i defined in (2)	$\begin{bmatrix} \cosh\theta_i & \sinh\theta_i \\ \sinh\theta_i & \cosh\theta_i \end{bmatrix}$, with θ_i defined in (5)	Type II	$\frac{2P}{3}$
Multirate FIR $(\lvert k_i \rvert > 1)$	$\begin{cases} 000100 & (M_0, M_1, M_2) \\ 100100 & (M_i,\ i \neq 2) \end{cases}$	$\begin{bmatrix} -sign(k_i)\sqrt{k_i^2-1} \\ -sign(k_i)\sqrt{k_i^2-1} \end{bmatrix}$ with k_i defined in (2)	$\begin{bmatrix} \cosh\theta_i & \sinh\theta_i \\ \sinh\theta_i & \cosh\theta_i \end{bmatrix}$, with θ_i defined in (7)	Type II	$\frac{2P}{3}$
Multirate IIR (c_i)	000000	$\begin{bmatrix} c_i \\ 0 \end{bmatrix}$ with c_i defined in (13)	$r_i = 1$ and $\theta_i = 0$	Type IV	$\frac{P}{3}$
Multirate IIR $(A_i(z))$	001111	$\begin{bmatrix} k_0 \\ k_1 \end{bmatrix}$ defined in (43)	$\begin{bmatrix} r_i\cos\theta_i & r_i\sin\theta_i \\ -r_i\sin\theta_i & r_i\cos\theta_i \end{bmatrix}$, with r_i, θ_i defined in $(39) or (41)$	Type IV	$\frac{P}{3}$
Multirate DT	000011	$\begin{bmatrix} \beta\cos((2L+2l+1)\omega_i+\eta_i) \\ \beta\cos((2L+2l+1)\omega_i+\eta_i) \end{bmatrix}$ with $l = i \bmod 2$, and L, β, ω_i, η_i defined in Table 1	$\begin{bmatrix} \cos 4\omega_i & \sin 4\omega_i \\ -\sin 4\omega_i & \cos 4\omega_i \end{bmatrix}$, with ω_i defined in Table 1	Type V-VIII	$\frac{P}{2}$
QRD-LSL	$\begin{cases} 0100101 & (M_{4i}) \\ 0101101 & (M_{4i+1}) \\ 0100100 & (M_{4i+2},\ M_{4i+3}) \end{cases}$	$\begin{bmatrix} 0 \\ 1 \end{bmatrix}$	$\begin{cases} R_A(\theta) = M_{4i}, M_{4i+1} \\ R_R(\theta) = M_{4i+2}, M_{4i+3} \end{cases}$	Type IX	$\frac{P}{4}$

performed are determined beforehand. All the parameters can be computed in advance so that the host processor can find the necessary parameters through table-look-up to reduce the set-up time in this mode. Next, the host processor needs to configure the interconnection network according to the function type as listed in Table III.

Once the computing engine has been initialized, it enters the *execution mode*. In the applications of FIR/IIR/QMF, the host processor continuously feeds the data sequence into the computing engine. All PE's are running in parallel and the host processor can collect the filtering outputs in a fully pipelined way. In the block DT application, the block input data is fed into the computing engine serially. Each PE of the programmable module array updates $X_C(k)$ and $X_S(k)$ in (14) and (15), $k = 0, 1, \cdots, N-1$, simultaneously. After the last datum enters the programmable module array, the evaluation of DT coefficients is completed. Then the interconnection network will combine the module outputs according to the combination function defined in Table I, and the transform coefficients can be obtained in parallel at the outputs of the network. At the same time, the host processor will reset all internal registers (delay elements) of the programmable modules to zero so that the next block transform can be conducted immediately.

B. Design Examples

In what follows, we will use some design examples to demonstrate how to convert a given system specification to the parameters used in the programmable modules. The orders of the numerator and the denominator in the IIR ARMA filter are restricted to be even so that we can perform all the necessary decompositions. Here, a ten-module DSP computing engine is used to carry out the given function. As a result, the maximum order of the FIR/IIR/QMF is ten and the transform size of the DT is also limited to ten. For the DT, we use an eight-point DCT as the design example due to its prevalence in the application of transform coding.

1) FIR Filtering: Given the FIR transfer function

$$\begin{aligned} H(z) = {} & 1 - 0.8843z^{-1} - 0.1327z^{-2} - 1.1219z^{-3} \\ & + 0.5328z^{-4} - 0.8882z^{-5} \\ & + 0.1038z^{-6} - 0.3786z^{-7} \end{aligned}$$

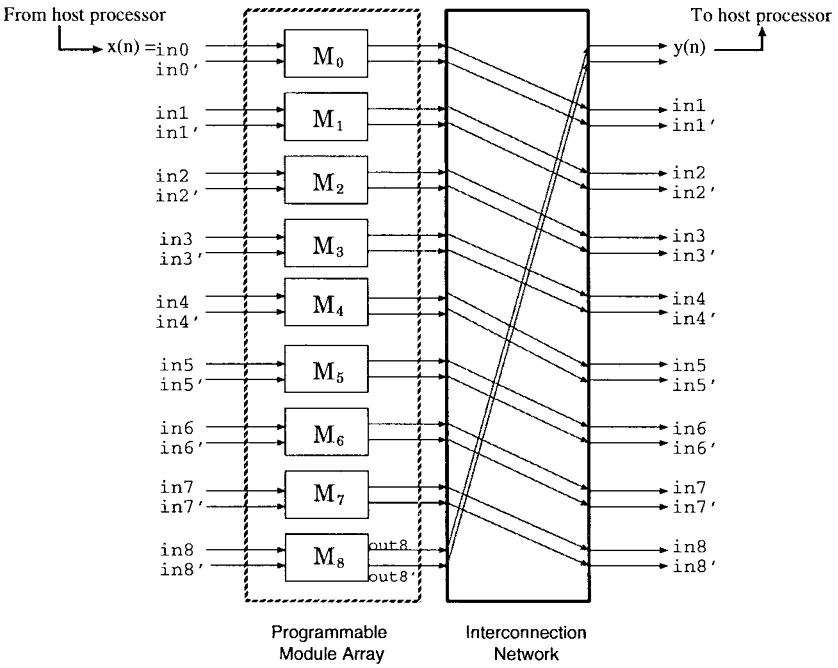


Fig. 8. Overall architecture of the DSP computing engine for FIR filtering.

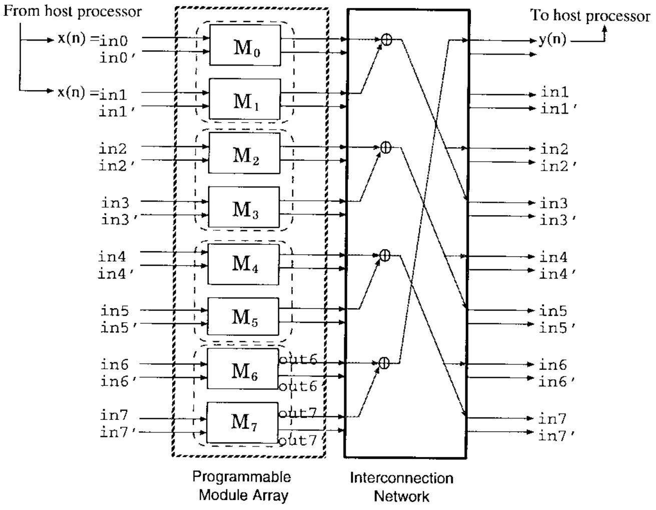


Fig. 9. Overall architecture of the DSP computing engine for IIR (ARMA) filtering.

$$+0.2195z^{-8} - 0.1094z^{-9} \tag{22}$$

we first apply (1)–(2) to compute all PARCOR coefficients

$$\begin{aligned} k_0 &= -0.4472, \quad k_1 = -0.6917, \quad k_2 = -0.5865, \\ k_3 &= -4.1573, \quad k_4 = 1.1595, \quad k_5 = 0.2655, \\ k_6 &= 0.2942, \quad k_7 = -0.1243, \quad k_8 = 0.1094. \end{aligned}$$

Then all parameters of each module, such as $f_{0,i}, f_{1,i},$ and θ_i, can be found by using (4)–(7). The complete settings are listed in Table IV(a).

2) QMF Filtering: Suppose that the predesigned power-symmetric low-pass filter described in Example 5.3.2 of [20] is used for the QMF filtering. We have the analysis filter

$$H_0(z) = \sum_{n=0}^{N-1} h_0(n)z^{-n} \tag{23}$$

with

$$\begin{aligned} h_0(0) &= 0.1605, \quad h_0(1) = 0.4156, \quad h_0(2) = 0.4592, \\ h_0(3) &= 0.1487, \quad h_0(4) = -0.1642, \quad h_0(5) = -0.1245, \\ h_0(6) &= 0.0825, \quad h_0(7) = 0.0888, \quad h_0(8) = -0.0508, \\ h_0(9) &= -0.0608, \quad h_0(10) = 0.0352, \quad h_0(11) = 0.0399, \\ h_0(12) &= -0.0256, \quad h_0(13) = -0.0244, \\ h_0(14) &= 0.0186, \quad h_0(15) = 0.0135, \quad h_0(16) = -0.0131, \\ h_0(17) &= -0.0074, \quad h_0(18) = 0.0129, \\ h_0(19) &= -0.0050. \end{aligned}$$

We can go through (8)–(9) to find all θ_i's in the modules, and the results are shown in Table IV(b).

3) IIR (ARMA) Filtering: Given the IIR (ARMA) filter

$$H(z) = \frac{1 + \sum_{i=1}^{M} p_i z^{-i}}{1 + \sum_{i=1}^{N} q_i z^{-i}} \tag{24}$$

with $M = 4$, $N = 10$, and

$$\begin{aligned} p_1 &= -1.7314, \quad p_2 = 1.6788, \quad p_3 = -0.7913, \\ p_4 &= 0.2304, \quad q_1 = 0.4036, \quad q_2 = 1.3227, \\ q_3 &= 0.2376, \quad q_4 = 1.1558, \quad q_5 = 0.0047, \quad q_6 = 0.6950, \\ q_7 &= -0.0733, \quad q_8 = 0.2735, \quad q_9 = -0.0542, \\ q_{10} &= 0.0788 \end{aligned}$$

TABLE III

SWITCH SETTING OF THE INTERCONNECTION NETWORK, WHERE N DENOTES THE ORDER OF THE FIR/QMF/IIR/QRD-LSL OR THE BLOCK SIZE OF THE DT

Type I (FIR, QMF)	Type II (Multirate FIR)	Type III (IIR)
1. $in_0 = x(n)$. 2. For $m = 0, 1, \ldots, (N-2)$ $in_{m+1} = out_m$, $in'_{m+1} = out'_m$. end 3. $y(n) = out_{N-1}$.	1. $in_i = x_i(n),\ i = 0, 1, 2$. 2. For $m = 0, 1, \ldots, (\frac{3N}{2} - 4)$ $in_{m+3} = out_m$, $in'_{m+3} = out'_m$. end 3. $y_i(n) = out_{(\frac{3N}{2}-3)+i}$, $i = 0, 1, 2$.	1. $in_i = x(n),\ i = 0, 1$. 2. For $m = 0, 1, \ldots, (N-3)$ $in_{m+2} = out_{2\lfloor \frac{m}{2} \rfloor} + out_{2\lfloor \frac{m}{2} \rfloor + 1}$. end 3. $y(n) = out_{N-1} + out_{N-2}$.
Type IV (Multirate IIR)	Type V (DCT)	Type VI (MLT)
1. $in_i = x_{\lfloor \frac{i}{2} \rfloor}(n),\ i = 0, 1, \ldots, 5$. 2. For $m = 0, 1, \ldots, (3N-7)$ $in_{m+6} = out_{2\lfloor \frac{m}{2} \rfloor} + out_{2\lfloor \frac{m}{2} \rfloor + 1}$. end 3. $y_i(n) = out_{(3N-5)+2i} + out_{(3N-6)+2i}$, $i = 0, 1, 2$.	1. $in_i = x(n)$, $i = 0, 1, \ldots, N-1$. 2. $X_{DCT}(i) = out_i$, $i = 0, 1, \ldots, N-1$.	1. $in_i = x(n)$, $i = 0, 1, \ldots, N-1$. 2. $X_{MLT}(i) = -s_i(out_{i+1} + out'_i)$, $i = 0, 1, \ldots, N-1$.
Type VII (DFT)	Type VIII (DHT)	Type IX (QRD-LSL)
1. $in_i = x(n)$, $i = 0, 1, \ldots, N-1$. 2. $X_{DFT}(i) = out_i + j * out'_i$, $i = 0, 1, \ldots, N-1$.	1. $in_i = x(n)$, $i = 0, 1, \ldots, N-1$. 2. $X_{DHT}(i) = out_i + out'_i$, $i = 0, 1, \ldots, N-1$.	1. $in'_i = x(n),\ i = 0, 1$. 2. For $m = 0, 1, \ldots, (4N-3)$ $in'_{m+2} = out'_m$, if $(m \bmod 4 = 0)$ then $\mu_{in}(m+3) = \mu_{out}(m)$, $\mu_{in}(m+2) = \mu_{out}(m+1)$. end 3. $f(n) = out'_{4N-2}$, $b(n) = out'_{4N-1}$.

we first rewrite it in cascade form

$$
\begin{aligned}
H(z) = &\left(1 + z^{-1}\frac{-0.2122 + 0.2175z^{-1}}{1 - 0.9192z^{-1} + 0.4225z^{-2}}\right)\\
&\times\left(1 + z^{-1}\frac{0.1500 - 0.2025z^{-1}}{1 - 0.7500z^{-1} + 0.5625z^{-2}}\right)\\
&\times\left(1 + z^{-1}\frac{-0.8000 - 0.6400z^{-1}}{1 + 0.8000z^{-1} + 0.6400z^{-2}}\right)\\
&\times\left(1 + z^{-1}\frac{-1.2728 - 0.8100z^{-1}}{1 + 1.2728z^{-1} + 0.8100z^{-2}}\right)\\
&\times\left(1 + z^{-1}\frac{-0.6400z^{-1}}{1 + 0.6400z^{-2}}\right). \qquad (25)
\end{aligned}
$$

Following the steps described in (38)–(43), we can find the parameters used in each second-order subfilter. The corresponding settings can be found in Table IV(c).

4) Block DCT: For the eight-point block DCT, we can calculate $f_{0,i}, f_{1,i}$, and θ_i of each programmable module by using Table I. The settings are listed in Table IV(d).

IV. SPEEDUP OF THE DSP COMPUTING ENGINE USING MULTIRATE APPROACH

In video-rate signal processing, the fundamental bottleneck is the processing speed of the processing elements. Although the aforementioned computing engine architecture has fully exploited the parallelism and pipelinability of each programmed function, the input data rate is still limited by the maximum speed of the adders and multipliers inside the programmable module. In applications such as HDTV and VOD, the demanding speed constraint will result in the use of expensive high-speed VLSI circuits and/or full-custom designs. Thus, the cost as well as the design cycle will be drastically increased.

In [12]–[14], it has been shown that the multirate approach provides a direct and efficient way to achieve very high-speed data processing at the algorithmic/architectural level. Therefore, if we can find a way to reconfigure the computing engine to perform multirate operations, the aforementioned speed constraint can be resolved. In this section, we will show

TABLE IV
SETTING OF THE (a) FIR FILTER, (b) QMF FILTER, (c) IIR (ARMA) FILTER, AND (d) EIGHT-POINT DCT

	M_0	M_1	M_2	M_3	M_4	M_5	M_6	M_7	M_8
S	000100	010100	010100	100100	100100	010100	010100	010100	010100
$f_{0,i}$	0.8944	0.7222	0.8100	4.0352	-0.5870	0.9641	0.9557	0.9922	0.9940
$f_{1,i}$	0.8944	0.7222	0.8100	4.0352	-0.5870	0.9641	0.9557	0.9922	0.9940
r_i	1	1	1	1	1	1	1	1	1
θ_i	0.4812	0.8512	0.6723	0.2454	-1.3027	-0.2720	-0.3032	0.1249	-0.1098
Interconnection	Type I								

(a)

	M_0	M_1	M_2	M_3	M_4	M_5	M_6	M_7	M_8	M_9
S	010000	010100	010100	010100	010100	010100	010100	010100	010100	010100
$f_{0,i}$	1	1	1	1	1	1	1	1	1	1
$f_{1,i}$	1	1	1	1	1	1	1	1	1	1
r_i	1	1	1	1	1	1	1	1	1	1
θ_i	-1.2022	0.6993	-0.4465	0.3051	-0.2146	0.1511	-0.1043	0.0690	-0.0426	0.0311
Interconnection	Type I									

(b)

	M_0	M_1	M_2	M_3	M_4	M_5	M_6	M_7	M_8	M_9
S	000000	001111	000000	001111	000000	001111	000000	001111	000000	001111
$f_{0,i}$	1	-0.5148	1	0.3600	1	1	1	1	1	1
$f_{1,i}$	0	0.0531	0	0.0231	0	-0.5774	0	-1	0	0
r_i	1	0.6500	1	0.7500	1	0.8000	1	0.9000	1	0.8000
θ_i	0	0.7854	0	1.0472	0	2.0944	0	2.3562	0	1.5708
Interconnection	Type III									

(c)

	M_0	M_1	M_2	M_3	M_4	M_5	M_6	M_7
S	000011	000011	000011	000011	000011	000011	000011	000011
$f_{0,i}$	0.3536	-0.4904	0.4619	-0.4157	0.3536	-0.2778	0.1913	-0.0975
$f_{1,i}$	0	-0.0975	0.1913	-0.2778	0.3536	-0.4157	0.4619	-0.4904
r_i	1	1	1	1	1	1	1	1
θ_i	0	0.3927	0.7854	1.1781	1.5708	1.9635	2.3562	2.7489
Interconnection	Type V							

(d)

how to map the multirate FIR/IIR/DT architectures with speed-up of two to our computing engine design. Since processing elements now operate at only half of input data rate, the speed performance of the proposed DSP engine can be doubled on-the-fly based on the same programmable modules and interconnection network.

A. Multirate FIR Architecture

The multirate FIR filtering architecture was proposed in [22] and [23]. Fig. 10 shows the multirate architecture to realize a given Nth-order FIR filter $H(z)$, where $H_0(z), H_1(z)$ are the *polyphase components* [20] of $H(z)$, and $\hat{H}(z) \triangleq H_1(z) + H_2(z)$. As we can see, the multirate architecture can be readily applied to the speed-up of the filtering operations. For example, it can process data at 100 MHz rate while only 50-MHz processing elements are required.

To map this multirate FIR architecture to our design, we first find the three $(N/2)$th-order FIR subfilters $H_0(z), H_1(z)$, and $\hat{H}(z)$ of the given FIR transfer function. Then we can implement each subfilter using the FIR lattice structure discussed in Section II-A. The resulting architecture is depicted in Fig. 11(a), where $\tilde{R}_i, \hat{R}_i$, and $\overline{R}_i$ correspond to the ith basic modules used in $H_0(z), \hat{H}(z)$, and $H_1(z)$, respectively. Note that each basic FIR module can be realized by the unified computational module in Fig. 7. Hence, we can map Fig. 11(a) onto our computing engine design with the mapping

$$\tilde{R}_i \rightarrow M_{3i}, \quad \hat{R}_i \rightarrow M_{3i+1}, \quad \overline{R}_i \rightarrow M_{3i+2} \qquad (26)$$

for $i = 0, 1, \cdots, N/2-1$. Besides, the interconnection network is set to Type II for the connections. Fig. 11(b) illustrates the realization of a multirate sixth-order FIR by using nine programmable modules. The detailed settings are described in Tables II and III.

The operation of the DSP computing engine to perform multirate FIR filtering is as follows. Once the programmable modules and interconnection network have been initialized, the host processor sends data at f_s rate to the downsampling circuit in Fig. 10. Then the outputs of the downsampling circuit,

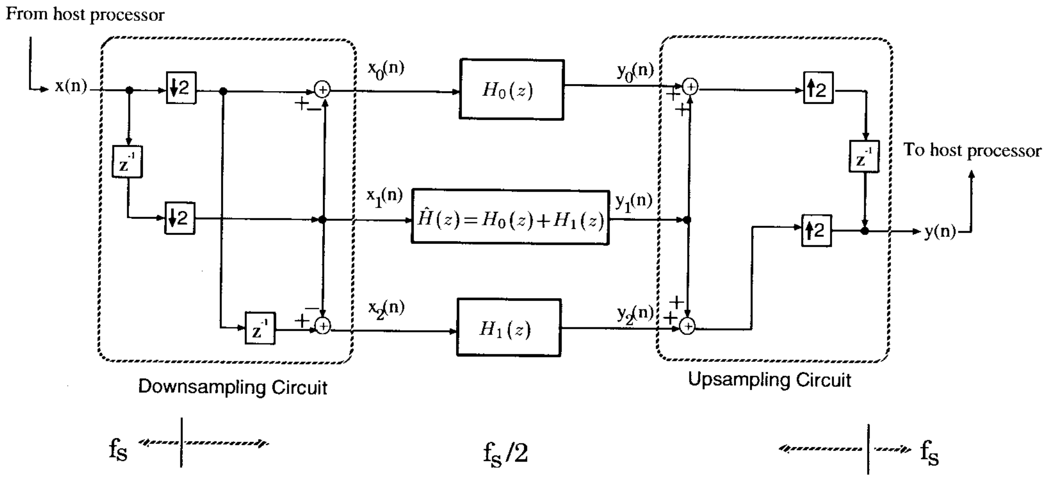
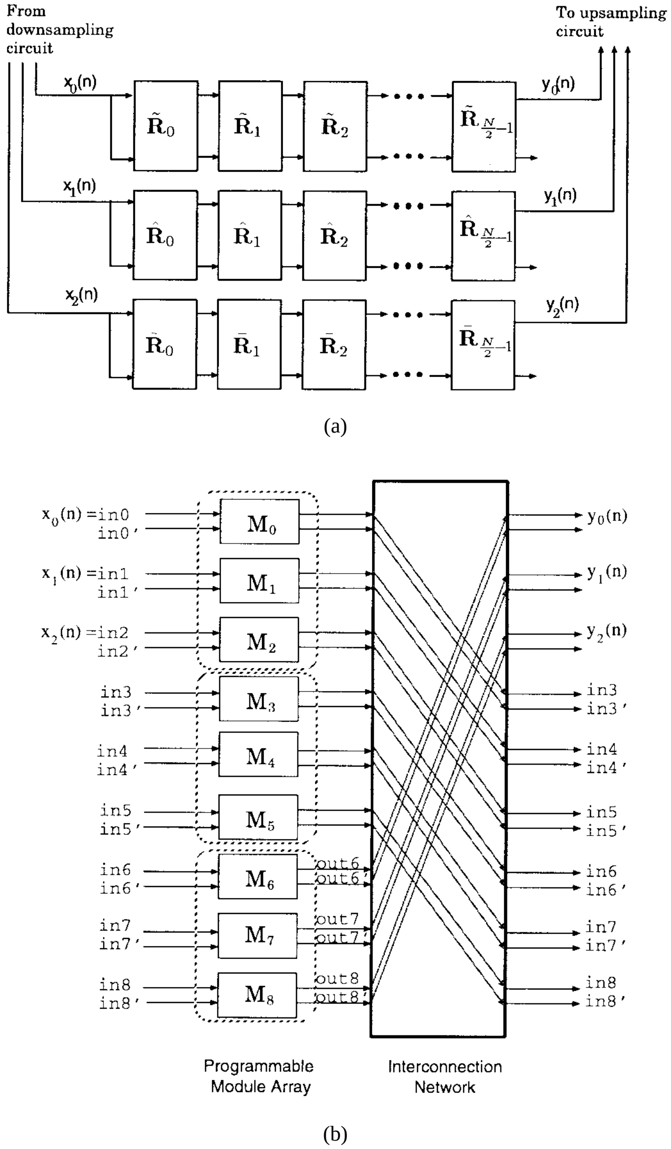


Fig. 10. Multirate filtering architecture in [22], where f_s is the data sample rate.



(a)

(b)

Fig. 11. (a) Multirate FIR filtering structure based on the FIR lattice structure. (b) Mapping part (a) to the DSP computing engine architecture; the figure demonstrates the multirate sixth-order FIR architecture using nine programmable modules.

$x_i(n), i=0,1,2,$ will be processed by the three FIR subfilters in parallel at only $f_s/2$ rate. After the subfilter outputs $y_i(n)$'s are generated, the FIR filtering output $y(n)$ is reconstructed through the upsampling circuit in a fully pipelined way, and the data rate of $y(n)$ is back to f_s. Since all PE's are running in the $f_s/2$ region, now the data rate is twice as fast as the one in Fig. 8. Namely, we double the speed performance at the architectural level without any specially designed high-speed circuits.

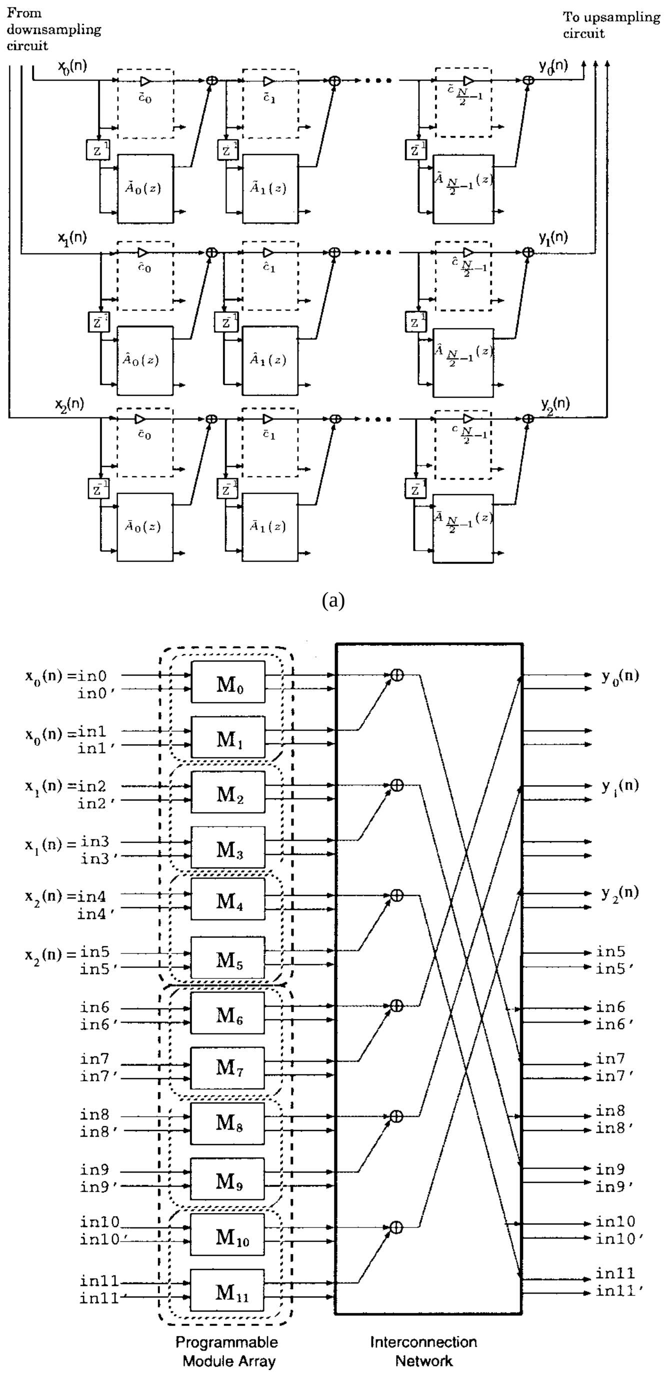
As we can see, the only new hardware for this multirate operation is the downsampling and upsampling circuits in Fig. 10 for the pre- and post-processing of the data. Since we need $N/2$ modules for the implementation of each subfilter, a total of $3N/2$ modules will be used to perform an Nth-order multirate FIR filtering operation; i.e., we only need 50% more PE's for the doubled speed performance. This overhead is handled by simply activating more PE's in the programmable module array and reconfigurating the interconnection network instead of implementing new types of PE's and new interconnection network explicitly.

B. Multirate IIR Architecture

The proposed computing engine design can also be reconfigurated to perform the multirate IIR filtering. Given an IIR system

$$H'(z)=\frac{1+\sum_{i=1}^{M}p_i z^{-i}}{1+\sum_{i=1}^{N}q_i z^{-i}} \tag{27}$$

($M \leq N, M, N$ are even numbers), we can follow the derivations in Appendix B to find the polyphase components of $H'(z), H_0'(z)$, and $H_1'(z)$, as well as $\hat{H}'(z) \triangleq H_0'(z)+H_1'(z)$. After substituting $H_0'(z), H_1'(z)$, and $\hat{H}'(z)$ for $H_0(z), H_1(z)$, and $\hat{H}(z)$, respectively, the multirate filtering structure in Fig. 10 becomes a multirate IIR filtering architecture with downsampling rate equal to two.



(a)

(b)

Fig. 12. (a) Multirate IIR filtering structure based on the IIR lattice structure discussed in Section II-C. (b) Mapping part (a) to the DSP computing engine architecture; the figure demonstrates the multirate fourth-order IIR architecture using 12 programmable modules.

The mapping of the multirate IIR filtering is similar to the FIR case. We first implement each of the subfilters $H_0'(z), H_1'(z)$, and $\hat{H}'(z)$, using the cascade IIR structure discussed in Section II-C. The corresponding parallel architecture is shown in Fig. 12(a), where $\{\tilde{c}_i, \tilde{A}_i(z)\}, \{\hat{c}_i, \hat{A}_i(z)\}$, and $\{\overline{c}_i, \overline{A}_i(z)\}, i = 0, 1, \cdots, N/2-1$, are used to realize the subfilters of $H_0'(z), \hat{H}'(z), H_1'(z)$, respectively. Then it can be mapped to our system architecture by

$$\begin{aligned} &\tilde{c}_i \rightarrow M_{3i}, \quad \hat{c}_i \rightarrow M_{3i+2}, \quad \overline{c}_i \rightarrow M_{3i+4}, \\ &\tilde{A}_i(z) \rightarrow M_{3i+1}, \quad \hat{A}_i(z) \rightarrow M_{3i+3}, \quad \overline{A}_i(z) \rightarrow M_{3i+5} \end{aligned} \tag{28}$$

for $i = 0, 1, \cdots, N/2-1$. Fig. 12(b) demonstrates the multirate fourth-order IIR architecture using 12 programmable modules. The detailed settings of the modules and interconnection network can be found in Tables II and III. Note that the order of the denominators in $H_0'(z)$ and $H_1'(z)$ is still N. This indicates that the use of Fig. 10 will triple the hardware cost; i.e., we will need $3N$ modules to realize an Nth-order multirate IIR filter.

C. Multirate Discrete Transform Architecture

In addition to multirate FIR/IIR filtering, the proposed system can also be reconfigured to perform multirate DT operations [13]. Note that the multirate DT operations in [13] are performed using the IIR-based computational modules. Here we derive the rotation-based multirate DT architecture so that it can be mapped to our design.

Split the input data sequence, $x(n), i = 0, 1, \cdots, L$, into the *even* sequence

$$x_e(n) = x(2n), \qquad n = 0, 1, \cdots, L/2-1 \tag{29}$$

and the *odd* sequence

$$x_o(n) = x(2n+1), \qquad n = 0, 1, \cdots, L/2-1 \tag{30}$$

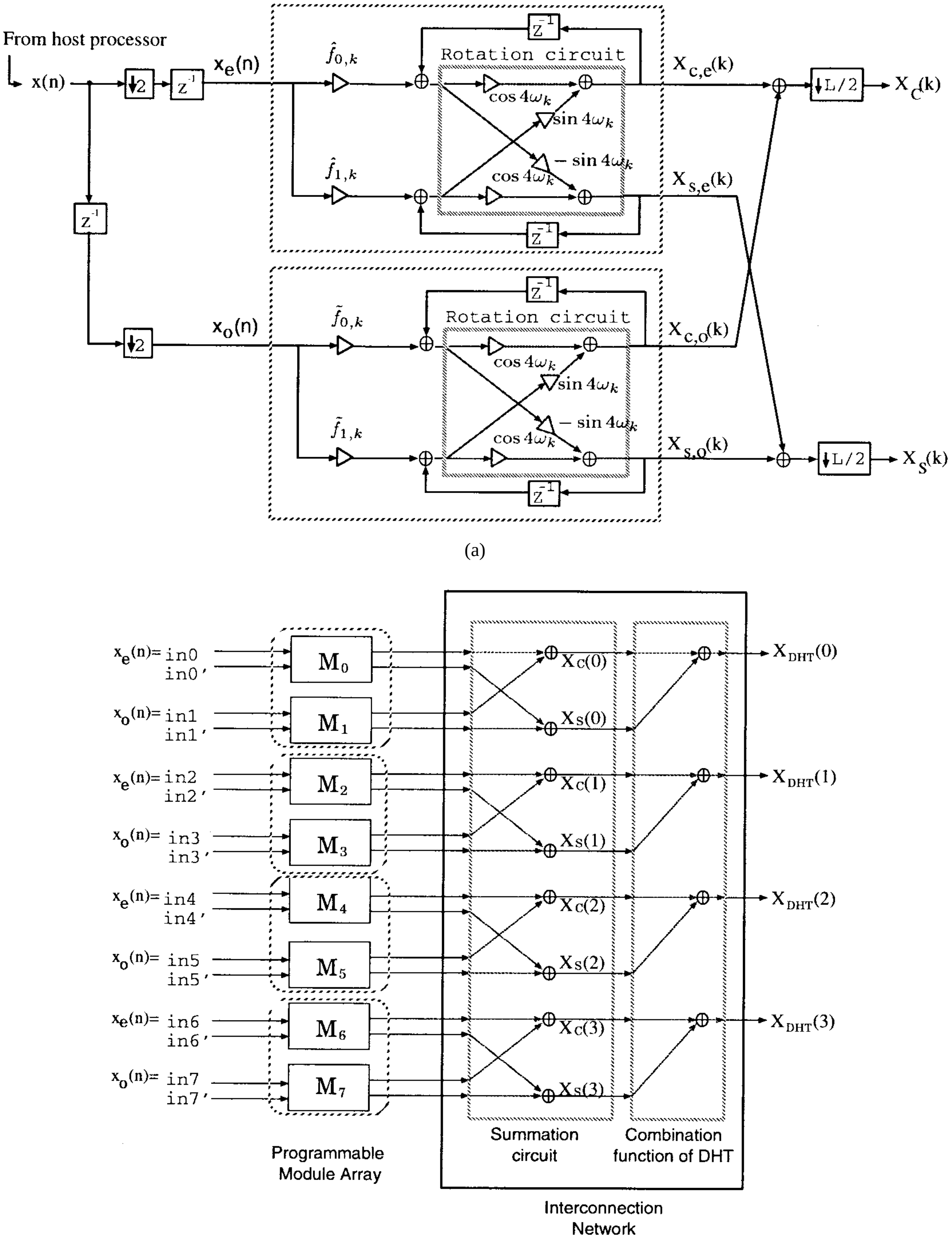
(14) and (15) can be rewritten as

$$\begin{aligned} X_C(k) =& \beta \sum_{n=0}^{L/2-1} \cos[(4n+1)\omega_k + \eta_k] x_e(n) \\ &+ \beta \sum_{n=0}^{L/2-1} \cos[(4n+3)\omega_k + \eta_k] x_o(n) \\ =& X_{C,e}(k) + X_{C,o}(k) \end{aligned} \tag{31}$$

$$\begin{aligned} X_S(k) =& \beta \sum_{n=0}^{L/2-1} \sin[(4n+1)\omega_k + \eta_k] x_e(n) \\ &+ \beta \sum_{n=0}^{L/2-1} \sin[(4n+3)\omega_k + \eta_k] x_o(n) \\ =& X_{S,e}(k) + X_{S,o}(k) \end{aligned} \tag{32}$$

for $k = 0, 1, \cdots, N-1$. Here we use $X_{C,e}(k)$ and $X_{C,o}(k)$ to denote the components of $X_C(k)$ that are generated by $x_e(n)$ and $x_o(n)$, respectively. Similarly, $X_{S,e}(k)$ and $X_{S,o}(k)$ are the even and odd components of $X_S(k)$. Following the derivations in [5] and [6], it can be shown that we can use the rotation-based module in Fig. 6(a) for the dual generation of even components, $X_{C,e}(k)$ and $X_{S,e}(k)$, by setting

$$\begin{aligned} \boldsymbol{f}_{k,e} &= \begin{bmatrix} \hat{f}_{0,k} \\ \hat{f}_{1,k} \end{bmatrix} = \begin{bmatrix} \beta \cos((2L+1)\omega_k + \eta_k) \\ \beta \sin((2L+1)\omega_k + \eta_k) \end{bmatrix} \\ \boldsymbol{R}_{k,e} &= \begin{bmatrix} \cos(4\omega_k) & \sin(4\omega_k) \\ -\sin(4\omega_k) & \cos(4\omega_k) \end{bmatrix}. \end{aligned} \tag{33}$$



(a)

(b)

Fig. 13. (a) Multirate architecture for the dual generation of $X_C(k)$ and $X_S(k)$. (b) Multirate four-point DHT architecture based on eight programmable modules.

Similarly, the same module can be used to obtain the odd components, $X_{C,o}(k)$ and $X_{S,o}(k)$, by setting

$$\boldsymbol{f}_{k,o}=\begin{bmatrix}\tilde{f}_{0,k}\\ \tilde{f}_{1,k}\end{bmatrix}=\begin{bmatrix}\beta\cos((2L+3)\omega_k+\eta_k)\\ \beta\sin((2L+3)\omega_k+\eta_k)\end{bmatrix}$$

$$\boldsymbol{R}_{k,o}=\begin{bmatrix}\cos(4\omega_k) & \sin(4\omega_k)\\ -\sin(4\omega_k) & \cos(4\omega_k)\end{bmatrix}. \quad (34)$$

The parallel architecture to realize (31)–(34) is depicted in Fig. 13(a). The input data sequence $x(n)$ is first decimated into two decimated sequences, $x_e(n)$ and $x_o(n)$, through the decimator.[1] Then $X_{C,e}(k), X_{S,e}(k), X_{C,o}(k)$, and $X_{S,o}(k)$ are generated by the two modules in parallel, and the outputs are summed up to obtain $X_C(k)$ and $X_S(k)$.

The multirate DT architecture can be mapped to the computing engine design by setting the parameters of module

[1] The extra delay element on the top is used to synchronize the two decimated sequences so that they can arrive at the two rotation modules at the same time.

TABLE V
SETTING OF THE MULTIRATE (a) FIR FILTER, (b) IIR (ARMA) FILTER, AND (c) EIGHT-POINT DCT

	M_0	M_1	M_2	M_3	M_4	M_5	M_6	M_7	M_8	M_9	M_{10}	M_{11}
S	000100	000100	000100	010100	100100	010100	010100	100100	010100	010100	010100	010100
$f_{0,i}$	0.9878	0.1154	-0.6574	0.8833	-0.4092	0.8005	0.9902	84.0896	0.9613	0.9756	0.3073	0.9923
$f_{1,i}$	0.9878	0.1154	-0.6574	0.8833	-0.4092	0.8005	0.9902	84.0896	0.9613	0.9756	0.3073	0.9923
r_i	1	1	1	1	1	1	1	1	1	1	1	1
θ_i	-0.1571	0.0731	0.8086	0.5086	-1.6262	0.6920	0.1406	0.0119	0.2827	0.2231	1.8484	0.1244
Interconnection	Type II											

(a)

	M_0	M_1	M_2	M_3	M_4	M_5	M_6	M_7	M_8	M_9	M_{10}	M_{11}
S	000000	001111	000000	001111	000000	001111	000000	001111	000000	001111	000000	001111
$f_{0,i}$	0	-2.0706	0	-1.1526	0	-0.6991	1	0.7742	2.4192	2.0129	0	-0.5266
$f_{1,i}$	0	0.2303	0	0.7602	0	1.6195	0	0.3163	0	0.6021	0	2.3668
r_i	1	0.8100	1	0.8100	1	0.8100	1	0.4225	1	0.4225	1	0.4225
θ_i	0	2.0943	0	2.0943	0	2.0943	0	1.5709	0	1.5709	0	1.5709
Interconnection	Type IV											

(b)

	M_0	M_1	M_2	M_3	M_4	M_5	M_6	M_7
S	000011	000011	000011	000011	000011	000011	000011	000011
$f_{0,i}$	0.5000	0.5000	-0.6533	-0.2706	0.5000	-0.5000	-0.2706	0.6533
$f_{1,i}$	0	0	-0.2706	-0.6533	0.5000	0.5000	-0.6533	0.2706
r_i	1	1	1	1	1	1	1	1
θ_i	0	0	0.3927	1.1781	0.7854	2.3562	1.1781	3.5343
Interconnection	Type V							

(c)

M_{2k} to $\boldsymbol{f}_{k,e}, \boldsymbol{R}_{k,e}$ and M_{2k+1} to $\boldsymbol{f}_{k,o}, \boldsymbol{R}_{k,o}$, respectively, for $k = 0, 1, \cdots, L/2 - 1$. Fig. 13 shows the multirate four-point DHT architecture based on eight programmable modules. The speed performance is doubled as in the multirate FIR/IIR case. There are two parts inside the interconnection network: one is the summation circuit to combine the even and odd outputs of the array. The other is the circuit to perform the combination function defined in Table I. In practical implementation, we can either add one summation circuit so that the switch settings of the DT in Table III can still be used, or we can define new switch settings by merging these two circuits together. The hardware overhead to perform the multirate DT is the doubled complexity.

D. Design Examples Using Multirate Operations

1) Multirate FIR Filtering: Given the FIR transfer function in (22), we first perform the polyphase decomposition which yields

$$\begin{aligned} H_0(z) =& 1 - 0.1327z^{-1} + 0.5328z^{-2} + 0.1038z^{-3} \\ &+ 0.2195z^{-4} \\ H_1(z) =& -0.8843 - 1.1219z^{-1} - 0.8882z^{-2} - 0.3786z^{-3} \\ &- 0.1094z^{-4} \\ \hat{H}(z) =& 0.1157 - 1.2546z^{-1} - 0.3554z^{-2} - 0.2748z^{-3} \\ &+ 0.1101z^{-4}. \end{aligned} \tag{35}$$

Then we can follow the steps in (1)–(2) and (4)–(7) to find the parameters for each filter in (35). The results are listed in Table V(a).

2) Multirate IIR (ARMA) Filtering: Now consider the IIR (ARMA) filter shown below

$$\begin{aligned} H'(z) \\ = \frac{1 - 0.4000z^{-1} + 0.1600z^{-2}}{1 - 1.8192z^{-1} + 2.0598z^{-2} - 1.1248z^{-3} + 0.3422z^{-4}}. \end{aligned} \tag{36}$$

We can find its polyphase components from (44)–(47) as

$$\begin{aligned} &H'_0(z) \\ &= \frac{1 + 1.4921z^{-1} + 0.2219z^{-2} + 0.0548z^{-3}}{1 + 0.8100z^{-1} + 0.8346z^{-2} + 0.1446z^{-3} + 0.1171z^{-4}} \\ &= \frac{1 + 1.3585z^{-1}}{1 + 0.8099z^{-1} + 0.6561z^{-2}} \\ &\quad \times \left(1 + z^{-1}\frac{0.1336 - 0.1382z^{-1}}{1 + 0.0001z^{-1} + 0.1785z^{-2}}\right) \\ &H'_1(z) \\ &= \frac{1.4192 + 0.5920z^{-1} + 0.0431z^{-2}}{1 + 0.8100z^{-1} + 0.8346z^{-2} + 0.1446z^{-3} + 0.1171z^{-4}} \\ &= \frac{1.4192 + 0.4587z^{-1}}{1 + 0.8099z^{-1} + 0.6561z^{-2}} \\ &\quad \times \frac{1 + 0.0940z^{-1}}{1 - 0.0001z^{-1} + 0.1785z^{-2}} \end{aligned}$$

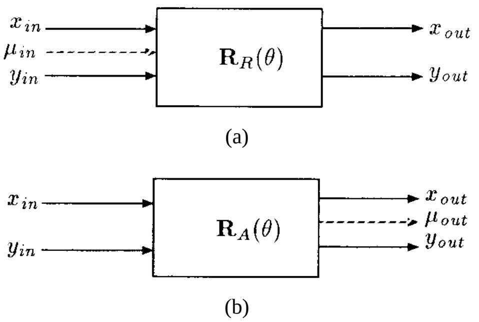


Fig. 14. (a) CORDIC in *vector rotation mode.* (b) CORDIC in *angle accumulation* mode.

$$\begin{aligned}\hat{H}'(z)&\\ &= \frac{2.4192+2.0841z^{-1}+0.2650z^{-2}+0.0548z^{-3}}{1+0.8100z^{-1}+0.8346z^{-2}+0.1446z^{-3}+0.1171z^{-4}}\\ &= \frac{1+0.7562z^{-1}}{1+0.8099z^{-1}+0.6561z^{-2}}\\ &\quad\times\left(2.4192+z^{-1}\frac{0.2543-0.3593z^{-1}}{1+0.0001z^{-1}+0.1785z^{-2}}\right). \qquad (37)\end{aligned}$$

The parameters of each AMRA filter in (37) can be computed from (38)–(43) in Appendix A. The corresponding parameter settings of the programmable modules are listed in Table V(b).

3) Multirate Eight-Point DCT: The rotation parameters θ_i's and the scaling factors $f_{0,i}$'s, $f_{1,i}$'s of the computational modules operating on the even and odd subsequences can be found by using (33) and (34), respectively. The settings are given in Table V(c).

V. Incorporation of the QRD-LSL Adaptive Filtering

In this section, we discuss how to incorporate the feature of adaptive filtering into the computing engine design. We will show that, with little modification of the programmable module design, the proposed system can also perform the QRD-LSL algorithm [7].

A. CORDIC Operation and QRD-LSL Architecture

The COordinate Rotation DIgital Computer (CORDIC) processor is capable of performing various rotation functions based on a sequence of shift-and-add operations [9]. There are two operating modes in the CORDIC processor: One is the *vector rotation* mode [see Fig. 14(a)] which rotates the two-input vector for a given angle θ. Let W be the total iteration number in CORDIC algorithm. In practical implementation, the rotation is performed by feeding a sequence of $\pm 1, \mu_i, i = 0, 1, \cdots, W$, to the CORDIC processor. The other operating mode of the CORDIC processor is the *angle accumulation* mode [see Fig. 14(b)]. The CORDIC processor rotates the input vector until one of input components is annihilated. Meanwhile, the μ_{out} sequence that reflects the performed rotation is generated. In the applications of RLS adaptive filtering, both modes are used for the updating of RLS parameters.

The QRD-LSL algorithm is one of the most promising candidates for the implementation of RLS adaptive filtering. Fig. 15(a) shows the overall architecture to perform linear prediction. The PE's (angle computers and rotators) are running in parallel, and forward/backward prediction errors $f(n)$ and $b(n)$ are obtained in a fully pipelined fashion without any feedback paths. The readers may refer to [7] and [24, Ch. 18] for detailed operations. The QRD-LSL can be implemented using the CORDIC processors by replacing the angle computer with CORDIC in angle accumulation mode $(R_A(\theta))$, and the rotator is replaced with CORDIC in vector rotation mode $(R_R(\theta))$. The resulting system is shown in Fig. 15(b), where the dashed lines denote the data paths of the μ_{in} and μ_{out} sequences. The μ_{out} sequences are first computed by the $R_A(\theta)$'s using the forward and backward signals at each stage. Later the generated μ_{out} sequences are sent to $R_R(\theta)$'s to rotate the signals at next stage.

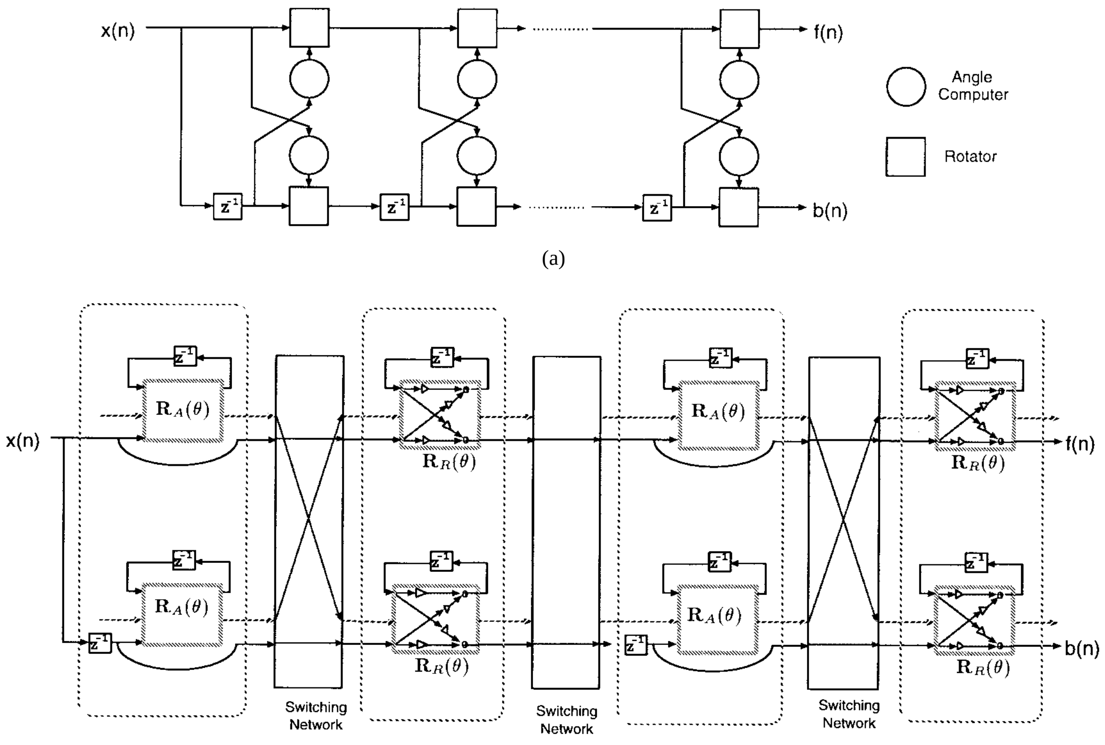
B. Mapping QRD-LSL to the DSP Computing Engine

From Fig. 15, we observe that the basic modules used in QRD-LSL are very similar to the unified programmable module in Fig. 7. Also, the connection of the modules can be easily handled by the interconnection network. We thus modify the programmable module by adding one direct path and one more switch to select this new direct path. One input port for μ_{in} and one output port for μ_{out} are also added for the propagation of the rotation parameters (see Fig. 16). Now based on this modified programmable module, we can implement the QRD-LSL in a very straightforward way. Fig. 17 shows the mapping of a fourth-order QRD-LSL based on our DSP computing engine. The detailed settings of the module array and the interconnection network can be found in Tables II and III.

VI. Performance and Comparison

Consider the programmable modules in Figs. 7 and 16. We can use either the general-purpose multipliers or the CORDIC processor to implement the rotation circuit. In the former case, the critical path is the path along two multipliers and two adders in the middle of the programmable module. Hence, the data throughput rate is approximately $1/2 \cdot T_{\text{MAC}}$, where T_{MAC} denotes the processing time of a MAC operation. If we use the CORDIC processor as the processing kernel of the programmable module, the data throughput rate is limited by the CORDIC processor and the scaling multipliers. It can be approximated by $1/T_{\text{CORDIC}} + T_{\text{MAC}}$, where T_{CORDIC} denotes the total processing time to finish one CORDIC operation.

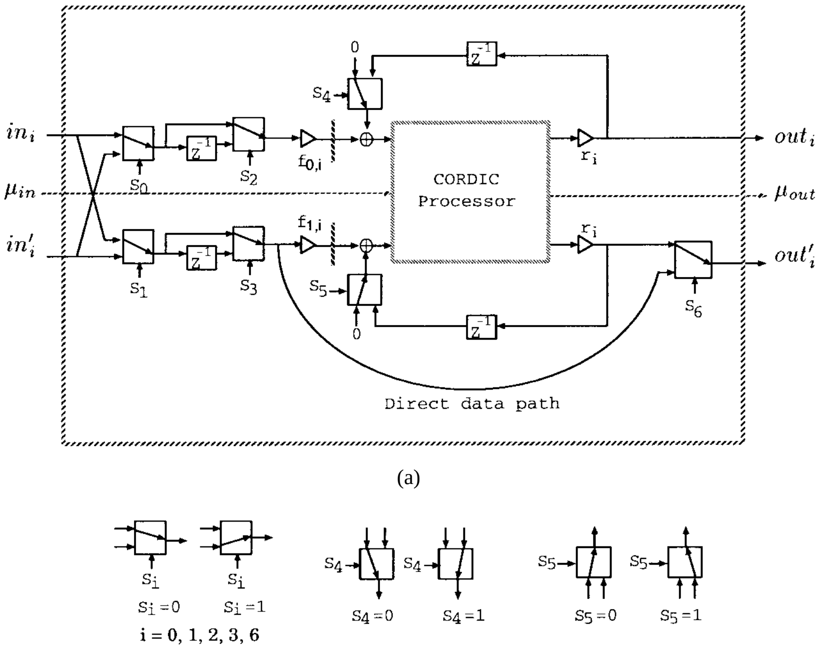
The speed performance of the proposed design can be judged by the following examples. To perform an Nth-order FIR filtering, the general-purpose programmable DSP processor requires $N \cdot T_{\text{MAC}}$ processing time for each serial input data. As a result, the data throughput rate will be degraded as N increases. On the contrary, our computing engine performs the FIR filtering at a fixed data rate of $1/2{\cdot}T_{\text{MAC}}$ (multiplier implementation) or $1/T_{\text{MAC}}{+}T_{\text{CORDIC}}$ (CORDIC implementation) for any $N \leq P$, where P is the total number of available programmable modules in the system. Moreover, the processing rate can be doubled in the



(a)

(b)

Fig. 15. (a) QRD-LSL adaptive filter structure in [7]. (b) Realizing the QRD-LSL using the CORDIC processors.



(a)

(b)

Fig. 16. (a) Modified programmable module with the QRD-LSL feature. (b) Switches used in the module.

multirate mode for any $N \leq 2P/3$, which is comparable to the speed performance of fully pipelined FIR ASIC designs. Another example is the computation of the DCT. Suppose that the fast DCT algorithm in [25] is employed to realize the transform function using a general-purpose programmable DSP processor. It takes approximately $N \cdot \log_2 N/2$ MAC operations to compute an N-point DCT; i.e., the averaged processing time of each serial input data is $\log_2 N/2 \cdot T_{\text{MAC}}$, which is a function of the block size N. Instead, our computing engine performs the time-recursive DCT at a fixed data rate of $1/2 \cdot T_{\text{MAC}}$. We can also perform the multirate DCT to double the processing speed by using twice as many programmable modules. The ASIC-like speed performance of the proposed design is due to the fact that we fully exploit the parallelism of pipelinability of each programmed DSP function at the algorithmic/architectural level.

TABLE VI
COMPARISON OF THE PROPOSED DSP COMPUTING ENGINE DESIGN WITH GENERAL-PURPOSE DSP PROCESSOR AND ASIC DESIGNS

	General-purpose programmable DSP processor	The proposed DSP computing engine design	ASIC designs
Programmed to perform various DSP functions?	Yes	Yes (programmed to perform FIR, QMF, IIR, DT and QRD-LSL).	Usually designed for specific functions.
The way to change programmed feature	Change firmware program.	Set module parameters and configurate interconnection network.	Not applicable
Scalable design?	No	Yes (higher-order FIR/QMF/IIR/DT/QRD-LSL can be achieved by invoking more programmable modules. The number of modules is proportional to the order of those DSP functions.)	Design dependent
Fully exploit the parallelism and pipelinability of each programmed function?	Not applicable for parallel and pipelined implementation.	Yes	Design dependent.
Data throughput rate to perform Nth-order FIR/QMF/IIR filtering	$\frac{1}{N \cdot T_{MAC}}$ (function of order N)	$\frac{1}{2 \cdot T_{MAC}}$ (fixed)	$\frac{1}{2 \cdot T_{MAC}}$ (fixed) for fully-pipelined design.
Data throughput rate to perform N-point DCT	$\frac{2}{\log_2 N \cdot T_{MAC}}$ (function of block size N)	$\frac{1}{2 \cdot T_{MAC}}$ (fixed)	$\frac{1}{2 \cdot T_{MAC}}$ (fixed) for fully-pipelined design.
Multirate data processing capability for high-speed applications?	Not applicable	Yes (only at the expense of invoking more programmable modules).	Yes (at the expense of using more silicon area. See the FIR design in [14] as an example).

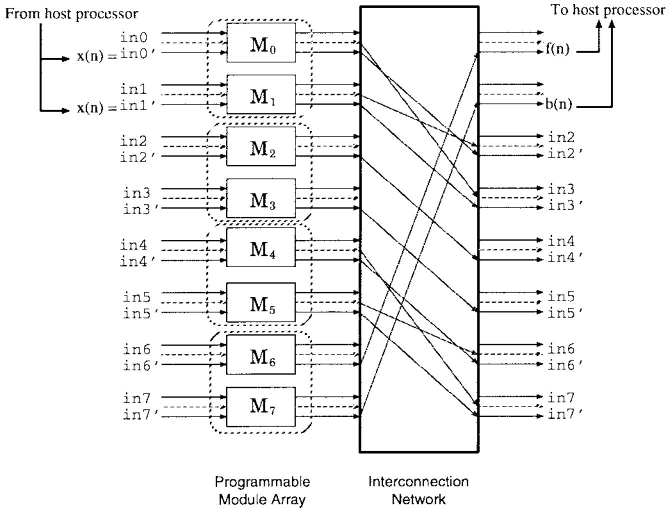


Fig. 17. Realizing the QRD-LSL using the DSP computing engine.

In addition to the speed performance, our design also has good flexibility in modifying the programmed functions in the FIR/IIR/QMF/DT/QRD-RLS. As an example, we can easily increase the order/block-size of those programmed functions at the expense of invoking more programmable modules while retaining the same system throughput rate. Although the general-purpose programmable DSP processor also has the flexibility of modifying the programmed function, the system throughput rate will be degraded as the order/block-size N increases. Besides, we can easily change function specifications (e.g., FIR tap coefficients) by reprogramming the parameters of the programmable modules, which is in general not the case for ASIC designs. The programmable feature can significantly reduce the hardware cost compared with ASIC-based implementations.

To summarize the above arguments, a brief comparison of the proposed system with general-purpose programmable DSP processor and ASIC designs is listed in Table VI. As we can see, our DSP computing engine design possesses the advantages of the other two approaches such as programmability, scalable design, and high data throughput rate. The real-time processing speed as well as the programmable feature of this computing engine makes it very attractive for video-rate DSP applications.

VII. CONCLUSIONS

In this paper, a system architecture of an adaptive reconfigurable DSP computing engine for numerically intensive front-end data processing is presented. It can adaptively perform various important DSP functions such as FIR, QMF, IIR, DT, and QRD-LSL for the host processor by simply loading the suitable parameters and reconfiguring the interconnection network. The proposed parallel architecture retains the advantage of the ASIC designs but is much more flexible. Moreover, the architecture is regular and modular, which makes it very suitable for VLSI implementation. We also showed that we can reconfigure the proposed computing engine to perform the multirate FIR/IIR/DT operations. The significance of this feature is that we can speed up the speed performance of the computing engine by two at the algorithmic/architectural level. Neither expensive high-speed dedicated circuits nor advanced VLSI technologies are used.

The focus of this work is on the "system architecture" aspect which considers how to map a set of important DSP algorithms onto the proposed CORDIC-based parallel system architecture. It is noteworthy that when we map many algorithms altogether, optimization of individual DSP algorithm/structure has to be sacrificed in some sense. For example, although the derived rotation-based IIR lattice structure is not as efficient as its direct form counterparts in terms of speed and complexity, it can easily fit into our rotation-based design. We believe inefficiency of this particular IIR as well as other programmed DSP functions is not the point. The issue is whether the system architecture is efficient enough to handle so many different DSP algorithms under the same framework.

As to the implementation of the proposed system, the designers can implement the whole system using dedicated hardware and/or chip solutions for speed-demanding DSP systems. It is also possible to apply field programmable gate array (FPGA) to realize the design. With the reconfigurable feature of the proposed design and the flexibility of FPGA, it becomes a very good candidate to serve as the DSP processing core in an FPGA-based rapid prototyping system. Furthermore, since the DCT-based motion estimation (ME) scheme in [26] employs DCT/DST as a basic processing kernel, and the computations are inherently local operations, we can also map the DCT-based ME scheme to our design so as to perform the function of motion estimation in video applications.

Appendix A
Conversion of Parameters for the Second-Order IIR Lattice Filter

For $i = 0, 1, \cdots, N/2 - 1$,

1) Find the poles of $H_i(z)$

$$p_{0,i} = \frac{-a_i + \sqrt{a_i^2 - 4b_i}}{2}, \quad p_{1,i} = \frac{-a_i - \sqrt{a_i^2 - 4b_i}}{2}. \tag{38}$$

2)

a) For the case $a_i^2 - 4b_i < 0$ (complex conjugate poles at $r_i e^{\pm \theta_i}$), compute the radius r_i and phase θ_i of the poles

$$r_i = \text{mag}(p_{0,i}), \quad \theta_i = \arg(p_{0,i}). \tag{39}$$

b) For the case $a_i^2 - 4b_i > 0$ (two real poles at $p_{0,i}$ and $p_{1,i}$), compute r_i and θ_i by equating

$$\begin{aligned} 2r_i \cos\theta_i &= p_{0,i} + p_{1,i} \\ r_i^2 &= p_{0,i} \cdot p_{1,i} \end{aligned} \tag{40}$$

which yields

$$r_i = \sqrt{p_{0,i} \cdot p_{1,i}}, \quad \theta_i = \cos^{-1} \frac{p_{0,i} + p_{1,i}}{2\sqrt{p_{0,i} \cdot p_{1,i}}}. \tag{41}$$

3) Solve k_0 and k_1 used in $A_i(z)$ by setting

$$\begin{aligned} r_i(k_0 \cos\theta_i + k_1 \sin\theta_i) &= d_i' \\ -r_i^2 k_0 &= e_i' \end{aligned} \tag{42}$$

which yields

$$k_0 = -e_i'/r_i^2, \quad k_1 = (d_i'/r_i - k_0 \cos\theta_i)/\sin\theta_i. \tag{43}$$

end.

Note that all r_i's should be less than one to ensure the stability of the IIR filtering. There are some limitations in this realization: first, the order of the ARMA filter is restricted to be even to facilitate the decomposition in (12). Second, we cannot realize the second-order IIR which has two multiple real poles or two real poles with opposite signs (r_i in (40) cannot be solved). In some cases, this situation can be avoided by arranging the real poles with the same sign as a pair, or imposing such constraints in the design phase of the filter.

Appendix B
Polyphase Decomposition of an IIR Function

Given the IIR system in (27), we first multiply $(1 + \Sigma_{i=1}^{N} (-1)^i q_i z^{-i})$ in the numerator and denominator of the transfer function. We then have

$$\begin{aligned} H'(z) &= \frac{\left(1 + \sum_{i=1}^{M} p_i z^{-i}\right)\left(1 + \sum_{i=1}^{N} (-1)^i q_i z^{-i}\right)}{1 + \sum_{i=1}^{N} \tilde{q}_i z^{-2i}} \\ &= \frac{N(z)}{D(z^2)} = \frac{N_0(z^2) + z^{-1} N_1(z^2)}{D(z^2)} \end{aligned} \tag{44}$$

where $N_0(z^2)$ and $N_1(z^2)$ are the polyphase components of the new numerator, $N(z)$, and $\tilde{q}_i$'s are given by

$$\tilde{q}_i = \begin{cases} (-1)^i q_i^2 + \sum_{j=0}^{i-1} (-1)^j q_j q_{2i-j}, & \text{if} \quad i < \dfrac{N}{2} \\ (-1)^i q_i^2 + \sum_{j=0}^{N-i-1} (-1)^j q_{N-j} q_{2i-N+j}, & \text{if} \quad i \geq \dfrac{N}{2}. \end{cases} \tag{45}$$

Thus, the polyphase decomposition of $H(z)$ can be written as

$$H'(z) = H_0'(z^2) + z^{-1} H_1'(z^2) \tag{46}$$

with

$$H_0'(z^2) = \frac{N_0(z^2)}{D(z^2)}, \quad H_1'(z^2) = \frac{N_1(z^2)}{D(z^2)}. \tag{47}$$

References

[1] K.-S. Lin *et al.*, "The TMS320 family of digital signal processors," *Proc. IEEE*, vol. 75, pp. 1143–1159, Sept. 1987.

[2] K. Aono *et al.*, "A video digital signal processor with a vector-pipeline architecture," *IEEE J. Solid-State Circuits*, vol. 27, pp. 1886–1893, Dec. 1992.

[3] K. Herrmann *et al.*, "Architecture and VLSI implementation of a RISC core for a monolithic video signal processor," in *VLSI Signal Processing VII*, J. Rabaey, P. M. Chau, and J. Eldon, Eds. Piscataway, NJ: IEEE Press, pp. 368–377, 1994.

[4] P. P. Vaidyanathan and P.-Q. Hoang, “Lattice structures for optimal design and robust implementation of two-channel perfect-reconstruction QMF banks,” *IEEE Trans. Acoust. Speech, Signal Processing*, vol. 36, pp. 81–94, Jan. 1988.

[5] E. Frantzeskakis, J. S. Baras, and K. J. R. Liu, “Time-recursive computation and real-time parallel architectures with application on the modulated lapped transform,” in *Proc. SPIE Advanced Signal Processing Algorithms, Architectures, and Implementations IV*, San Diego, July 1993, pp. 100–111.

[6] ———, “Time-recursive computation and real-time parallel architectures: A framework,” *IEEE Trans. Signal Processing*, vol. 43, pp. 2762–2775, Nov. 1995.

[7] I. K. Proudler, J. G. McWhirter, and T. J. Shepherd, “Computationally efficient QR decomposition approach to least squares adaptive filtering,” in *Proc. Inst. Electron. Eng.–F*, vol. 138, pp. 341–353, Aug. 1991.

[8] H. M. Ahmed, “Alternative arithmetic unit architectures for VLSI digital signal processors,” in *VLSI and Modern Signal Processing*, S. Y. Kung, H. J. Whitehouse, and T. Kailath, Eds. Englewood Cliffs, NJ: Prentice-Hall, ch. 16, pp. 277–303, 1985.

[9] Y. H. Hu, “CORDIC-based VLSI architectures for digital signal processing,” *IEEE Signal Processing Mag.*, vol. 9, pp. 16–35, July 1992.

[10] J.-H. Hsiao, L.-G. Chen, T.-D. Chiueh, and C.-T. Chen, “High throughput CORDIC-based systolic array design for the Discrete Cosine Transform,” *IEEE Trans. Circuits Syst. Video Technol.*, vol. 5, pp. 218–225, June 1995.

[11] S. Molloy *et al.*, “An algorithm-driven processor design for video compression,” in *Proc. IEEE Int. Conf. Image Processing*, Austin, TX, 1994, pp. III.611–615.

[12] A.-Y. Wu and K. J. R. Liu, “A low-power and low-complexity DCT/IDCT VLSI architecture based on Backward Chebyshev Recursion,” in *Proc. IEEE Int. Symp. Circuits and Systems*, London, May 1994, pp. IV.155–158.

[13] ———, “Algorithm-based low-power transform coding architectures,” in *Proc. IEEE Int. Conf. Acoust. Speech, Signal Processing*, Detroit, MI, May 1995, pp. 3267–3270.

[14] ———, “Low-power design methodology for DSP systems using multirate approach,” in *Proc. IEEE Int. Symp. Circuits and Systems*, Atlanta, GA, May 1996, pp. IV.292–295.

[15] L. B. Jackson, *Digital Filters and Signal Processing*, 2nd ed. Norwell, MA: Kluwer, 1989.

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

[17] J. Makhoul, “Linear prediction: A tutorial review,” *Proc. IEEE*, vol. 63, pp. 561–580, Apr. 1975.

[18] K. N. Ngan and W. L. Chooi, “Very low bit rate video coding using 3D subband approach,” *IEEE Trans. Circuits Syst. Video Technol.*, vol. 4, pp. 309–316, June 1994.

[19] D. Taubman and A. Zakhor, “Multirate 3-D subband coding of video,” *IEEE Trans. Image Processing*, vol. 3, pp. 572–588, Sept. 1994.

[20] P. P. Vaidyanathan, *Multirate Systems and Filter Banks*. Englewood Cliffs, NJ: Prentice Hall, 1993.

[21] H. S. Malvar, “Lapped transforms for efficient transform/subband coding,” *IEEE Trans. Acoust. Speech, Signal Processing*, vol. 38, pp. 969–978, June 1990.

[22] Z. J. Mou and D. Duhamel, “Fast FIR filtering: Algorithm and implementations,” *Signal Processing*, vol. 13, pp. 377–384, 1987.

[23] M. Vetterli, “Running FIR and IIR filtering using multirate filter banks,” *IEEE Trans. Acoust. Speech, Signal Processing*, vol. 36, pp. 730–738, May 1988.

[24] S. Haykin, *Adaptive Filter Theory*, 2nd ed. Englewood Cliffs, NJ: Prentice-Hall, 1991.

[25] B. G. Lee, “A new algorithm to compute the discrete cosine transform,” *IEEE Trans. Acoust. Speech, Signal Processing*, vol. 32, pp. 1243–1245, Dec. 1984.

[26] U. V. Koc and K. J. R. Liu, “Discrete-cosine/sine-transform based motion estimation,” in *Proc. IEEE Int. Conf. Image Processing (ICIP-94)*, vol. 3, Austin, TX, pp. 771–775, Nov. 1994.

[27] Z. Wang, “Fast algorithms for the discrete W transform and for the discrete Fourier transform,” *IEEE Trans. Acoust. Speech, Signal Processing*, vol. 32, pp. 803–816, Aug. 1984.

[28] H. S. Malvar, “Extended lapped transforms: Properties, applications, and fast algorithms,” *IEEE Trans. Signal Processing*, vol. 40, pp. 2703–2714, Nov. 1992.

An-Yeu Wu (S’91–M’95) received the B.S. degree from National Taiwan University in 1987 and the M.S. and Ph.D. degrees from the University of Maryland, College Park, in 1992 and 1995, respectively, all in electrical engineering.

During 1987–1989, he served as a Signal Officer in the Army, Taipei, Taiwan, for his mandatory military service. During 1990–1995, he was a Graduate Teaching and Research Assistant with the Department of Electrical Engineering and Institute for Systems Research at the University of Maryland, College Park. From August 1995–July 1995, he was a Member of Technical Staff at AT&T Bell Laboratories, Murray Hill, NJ, working on high-speed transmission IC designs. He is currently an Associate Professor with the Electrical Engineering Department of National Central University, Taiwan. His research interests include adaptive signal processing, multirate signal processing, and low-power/high-performance VLSI architectures for DSP and communication applications.

K. J. Ray Liu (S’86–M’90–SM’93) received the B.S. degree from the National Taiwan University in 1983 and the Ph.D. degree from the University of California, Los Angeles, in 1990, both in electrical engineering.

Since 1990, he has been with Electrical Engineering Department and Institute for Systems Research of University of Maryland at College Park, where he is an Associate Professor. During his sabbatical leave in 1996–97, he was Visiting Associate Professor at Stanford University and Chief Scientist of NeoParadigm Labs. His research interests span all aspects of signal processing with application to image and video, wireless communications, networking, and medical and biomedical technology.

Dr. Liu received numerous awards including the 1994 National Science Foundation Young Investigator Award, the IEEE Signal Processing Society’s 1993 Senior Award (Best Paper Award), the George Corcoran Award in 1994 for outstanding contributions to electrical engineering education, and the 1995–96 Outstanding Systems Engineering Faculty Award in recognition of outstanding contributions in interdisciplinary research, both from the University of Maryland, and many others. He is an Associate Editor of IEEE TRANSACTIONS ON SIGNAL PROCESSING, an Editor of *Journal of VLSI Signal Processing*, a member of Design and Implementation of Signal Processing Systems Technical Committee, and a founding member of Multimedia Signal Processing Technical Committee of IEEE Signal Processing Society.

Arun Raghupathy was born in Madras, India, in 1972. He received the B.Tech. degree in electronics and communications engineering from the Indian Institute of Technology, Madras, in 1993 and the M.S. degree in electrical engineering from the University of Maryland, College Park, in 1995, where he is now working towards the Ph.D. degree.

His research emphasis is on the design of low-power high-performance VLSI signal processing systems. His other interests include development of systems for digital communications and multimedia applications.