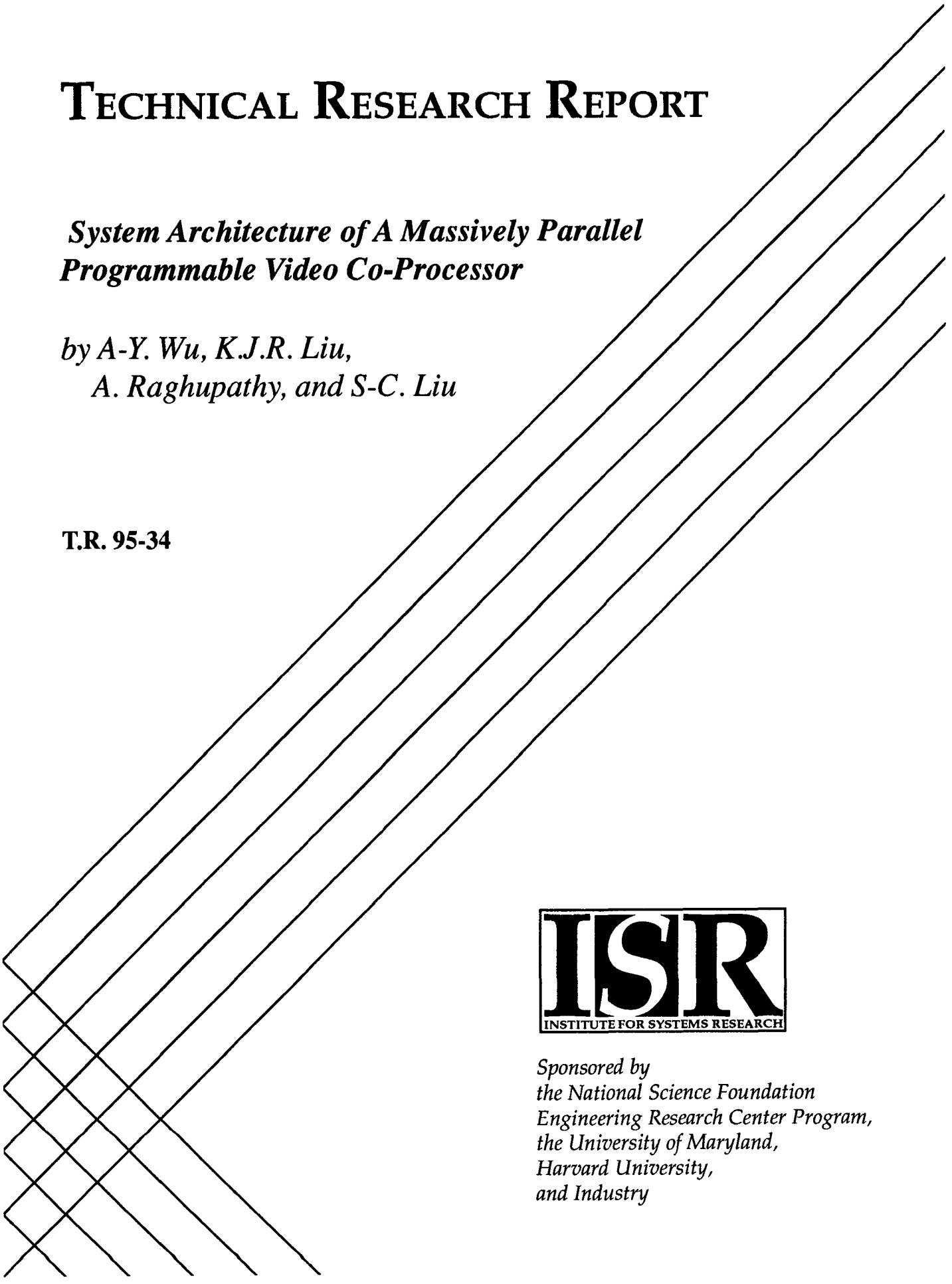
TECHNICAL RESEARCH REPORT

System Architecture of A Massively Parallel Programmable Video Co-Processor

by A-Y. Wu, K.J.R. Liu,
A. Raghupathy, and S-C. Liu

T.R. 95-34



Sponsored by
the National Science Foundation
Engineering Research Center Program,
the University of Maryland,
Harvard University,
and Industry

System Architecture of A Massively Parallel Programmable Video Co-Processor

An-Yeu Wu **K. J. Ray Liu** **Arun Raghupathy** **Shang-Chieh Liu**

Electrical Engineering Department and Institute for Systems Research
University of Maryland
College Park, MD 20742
Phone: (301) 405-6619, Fax: (301) 405-6707

ABSTRACT

Modern video applications call for computationally intensive data processing at very high data rate. In order to meet the high-performance/low-cost constraints, the state-of-art video processor should be a programmable design to perform various tasks in video application whereas the computational power and the manufacturing cost should not be sacrificed for exchange of such flexibility. In this paper, we present a programmable video co-processor design for numerically intensive front-end video/image communications. The resulting system is a massively parallel architecture that is capable of performing most low-level computationally intensive tasks including FIR/IIR filtering, subband filtering, discrete orthogonal transforms (DT), adaptive filtering, and motion estimation, for the host processor. Also, an interconnection network is used to configurate the system for desired data paths. Since the properties of each programmed function such as parallelism and pipelinability have been fully exploited in the design, the computational power of this co-processor is as fast as that of the ASIC designs which are optimized for individual specific applications. We also show that the system can be easily reconfigurated to perform multirate FIR/IIR/DT operations at negligible hardware overhead. Therefore, we can cope with extremely high-speed data by using the same processing elements. This feature can also be applied to the low-power implementation of this co-processor since the multirate operations can "compensate" the increased delay caused by the low supply voltage in the low-power design without hindering the system performance. The programmable/high-speed properties of the proposed co-processor design makes it very suitable for video-rate applications.

This work was supported in part by the ONR grant N00014-93-10566 and the NSF NYI Award MIP9457397.

1 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 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 of basic operations (addition/multiplications) per second to ensure the real-time performance of those video applications. As a consequence, the traditional general-purpose programmable digital signal processing (DSP) processor is not applicable under such speed constraint. Although the performance of the DSP processor can be improved by utilizing advanced VLSI technology and special arithmetic kernels [1][2], the manufacturing cost as well as the complexity of the design will be enormously increased. This approach is referred to as the *technology-driven* solution [3]. On the other hand, dedicated VLSI application-specific integrated circuit (ASIC) chip that is optimally designed for a given function can handle the demanding computational tasks. However, such an approach will also increase the manufacturing cost and system area since a collection of ASIC chips are required for the different tasks in video applications. Hence we try to find a programmable video processor design that has the flexibility of a general DSP processor while it still can meet the stringent speed requirement like the ASIC chips.

Recently, it has been observed that in order to handle the signal processing applications in a very efficient way, the properties of the DSP algorithms should be fully exploited so as to find the parallelism and pipelinability for very high-speed implementations of those algorithms. As an example, the rotation-based CORDIC processor is suggested as a basic module to perform FIR and IIR filtering in a fully pipelined way [4][5]. Also, an inner-product processing kernel was found to be the common part for some signal processing operations [3]. These solutions exploit the inherent properties of the algorithms and are referred to as the *algorithm-driven* solutions [3].

The major goal of this paper is to integrate the rotation-based FIR/IIR architecture, Quadrature Mirror Filter (QMF) lattice structure [6], discrete transform (DT) architecture [7][8], adaptive lattice filter [9], and DCT-based motion estimation scheme [10], into one universal programmable architecture. It will serve as a co-processor in the video system to perform all front-end computationally intensive functions for the host processor. We will first examine the inherent properties of each function, then find the common rotation-based computational modules that can serve as the basic processing elements. Hence, this design is also an *algorithm-driven* solution for video applications but is much more general-purpose. The resulting system consists of an array of identical programmable modules and one programmable interconnection network. Each programmable module will act as the basic computational module in each programmed function by setting suitable parameters and switches. The interconnection network is used to connect the modules and to combine the appropriate module outputs according to the data paths. The proposed architecture is very suitable for VLSI implementation due to its modularity and regularity.

The second goal of this paper is to improve the speed performance of the system based on the multirate approach [11]. In video signal processing, the major constraint is the processing speed of the video processor.

Such speed constraint will result in the use of expensive fast multiplier/adder circuits or full-custom design. Thus, the cost and the design cycle will increase drastically. The multirate approach is a powerful tool for high-speed/low-power DSP applications [12]. 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 processing elements. Thus, we can speed up the system at the algorithmic/architectural level without using expensive high-speed processing elements. We will show that by simply setting new parameters to the programmable modules and reconfigurating the interconnection network, we can speed up the system by performing the multirate operations on-the-fly without modifying the overall architecture.

In the last part, we will show how to incorporate the feature of adaptive filtering into our co-processor design. The recursive least-squares (RLS) filter, which is widely used in channel equalization, system identification, and image restoration, will become another computationally intensive data processing used in portable video equipments since wireless communication requires fast adaptation to the highly non-stationary mobile channel. We will show that, with little modification to the programmable module design, the proposed co-processor can also perform the QR-decomposition based RLS lattice algorithm (QRD-LSL) [9][13, Chap.18] in a fully pipelined way.

The organization of the this paper is as follows: Section 2 presents the basic programmable co-processor design for the FIR, IIR, QMF filtering, and discrete transforms. In Section 3, the speed up of the co-processor design based on the multirate approach is discussed. Later the incorporation of the QRD-LSL array into our co-processor design is presented in Section 4 followed by the conclusions.

2 Video Co-processor Design for the FIR/QMF/IIR/DT

In this section, the design of the video co-processor under normal operation (without speed-up) is discussed. We first examine the the basic operations of the FIR filtering, QMF bank, IIR filtering, and discrete orthogonal transforms (DT), to find the basic computational modules. Later, a universal programmable module which integrates those basic computational modules in FIR/QMF/IIR/DT is derived. We will show that, by setting appropriate parameters to the module and connecting them via a programmable interconnection network, we are able to perform all functions in the FIR/QMF/IIR/DT in a fully-pipelined way.

2.1 Basic Module in FIR

The finite impulse response (FIR) filter is widely used in the areas of image processing and equalization. In addition to the multiply-and-accumulate (MAC) implementation of the filtering operation, an alternative realization of the FIR filter is the lattice structure as shown in Fig.1 [14]. It consists of N basic lattice sections that are connected in a cascade form. The advantages of the lattice structure over the MAC implementation is its robustness to the coefficient quantization effect and the smaller dynamic range due to the orthogonal operation used in each lattice.

Given a 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 [15]:

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

2. For $i = N-1, N-2 \ldots, 0$

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

end

where the parameter k_i's, $i = 0, 1, \ldots, N-1$, are known as the *reflection coefficients*, or the *PARtial CORrelation* coefficients (PARCOR) in the theory of linear prediction [16]. After k_i's are computed, the lattice section of the FIR filter can be described by

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

or equivalently

1. For $|k_i| < 1$,

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

with

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

2. For $|k_i| > 1$,

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

where $sign(k_i)$ denotes the sign of k_i and θ is defined by

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

In general, (3) is preferred for the implementation using general-purpose multipliers. In our unified design, we will employ (4) and (6), which can be implemented by using the CORDIC processor in *hyperbolic mode* [5] together with two scaling multipliers, to realize the basic modules (see Fig.1(a)(b)). Note that we need to swap the two inputs for the case $|k_i| > 1$ since the input vector is inverted in (6). These two basic modules constitute the FIR lattice structure as shown in Fig.1(c).

2.2 Basic Module in QMF

The Quadrature Mirror Filter (QMF) plays a key role in image compression and subband coding [17][18]. Recently, the two-channel paraunitary QMF lattice was proposed [6]. It possesses all the advantages of the lattice structure such as robustness to coefficient quantization, smaller dynamic range, and modularity. Such properties are preferred to the MAC-based realization when the filter bank is implemented using 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. As a consequence, the module in Fig.1(b) can be readily used for the QMF lattice by setting the scaling multiplers equal to one and the CORDIC processor to the *circular mode.*

It has been shown in [6] that every two-channel (real-coefficient, FIR) paraunitary QMF bank can be represented using the QMF lattice. Given a pre-designed power-symmetric FIR analysis filter $H_0(z)$ with unit sample response $h_0(n), n = 0, 1, \ldots, N$, 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 [11, chap.6]:

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

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

end

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.

2.3 Basic Module in IIR

Next we want to consider the basic module for infinite impulse response (IIR) filtering. The lattice structure for an IIR system (all-pole and ARMA) [14] is shown in Fig.3. Although the basic lattice module in IIR 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 no longer exists if we want to implement a general IIR (ARMA) filter (see Fig.3(b)). Therefore, we try to find an IIR module that has similar data path as in the FIR/QMF lattice while retaining the modularity property of the design.

2.3.1 Second-order IIR Lattice Structure

Fig.4 shows the lattice structure that can be used to realize a second-order IIR. It is also known as the "couple-form" of the second-order IIR filter which is robust to the coefficient quantization error under fixed-point arithmetic [15, chap.6]. It can be shown that the transfer functions of the two outputs are given by

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

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

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

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

where K is a scaling constant [1] and each subfilter $H_i(z)$ is of the form

$$\begin{aligned} H_i(z) &= \frac{1 + c_i z^{-1} + d_i z^{-2}}{1 + a_i z^{-1} + b_i z^{-2}} \\ &= \frac{1}{1 + a_i z^{-1} + b_i z^{-2}} + z^{-1} \frac{c_i + d_i z^{-1}}{1 + a_i z^{-1} + b_i z^{-2}} \\ &= A_{i,0}(z) + z^{-1} A_{i,1}(z). \end{aligned} \tag{13}$$

Comparing (10) and (11) with (13), we see that $A_{i,0}(z)$ and $A_{i,1}(z)$ can be realized by either $\tilde{H}_0(z)$ or $\tilde{H}_1(z)$ with appropriate settings of the parameters k_0, k_1, r, and θ. The conversion of the parameters is derived in Appendix, where $\tilde{H}_0(z)$ is chosen for the realization.

Now based on (12) and (13), we can realize $H(z)$ using the signal diagram depicted in Fig.5, in which each stage performs the filtering for $H_i(z)$ and all $A_{i,0}$, $A_{i,1}$, $i = 0, 1, \ldots, N/2 - 1$, are realized by the second-order IIR module in Fig.4.

[1] Here, we have assumed that $K = 1$. This simple scaling operation can be done by the host processor after it collects the outputs from the video co-processor.

2.4 Basic Module in Discrete Transforms

Performing the discrete transforms (DT) based on the rotation circuit has been extensively studied in the literature [5]. Recently, Frantzeskakis *et al.* [7][8] proposed a unified rotation-based architecture for the DT. By exploiting the "shift property" and "periodicity property" of the orthogonal transforms, the authors have shown that any orthogonal transform can be implemented using a time-recursive architecture. In the following, we will first review the time-recursive MLT architecture in [7][8]. Later, we will show how this MLT architecture can be used to realize most of the orthogonal transforms.

2.4.1 Time-Recursive MLT Architecture

The Modulated Lapped Transform (MLT) is defined as [19]:

$$X_{MLT}(k) = c_k\sqrt{\frac{2}{N}}\sum_{n=0}^{2N-1}\sin\frac{\pi}{2N}(n+\frac{1}{2})\cos[\frac{\pi}{N}(k+\frac{1}{2})(n+\frac{1}{2}+\frac{N}{2})]\,x(n) \tag{14}$$

for $k = 0, 1, \ldots, N-1$, where $c_k = (-1)^{(k+2)/2}$ if k is even, and $c_k = (-1)^{(k-1)/2}$ if k is odd. After some algebraic manipulations, the MLT can be decomposed into

$$X_{MLT}(k) = -c_k[\,X_C(k+1) + X_S(k)\,] \tag{15}$$

where

$$X_C(k) \triangleq \beta\sum_{n=0}^{L-1}\cos[(2n+1)\omega_k + \eta_k]x(n), \tag{16}$$

$$X_S(k) \triangleq \beta\sum_{n=0}^{L-1}\sin[(2n+1)\omega_k + \eta_k]x(n), \tag{17}$$

with block size $L = 2N$ and

$$\beta \triangleq \frac{1}{\sqrt{2N}}, \quad \omega_k \triangleq \frac{\pi k}{2N}, \text{ and } \eta_k \triangleq \frac{\pi}{2}(k+\frac{1}{2}). \tag{18}$$

(15) describes how the two functions, $X_C(k)$ and $X_S(k)$, are combined together to obtain the MLT coefficients. In the following discussions, we shall refer to (15) as the *combination function.*

In [7][8], a rotation-based module was derived for the dual generation of $X_C(k)$ and $X_S(k)$ as depicted in Fig.6(a), where the scaling multipliers and the rotation operation are given by

$$\mathbf{f}_k = \begin{bmatrix} f_{0,k} \\ f_{1,k} \end{bmatrix} = \begin{bmatrix} \beta\cos((2L+1)\omega_k + \eta_k) \\ \beta\sin((2L+1)\omega_k + \eta_k) \end{bmatrix} \tag{19}$$

and

$$\mathbf{R}_k = \begin{bmatrix} \cos\theta_k & \sin\theta_k \\ -\sin\theta_k & \cos\theta_k \end{bmatrix} = \begin{bmatrix} \cos(2\omega_k) & \sin(2\omega_k) \\ -\sin(2\omega_k) & \cos(2\omega_k) \end{bmatrix}, \tag{20}$$

respectively. The rotation-based 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 (16) and (17) are available at the module outputs.

The aforementioned module can be used as a basic building block to implement MLT based on (15). Fig.6(b) illustrates the overall time-recursive MLT architecture for the case $N = 8$. It consists of two parts: One is the *module array* which computes $X_C(k)$ and $X_S(k)$, $k = 0, 1, \ldots, N-1$, in parallel. The other is the *interconnection network* which selects and combines the array outputs to generate the MLT coefficients according to the combination function defined in (15).

In [12], it has been observed that the MLT architecture in Fig.6 can realize most existing discrete sinusoidal transforms by setting appropriate parameters and defining the combination functions. For example, $X_C(k)$ in (16) is equivalent to the DCT by setting

$$L = N, \quad \beta = \tilde{c}_k, \quad \omega_k = \frac{k\pi}{2N}, \text{ and } \eta_k = 0, \tag{21}$$

where

$$\tilde{c}_k = \begin{cases} \sqrt{\frac{1}{N}}, & \text{if } k = 0 \\ \sqrt{\frac{2}{N}}, & \text{otherwise.} \end{cases} \tag{22}$$

is the scaling factor of the DCT/IDCT. As a result, the MLT module array in Fig.6 can compute the DCT in parallel. The other example is the DFT with real-valued inputs. With the following parameter setting

$$L = N, \quad \beta = \frac{1}{\sqrt{N}}, \quad \omega_k = \frac{-k\pi}{N}, \text{ and } \eta_k = -\omega_k, \tag{23}$$

(16) and (17) become

$$X_C(k) = \frac{1}{\sqrt{N}} \sum_{n=0}^{N-1} \cos(\frac{-2\pi}{N} kn)\, x(n), \quad X_S(k) = \frac{1}{\sqrt{N}} \sum_{n=0}^{N-1} \sin(\frac{-2\pi}{N} kn)\, x(n), \tag{24}$$

which are the real part and the imaginary part of the DFT, respectively. The DHT can be computed using the same parameter setting as the DFT except that the interconnection network in Fig.6(b) performs as

$$X_{DHT}(k) = X_C(k) + X_S(k). \tag{25}$$

The parameter settings as well as the corresponding combination functions for most DT are summarized in Table 1.

2.5 Unified Module Design

From Fig.1, Fig.2, Fig.4, and Fig.6, we observe that all the architectures share the common computational module with only some minor differences in the data path, the module parameters (multiplier coefficients and rotation angle), and the way the modules are connected. We are thus motivated to integrate those basic modules into one universal programmable module.

Fig.7 shows the proposed programmable module. It consists of six switches, four scaling multipliers and one rotation circuit. The switch set $\mathbf{S} \triangleq [s_0 s_1 s_2 s_3 s_4 s_5]$ controls the data path inside the module. The switch pair s_0 and s_1 select the input from either in_i or in_i': With $s_0 s_1 = 00$, in_i becomes the common input of the lattice which is required in the first stage of FIR and in the IIR module. Using $s_0 s_1 = 10$, we can swap the inputs for the FIR lattice when $|k_i| > 1$. Switches s_2 and s_3 decide if the delay element is used or not: With $s_2 s_3 = 01$, 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 $s_2 s_3 = 11$, the delay element in Fig.5 can be incorporated into the module $A_{i,1}$. Therefore, we do not need to implement it explicitly 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 will be added with the current inputs as in the IIR and DT case. The setting $s_4 s_5 = 00$ will disconnect the feedback paths. The scaling multipliers and the rotation circuit are commonly used in all basic modules of the FIR/QMF/IIR/DT. The parameters $\mathbf{f}_i$ and θ_i can be determined from our discussions in Section 2.1-Section 2.4. The two extra multipliers r_i's at the outputs of the rotation circuit are required if we want to incorporate IIR filtering function into this universal design. The complete settings of the programmable module for the FIR/QMF/IIR/DT are listed in Table 2.

2.6 Video Co-processor Design

Based on the above programmable module, we are ready to design the video co-processor that is capable of performing parallel implementation for any function in the FIR/QMF/IIR/DT. Fig.8 shows the video co-processor architecture under the FIR mode. It consists of two parts: One is the *programmable module array* with P identical programmable modules. The other is the *programmable interconnection network* to connect those programmable modules according to the data paths. In the FIR/QMF/IIR, 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. The connections of IIR modules can be also achieved using the network setting in Fig.9. On the other hand, the DT architecture in Section 2.4 performs the block transforms in a SIPO way. The interconnection network will be configurated according to the combination functions defined in Table 1. The detailed settings of the interconnection network used in this paper (Type I-IX) are described in Table 3.

The operation of the co-processor is as follows: In the *initialization mode*, the host processor will compute all the necessary parameters $\mathbf{f}_i, r_i, \theta_i$ according to the function type (FIR/QMF/IIR/DT) and the function to be performed. In general, the functions to be performed are determined beforehand. Hence, 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 reconfigurate the interconnection network according to the function type.

Once the video co-processor is initialized, it enters the *execution mode*. In the applications of FIR/IIR/QMF, the host processor continuously feeds the data sequence into the co-processor. After the first output data is ready, the processor can collect the filtering outputs in a fully pipelined way. In the block DT application, the block input data is fed into the co-processor serially. After the last datum enters the unified module array, the evaluations of $X_C(k)$ and $X_S(k)$ in (16) and (17) are complete. Then the interconnection network will combine the module outputs according to the combination function defined in Table 1, 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.

The real-time processing speed as well as the programmable feature of this design makes it very attractive for video-rate applications. The programmable feature can significantly reduce the hardware cost compared to the ASIC-based implementations. Meanwhile, we do not trade the processing speed for this flexibility since all operations are performed in a parallel and fully-pipelined way as in the ASIC implementation. Moreover, the resulting system is modular and regular, hence are suitable for VLSI implementation.

2.7 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 decomposition. Here, a 10-module co-processor is used to carry out the given function. As a result, the maximum order of the FIR/IIR/QMF is 10 and the transform size of the DT is also limited to 10. For the DT, we will use an 8-point DCT as an example due to its prevalence in the application of transform coding.

2.7.1 FIR Filtering

Given the FIR transfer function

$$\begin{aligned} H(z) &= 1 - 0.8843z^{-1} - 0.1327z^{-2} - 1.1219z^{-3} + 0.5328z^{-4} - 0.8882z^{-5} \\ &+ 0.1038z^{-6} - 0.3786z^{-7} + 0.2195z^{-8} - 0.1094z^{-9}, \end{aligned} \tag{26}$$

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

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

Then all parameters of each module, such as $\mathbf{f}_i$ and θ_i, can be found by using (4)-(7). The complete settings are listed in Table 4(a).

2.7.2 QMF Filtering

Suppose that the pre-designed power-symmetric low-pass filter described in [11, Example 5.3.2] is used for the QMF filtering. We have the analysis filter

$$H_0(z) = \sum_{n=0}^{N-1} h_0(n) z^{-n} \tag{27}$$

with

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

We can go through (8)-(9) to find all θ_i's in the modules and the results are shown in Table 4(b).

2.7.3 IIR (ARMA) Filtering

Given the IIR (ARMA) filter

$$H(z) = \frac{1 + \sum_{i=1}^{M} p_i z^{-i}}{1 + \sum_{i=1}^{N} q_i z^{-i}} \tag{28}$$

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

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

we first rewrite it in cascade form:

$$\begin{aligned} H(z) &= \frac{1 - 1.1314z^{-1} + 0.6400z^{-2}}{1 - 0.9192z^{-1} + 0.4225z^{-2}} \times \frac{1 - 0.6000z^{-1} + 0.3600z^{-2}}{1 - 0.7500z^{-1} + 0.5625z^{-2}} \\ &\times \frac{1}{1 + 0.8000z^{-1} + 0.6400z^{-2}} \times \frac{1}{1 + 1.2728z^{-1} + 0.8100z^{-2}} \\ &\times \frac{1}{1 + 0.6400z^{-2}}. \end{aligned} \tag{29}$$

Following the steps described in (49)-(56), we can find the parameters used in each 2nd-order subfilter. The corresponding settings are in Table 4(c).

2.7.4 Block DCT

For the block 8-point DCT, we can calculate $f_{0,i}$, $f_{1,i}$ and θ_i for each module using (19)-(22). The settings are listed in Table 4(d).

3 Speed-Up of the Video Co-processor Architecture

In video signal processing, the fundamental bottleneck is the processing speed of the processing elements. Although the above mentioned co-processor architecture has fully exploited the parallelism and pipelinability for each programmed function, the input data rate is still limited by the speed of the adders and multipliers inside the programmable module. In the video-rate applications such as HDTV, this speed constraint will result in the use of expensive high-speed multiplier/adder circuits or full-custom design. Thus, the cost as well as the design cycle will increase drastically.

Recently, the multirate FIR/IIR filtering architecture has been proposed [20][21]. Fig.10 shows the multirate architecture to realize a given function $H(z)$, where $H_0(z)$, $H_1(z)$ are the polyphase components 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 very high-speed filtering operation. For example, it can process data at 100 MHz rate while only 50 MHz processing elements are required. Thus, the aforementioned speed constraint can be resolved at the algorithmic/architectural level by trading some hardware overhead or chip area. In this section, we will show that our video co-processor can be easily reconfigurated to perform the multirate FIR and IIR (ARMA) filtering. That is, we can speed up the co-processor on the spot. Moreover, the incorporation of the multirate DT architecture in [12] is also considered.

3.1 Multirate FIR Architecture

Given a Nth-order FIR filter

$$H(z) = \sum_{i=0}^{N} h_i z^{-i}, \tag{30}$$

its polyphase components are given by [11]

$$H_0(z^2) = \sum_{i=0}^{N/2} h_{i,0} z^{-2i}, \quad H_1(z^2) = \sum_{i=0}^{N/2} h_{i,1} z^{-2i}, \tag{31}$$

with $h_{i,0} = h_{2i}$ and $h_{i,1} = h_{2i+1}$, for $i = 0, 1, \ldots, N/2$. We also have

$$\hat{H}(z^2) = H_0(z^2) + H_1(z^2) = \sum_{i=0}^{N/2} \hat{h}_i z^{-2i} \tag{32}$$

where $\hat{h}_i = h_{i,0} + h_{i,1}$, for $i = 0, 1, \ldots, \frac{N}{2}$ [2]. The implementation of the multirate FIR is as follows: We first implement each $(\frac{N}{2})$th-order FIR subfilter using the FIR lattice structure discussed in Section 2.1. The resulting architecture is depicted in Fig.11(a), where $\tilde{R}_i$, $\hat{R}_i$, and $\bar{R}_i$ correspond to the ith basic modules used in $H_0(z)$, $\hat{H}(z)$, and $H_1(z)$, respectively. Next, we can map Fig.11(a) to our video co-processor with

[2] In Fig.10, the data sample rate is reduced from f_s to $f_s/2$ after the downsampling circuit. As a consequence, the delay z^{-2} at f_s is equivalent to z^{-1} at $f_s/2$, and we use notations $H_0(z), \hat{H}(z), H_1(z)$ instead of $H_0(z^2), \hat{H}(z^2), H_1(z^2)$ in Fig.10.

the mapping

$$\begin{aligned} \tilde{R}_i &\longrightarrow M_{3i}, \\ \hat{R}_i &\longrightarrow M_{3i+1}, \\ \bar{R}_i &\longrightarrow M_{3i+2}, \end{aligned} \tag{33}$$

for $i = 0, 1, \ldots, N/2-1$. Besides, the interconnection network is set to Type II for the connections. Fig.11(b) illustrates the realization of a $6th$-order FIR by the use of 9 programmable modules. The detailed setting of the video co-processor is described in Table 2 and 3.

Once the video co-processor has been initialized, the host processor can send data at f_s rate to the downsampling circuit in Fig.10. Then the outputs of the downsampling circuit, $x_i(n), i = 0, 1, 2$, 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 for $y(n)$ is back to f_s.

As we can see, the only hardware overhead for this multirate operation is the downsampling circuit and upsampling circuit 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 for a Nth-order FIR filter. That is, the system performance is doubled at the expense of 50% hardware overhead. Nevertheless, this overhead is handled by simply activating more modules in the array and reconfigurating the interconnection network rather than implementing it explicitly.

3.2 Multirate IIR Architecture

For the IIR systems, the polyphase decomposition of the transfer function is not as straightforward as in the FIR case. Given an IIR system

$$H(z) = \frac{1 + \sum_{i=1}^{M} p_i z^{-i}}{1 + \sum_{i=1}^{N} q_i z^{-i}} \tag{34}$$

($M \leq N$, M, N are even numbers), we first multiply $(1 + \sum_{i=1}^{N}(-1)^i q_i z^{-i})$ in the numerator and denominator of the transfer function. We then have

$$H(z) = \frac{(1 + \sum_{i=1}^{M} p_i z^{-i})(1 + \sum_{i=1}^{N}(-1)^i q_i z^{-i})}{1 + \sum_{i=1}^{N} \tilde{q}_i z^{-2i}} = \frac{N(z)}{D(z^2)} = \frac{N_0(z^2) + z^{-1} N_1(z^2)}{D(z^2)} \tag{35}$$

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

$$\tilde{q}_i = \begin{cases} (-1)^i q_i^2 + \sum_{j=0}^{i-1}(-1)^j q_j q_{2i-j}, & \text{if} \quad i < \frac{N}{2} \\ (-1)^i q_i^2 + \sum_{j=0}^{N-i-1}(-1)^j q_{N-j} q_{2i-N+j}, & \text{if} \quad i \geq \frac{N}{2}. \end{cases} \tag{36}$$

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

$$H(z) = H_0(z^2) + z^{-1}H_1(z^2) \tag{37}$$

with

$$H_0(z^2) = \frac{N_0(z^2)}{D(z^2)}, \quad H_1(z^2) = \frac{N_1(z^2)}{D(z^2)}. \tag{38}$$

Note that the maximum order of $H_0(z)$ and $H_1(z)$ is still N; *i.e.*, the orders of the subfilters do not decrease after the decomposition. This indicates that the use of Fig.10 will triple the hardware cost. The implementation of the IIR multirate filtering is similar to the FIR case. We first implement each of the subfilters, $H_0(z)$, $H_1(z)$, and $\hat{H}(z) = H_0(z)+H_1(z)$, using the cascade IIR structure discussed in Section 2.3. The corresponding parallel architecture is shown in Fig.12(a), where $\{ \tilde{A}_{i,0}(z), \tilde{A}_{i,1}(z) \}$, $\{ \hat{A}_{i,0}(z), \hat{A}_{i,1}(z) \}$, and $\{ \bar{A}_{i,0}(z), \bar{A}_{i,1}(z) \}$, $i = 0, 1, \ldots, N/2-1$, are the subfilters of $H_0(z)$, $\hat{H}(z)$, $H_1(z)$, respectively. Then it can be mapped to our co-processor architecture by

$$\begin{array}{lll} \tilde{A}_{i,0}(z) \longrightarrow M_{3i}, & \hat{A}_{i,0}(z) \longrightarrow M_{3i+2}, & \bar{A}_{i,0}(z) \longrightarrow M_{3i+4}, \\ \tilde{A}_{i,1}(z) \longrightarrow M_{3i+1}, & \hat{A}_{i,1}(z) \longrightarrow M_{3i+3}, & \bar{A}_{i,1}(z) \longrightarrow M_{3i+5}, \end{array} \tag{39}$$

for $i = 0, 1, \ldots, N/2-1$. Fig.12(b) demonstrates the multirate $4th$-order IIR architecture using 12 programmable modules. The detailed settings of the modules and interconnection network can be found in Table 2 and 3. In general, we will need $3N$ modules to realize a Nth-order multirate IIR filter.

3.3 Multirate Discrete Transform Architecture

The multirate DT architecture was discussed in [12], in which the multirate computation of the DT is applied to low-power VLSI implementation of the transform-coding kernels. Since the multirate DT in [12] is derived based on the second-order IIR in direct form, it is not applicable to the programmable architecture proposed here. We will derive the rotation-based multirate DT architecture so that it can be incorporated into our co-processor design.

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

$$x_e(n) = x(2n), \quad n = 0, 1, \ldots, L/2-1, \tag{40}$$

and the *odd* sequence

$$x_o(n) = x(2n+1), \quad n = 0, 1, \ldots, L/2-1, \tag{41}$$

(16) and (17) can be rewritten as

$$\begin{aligned} X_C(k) &= \beta \sum_{n=0}^{L/2-1} \cos[(4n+1)\omega_k + \eta_k]\, x_e(n) + \beta \sum_{n=0}^{L/2-1} \cos[(4n+3)\omega_k + \eta_k]\, x_o(n) \\ &= X_{C,e}(k) + X_{C,o}(k), \end{aligned} \tag{42}$$

$$\begin{aligned} X_S(k) &= \beta \sum_{n=0}^{L/2-1} \sin[(4n+1)\omega_k + \eta_k]\, x_e(n) + \beta \sum_{n=0}^{L/2-1} \sin[(4n+3)\omega_k + \eta_k]\, x_o(n) \\ &= X_{S,e}(k) + X_{S,o}(k), \end{aligned} \tag{43}$$

for $k = 0, 1, \ldots, N-1$. Following the derivations in [7][8], it can be shown that we can use the rotation-based module in Fig.6(a) for the dual generation of $X_{C,e}(k)$ and $X_{S,e}(k)$ by setting

$$\mathbf{f}_{k,e} = \begin{bmatrix} \hat{f}_{0,k} \\ \hat{f}_{1,k} \end{bmatrix} = \begin{bmatrix} \beta\cos((2L+1)\omega_k + \eta_k) \\ \beta\sin((2L+1)\omega_k + \eta_k) \end{bmatrix}, \qquad \mathbf{R}_{k,e} = \begin{bmatrix} \cos(4\omega_k) & \sin(4\omega_k) \\ -\sin(4\omega_k) & \cos(4\omega_k) \end{bmatrix}. \tag{44}$$

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

$$\mathbf{f}_{k,o} = \begin{bmatrix} \tilde{f}_{0,k} \\ \tilde{f}_{1,k} \end{bmatrix} = \begin{bmatrix} \beta\cos((2L+3)\omega_k + \eta_k) \\ \beta\sin((2L+3)\omega_k + \eta_k) \end{bmatrix}, \qquad \mathbf{R}_{k,o} = \begin{bmatrix} \cos(4\omega_k) & \sin(4\omega_k) \\ -\sin(4\omega_k) & \cos(4\omega_k) \end{bmatrix}. \tag{45}$$

The parallel architecture to realize (42)-(45) is depicted in Fig.13(a). The input data sequence $x(n)$ is first decimated into $x_e(n)$ and $x_o(n)$ through the decimator (the extra delay element on the top is used to synchronize the two output sequences so that they can arrive at the two rotation modules at the same time.). 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 co-processor design by setting the parameters $\mathbf{f}_{k,e}, \mathbf{R}_{k,e}$ to module M_{2k} and $\mathbf{f}_{k,o}, \mathbf{R}_{k,o}$ to M_{2k+1}, for $k = 0, 1, \ldots, N/2-1$. Fig.13 shows the multirate 4-point DHT architecture based on 8 programmable modules. 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 1. In real implementation, we can either add one summation circuit so that the switch settings for the DT in Table 3 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.

3.4 Design Examples Using Multirate Operations

3.4.1 Multirate FIR Filtering

Given the FIR transfer function in (26), we first perform the polyphase decomposition which yields

$$\begin{aligned} H_0(z) &= 1 - 0.1327z^{-1} + 0.5328z^{-2} + 0.1038z^{-3} + 0.2195z^{-4}, \\ H_1(z) &= -0.8843 - 1.1219z^{-1} - 0.8882z^{-2} - 0.3786z^{-3} - 0.1094z^{-4}, \\ \hat{H}(z) &= 0.1157 - 1.2546z^{-1} - 0.3554z^{-2} - 0.2748z^{-3} + 0.1101z^{-4}. \end{aligned} \tag{46}$$

Then we can follow the steps in (1)-(2) and (4)-(7) to find the parameters for each filter in (46). The results are listed in Table 5(a).

3.4.2 Multirate IIR (ARMA) Filtering

Consider the IIR (ARMA) filter shown below

$$H(z) = \frac{1 - 0.4000z^{-1} + 0.1600z^{-2}}{1 - 1.8192z^{-1} + 2.0598z^{-2} - 1.1248z^{-3} + 0.3422z^{-4}}. \quad (47)$$

We can find its polyphase components from (35)-(36) as

$$\begin{aligned} H_0(z) &= \frac{1 + 1.4921z^{-1} + 0.2219z^{-2} + 0.0548z^{-3}}{1 + 0.8100z^{-1} + 0.8346z^{-2} + 0.1446z^{-3} + 0.1171z^{-4}}, \\ H_1(z) &= \frac{1.4192 + 0.5920z^{-1} + 0.0431z^{-2}}{1 + 0.8100z^{-1} + 0.8346z^{-2} + 0.1446z^{-3} + 0.1171z^{-4}}, \\ \hat{H}(z) &= \frac{2.4192 + 2.0841z^{-1} + 0.2650z^{-2} + 0.0548z^{-3}}{1 + 0.8100z^{-1} + 0.8346z^{-2} + 0.1446z^{-3} + 0.1171z^{-4}}. \end{aligned} \quad (48)$$

Then the necessary parameters of each AMRA filter in (48) can be computed from (49)-(56) in Appendix. the parameter settings for the programmable module are in Table 5(b).

3.4.3 Multirate 8-point DCT

The rotation parameters θ_i's and the scaling factors $f_{0,i}$'s, $f_{1,i}$'s for the modules operating on the even and odd subsequences can be found by using (44) and (45), respectively. See settings in Table 5(c).

4 Incorporation of the QRD-LSL Architecture

In the last part, we will incorporate the feature of adaptive filtering into the co-processor design. We will show that, with little modification of the programmable module design, the proposed co-processor can also perform the QR-decomposition based recursive least-squares lattice (QRD-LSL) algorithm [9].

4.1 CORDIC Operation and QRD-LSL Architecture

The CORDIC is capable of evaluating various rotation functions based on the shift-and-add operations [5]. There are two operation modes in the CORDIC processor: one is the *vector rotation* mode (see Fig.14(a)) which will rotate the 2-input vector for a given angle θ. Let W be the total iteration number in CORDIC algorithm. In real implementation, the rotation is performed by feeding a sequence of ± 1, μ_i, $i = 0, 1, \ldots, W$, to the CORDIC processor. Suppose that the rotation circuit inside our programmable module is implemented using the CORDIC processor. The values of μ_{in}'s can be calculated in advance and will be loaded to the module array during the initialization mode. On the other hand, the CORDIC in *angle accumulation* mode (see Fig.14(b)) is to rotate the input vector until one of input components is annihilated; meanwhile, the

μ_{out} sequence that reflect the performed rotation are generated. In the applications of adaptive filtering, we will use this mode for the updating of the RLS parameters.

The QRD-LSL algorithm is one of the most promising candidate for the implementation of the recursive least-square (RLS) adaptive filtering. Fig.15(a) shows the overall architecture to perform the linear prediction. The readers may refer to [9][13, Chap.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 ($R_A(\theta)$), while 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 for the μ sequences. The μ_i sequences will be first computed by the $R_A(\theta)$'s using the forward and backward signals at each stage. Later the generated μ_i sequences will be sent to $R_R(\theta)$'s to rotate the signals at each stage. The CORDIC-based QRD-LSL can be considered as a special case of the CORDIC-based QRD-RLS array discussed in [5].

4.2 Mapping QRD-LSL to the Programmable Video Co-processor

From Fig.15, we observe that the basic modules used in QRD-LSL are very similar to our programmable module. Also, the connections can be easily handled by the interconnection network. We thus modify the programmable module by adding one direct path as well as one more switch for selecting this new path. On the other hand, 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 using the new programmable module, we can implement the QRD-LSL in a very straightforward way. The detailed settings of the module array as well as the connection type can be found in Table 2 and 3. Fig.17 shows the implementation of a 4th-order QRD-LSL based on our programmable co-processor, where the adaptive filtering is performed in a fully-pipelined way without any feedback path.

5 Conclusions

In this paper, a programmable video co-processor design for numerically intensive front-end video/image communications is presented. The proposed parallel architecture can perform various functions (FIR/QMF/IIR/DT/QRD-LSL) for the host processor by simply loading the suitable parameters and reconfigurating the interconnection network. The parallel design as well as the fully-pipelined operation of the co-processor architecture retains all advantages of the ASIC design but is much more flexible. Moreover, the architecture is regular and modular. As a consequence, they are very suitable for VLSI implementation.

We also showed that we can reconfigurate the video co-processor to perform the multirate FIR/IIR/DT with only two additional small circuits for the down- and up-sampling operations. The significance of the feature is twofold: Firstly, we can speed up the performance of the co-processor since the processing elements can now process data that are twice as fast as its processing speed. Secondly, the multirate operation can be applied to the low-power implementation as discussed in [12]. In most low-power VLSI designs, the supply voltage is usually reduced to lower the total power consumption. However, the device speed will be degraded as the supply voltage goes down. In [12], it has been shown that the multirate scheme can "compensate"

the increased delay caused by the low supply voltage without hindering the system performance. Thus, the co-processor can have a switch for the supply voltage. Under normal operation, the supply voltage is 5V. When the job is not computationally demanding, the supply voltage is switched to 3.1V and the co-processor switched to multirate mode. The system will still maintain the processing speed even though each processing element inside the module has been slowed down by the reduced voltage.

Another interesting application is to incorporate the DCT-based motion estimation (ME) scheme [22][10] into our co-processor design. Since the DCT-based ME requires DCT/DST as a basic processing kernel, and the computations is inherently highly local operation, the programmable co-processor can be modified to perform the function of ME.

Appendix

Conversion of Parameters for the second-order IIR Lattice Filter

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

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

$$p_{0,i} = \frac{-a_i + \sqrt{a_i^2 - 4b_i}}{2}, \quad p_{1,i} = \frac{-a_i - \sqrt{a_i^2 - 4b_i}}{2}. \tag{49}$$

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

$$r_i = \text{mag}(p_{0,i}), \quad \theta_i = \arg(p_{0,i}). \tag{50}$$

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

$$\begin{cases} 2r_i \cos\theta_i = p_{0,i} + p_{1,i} \\ r_i^2 = p_{0,i} \cdot p_{1,i}, \end{cases} \tag{51}$$

   which yields

$$\begin{cases} r_i = \sqrt{p_{0,i} \cdot p_{1,i}} \\ \theta_i = \cos^{-1} \frac{p_{0,i}+p_{1,i}}{2\sqrt{p_{0,i} \cdot p_{1,i}}}. \end{cases} \tag{52}$$

3. Solve k_0 and k_1 used in $A_{i,0}(z)$ by setting

$$\begin{cases} r_i(k_0 \cos\theta_i + k_1 \sin\theta_i) = 1 \\ -r_i^2 k_0 = 0 \end{cases} \tag{53}$$

   which yields

$$\begin{cases} k_0 = 0 \\ k_1 = 1/(r_i \sin\theta_i). \end{cases} \tag{54}$$

4. Solve k_0 and k_1 used in $A_{i,1}(z)$ by setting

$$\begin{cases} r_i(k_0 \cos\theta_i + k_1 \sin\theta_i) = c_i \\ -r_i^2 k_0 = d_i \end{cases} \tag{55}$$

which yields

$$\begin{cases} k_0 = -d_i/r_i^2 \\ k_1 = (c_i/r_i - k_0 \cos\theta_i)/\sin\theta_i. \end{cases} \tag{56}$$

end.

All r_i's should be less than one to ensure the stability of the IIR filtering. There are some limitations in this realization: Firstly, we cannot realize the second-order IIR which has two multiple real poles or two real poles with opposite signs (r_i in (51) 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. Secondly, the order of the ARMA filter is restricted to be even to facilitate the decomposition in (12).

References

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

[2] K. Herrmann, M. Seifert, K. Gaedke, H. Jeschke, and P. Pirsch, "Architecture and VLSI implementation of a RISC core for a monolithic video signal processor," in *VLSI Signal Processing Workshop, VII* (J. Rabaey, P. M. Chau, and J. Eldon, eds.), pp. 368–377, San Diego: IEEE Press, Oct. 1994.

[3] S. Molloy, B. Schoner, A. Madisetti, R. Jain, and R. Matic, "An algorithm-driven processor design for video compression," in *Proc. IEEE Int. Conf. Image Processing*, (Austin, Texas), pp. III.611–615, 1994.

[4] 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.), ch. 16, pp. 277–303, Prentice-Hall, 1985.

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

[6] 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.

[7] E. Frantzeskakis, J. S. Baras, and K. J. R. Liu, "Time-recursive computation and real-time parallel architectures, Part I: Framework," Tech. Rep. TR 93-17r1, Institute for Systems Research, University of Maryland, 1993.

[8] E. Frantzeskakis, J. S. Baras, and K. J. R. Liu, "Time-recursive computation, Part II: Methodology, and application on QMF banks and ELT," Tech. Rep. TR 93-18r1, Institute for Systems Research, University of Maryland, 1993.

[9] I. K. Proudler, J. G. McWhirter, and T. J. Shepherd, "Computationally efficient QR decomposition approach to least squares adaptive filtering," *IEE Proceedings-F*, vol. 138, pp. 341–353, Aug. 1991.

[10] U. V. Koc and K. J. R. Liu, "Discrete-cosine/sine-transform based motion estimation," in *Proceedings of IEEE International Conference on Image Processing (ICIP-94)*, vol. 3, (Austin, Texas), pp. 771–775, Nov. 1994.

[11] P. P. Vaidyanathan, *Multirate systems and filter banks.* Englewood Cliffs, NJ: Prentice Hall, 1993.

[12] A.-Y. Wu and K. J. R. Liu, "Algorithm-based low-power transform coding architectures," in *To appear in Proc. IEEE Int. Conf. Acoust. Speech, Signal Processing*, (Detroit), May 1995.

[13] S. Haykin, *Adaptive Filter Theory.* Prentice-Hall, Englewood Cliffs, N.J., 2nd ed., 1991.

[14] L. B. Jackson, *Digital Filters and Signal Processing.* Kluwer Academic Publishers, 2nd ed., 1989.

[15] A. V. Oppenheim and R. W. Schafer, *Discrete-time Signal Processing.* Prentice Hall, 1989.

[16] J. Makhoul, "Linear Prediction: A tutorial review," *Proc. IEEE*, vol. 63, pp. 561–580, April 1975.

[17] 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.

[18] D. Taubman and A. Zakhor, "Multirate 3-D subband coding of video," *IEEE Trans. Image Processing*, vol. 3, Sept. 1994.

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

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

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

[22] U. V. Koc and K. J. R. Liu, "Low-complexity motion estimation scheme utilizing sinusoidal orthogonal principle," in *Proc. IEEE Workshop on Visual Signal Processing and Communications*, (New Brunswick), pp. 57–62, Sept. 1994.

[23] 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.

	L	β	ω_k	η_k	Combination Function
DCT	N	$\tilde{c}_k$	$\frac{k\pi}{2N}$	0	$X_{DCT}(k) = X_C(k)$
IDCT	N	$\tilde{c}_1$	$\frac{\pi}{2N}(k+\frac{1}{2})$	$-\omega_k$	$X_{IDCT}(k) = X_C(k) + (\tilde{c}_0 - \tilde{c}_1)x(0)$
DST-IV in [23]	N	$\tilde{c}_1$	$\frac{\pi}{2N}(k+\frac{1}{2})$	0	$X_{DST}(k) = X_S(k)$
IDST-IV in [23]	N	$\tilde{c}_1$	$\frac{\pi}{2N}(k+\frac{1}{2})$	0	$X_{IDST}(k) = X_S(k)$
MLT	$2N$	$\frac{1}{\sqrt{2N}}$	$\frac{k\pi}{2N}$	$\frac{\pi}{2}(k+\frac{1}{2})$	$X_{MLT}(k) = -c_k[\, X_C(k+1) + X_S(k)\,]$
ELT	$4N$	$\frac{1}{2\sqrt{2N}}$	$\frac{\pi}{2N}(k+\frac{1}{2})$	$\frac{\pi}{2}(k+\frac{1}{2})$	$X_{ELT}(k) = -X_S(k+1) + \sqrt{2}X_C(k) + X_S(k-1)$
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	$\frac{1}{\sqrt{N}}$	$-\frac{k\pi}{N}$	$-\omega_k$	$X_{DHT}(k) = X_C(k) + X_S(k).$

Table 1: Parameter settings for the unified discrete transformation architecture, where $Re\{X_{DFT}(k)\}$ and $Im\{X_{DFT}(k)\}$ denote the the real part and the imaginary part of the DFT, respectively.

	$\mathbf{S} = [s_0 s_1 s_2 s_3 s_4 s_5 s_6]$	$\mathbf{f}_i = [f_{0,i}, f_{1,i}]^T$	$\mathbf{R}_i$	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 ($\lvert k_i\rvert > 1$)	$\begin{cases} 000100 & (M_0) \\ 100100 & (M_i,\ i \neq 0) \end{cases}$	$\begin{bmatrix} -\text{sign}(k_i)\sqrt{k_i^2-1} \\ -\text{sign}(k_i)\sqrt{k_i^2-1} \end{bmatrix}$ 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 ($A_{i,0}$)	000011	$\begin{bmatrix} k_0 \\ k_1 \end{bmatrix}$ defined in (53)	$\begin{bmatrix} r_i\cos\theta_i & r_i\sin\theta_i \\ -r_i\sin\theta_i & r_i\cos\theta_i \end{bmatrix}$, with r_i, θ_i defined in (49), or (51)	Type III	$\frac{2P}{3}$
IIR ($A_{i,1}$)	001111	$\begin{bmatrix} k_0 \\ k_1 \end{bmatrix}$ defined in (55)	$\begin{bmatrix} r_i\cos\theta_i & r_i\sin\theta_i \\ -r_i\sin\theta_i & r_i\cos\theta_i \end{bmatrix}$, with r_i, θ_i defined in (49), or (51)	Type III	$\frac{2P}{3}$
DT	000011	$\begin{bmatrix} \beta\cos((2L+1)\omega_k+\eta_k) \\ \beta\sin((2L+1)\omega_k+\eta_k) \end{bmatrix}$ with L, β, ω_k, η_k defined in Table 1	$\begin{bmatrix} \cos 2\omega_k & \sin 2\omega_k \\ -\sin 2\omega_k & \cos 2\omega_k \end{bmatrix}$, with ω_k defined in Table 1	Type V-VIII	P
Multirate 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 II	P
Multirate FIR ($\lvert k_i\rvert > 1$)	$\begin{cases} 000100 & (M_0) \\ 100100 & (M_i,\ i \neq 0) \end{cases}$	$\begin{bmatrix} -\text{sign}(k_i)\sqrt{k_i^2-1} \\ -\text{sign}(k_i)\sqrt{k_i^2-1} \end{bmatrix}$ 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	P
Multirate IIR ($A_{i,0}$)	000011	$\begin{bmatrix} k_0 \\ k_1 \end{bmatrix}$ defined in (53)	$\begin{bmatrix} r_i\cos\theta_i & r_i\sin\theta_i \\ -r_i\sin\theta_i & r_i\cos\theta_i \end{bmatrix}$, with r_i, θ_i defined in (49), or (51)	Type IV	$\frac{P}{3}$
Multirate IIR ($A_{i,1}$)	001111	$\begin{bmatrix} k_0 \\ k_1 \end{bmatrix}$ defined in (55)	$\begin{bmatrix} r_i\cos\theta_i & r_i\sin\theta_i \\ -r_i\sin\theta_i & r_i\cos\theta_i \end{bmatrix}$, with r_i, θ_i defined in (49), or (51)	Type IV	$\frac{P}{3}$
Multirate DT	000011	$\begin{bmatrix} \beta\cos((2L+2l+1)\omega_k+\eta_k) \\ \beta\cos((2L+2l+1)\omega_k+\eta_k) \end{bmatrix}$ with $l = k \bmod 2$, and L, β, ω_k, η_k defined in Table 1	$\begin{bmatrix} \cos 4\omega_k & \sin 4\omega_k \\ -\sin 4\omega_k & \cos 4\omega_k \end{bmatrix}$, with ω_k defined in Table 1	Type V-VIII	$\frac{P}{2}$
QRD-LSL	$\begin{cases} 1000101 & (M_{4k}) \\ 1001101 & (M_{4k+1}) \\ 1000100 & (M_{4k+2},\ M_{4k+3}) \end{cases}$	$\begin{bmatrix} 1 \\ 1 \end{bmatrix}$	$\begin{cases} R_A(\theta) = M_{4k}, M_{4k+1} \\ R_R(\theta) = M_{4k+2}, M_{4k+3} \end{cases}$	Type IX	$\frac{P}{4}$

Table 2: Setting for the 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.

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 $m = 0, 1, \ldots, (N-4)$ $in_{m+3} = out_m$, $in'_{m+3} = out'_m$. end 3. $y_i(n) = out_{(N-3)+i}$, $i = 0, 1, 2$.	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. $y(n) = out_{N-1} + out_{N-2}$, $i = 0, 1$.
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 $m = 0, 1, \ldots, (N-7)$ $in_{m+6} = out_{2\lfloor \frac{m}{2} \rfloor} + out_{2\lfloor \frac{m}{2} \rfloor + 1}$. end 3. $y_i(n) = out_{(N-5)+i} + out_{(N-6)+i}$, $i = 0, 1, 2$.	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 $m = 0, 1, \ldots, (N-3)$ $in'_{m+2} = out'_m$, if (m mod $4 = 0$) then $\mu_{in}(m+3) = \mu_{out}(m)$, and $\mu_{in}(m+2) = \mu_{out}(m+1)$. end 3. $f(n) = out_{N-2}$, $b(n) = out_{N-1}$.

Table 3: Switch settings for the interconnection network.

	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	000011	001111	000011	001111	000011	001111	000011	001111	000011	001111
$f_{0,i}$	0	-1.5148	0	-0.6400	0	0	0	0	0	0
$f_{1,i}$	-2.1757	0.9467	-1.5396	0.5543	-1.4434	0	-1.5713	0	-1.2500	0
r_i	0.6500	0.6500	0.7500	0.7500	0.8000	0	0.9000	0	0.8000	0
θ_i	-0.7854	-0.7854	-1.0472	-1.0472	-2.0944	0	-2.3562	0	-1.5708	0
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)

Table 4: Settings for the (a) FIR filter, (b) QMF filter, (c) IIR (ARMA) filter, and (d) 8-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	000011	001111	000011	001111	000011	001111	000011	001111	000011	001111	000011	001111
$f_{0,i}$	0	-0.0614	0	-0.1104	0	-0.0657	-7.6102	0	-4.2364	0	0	0
$f_{1,i}$	1.4256	0.1551	3.4488	0.2992	2.0232	0.8060	2.3669	0	2.3669	0	2.3669	0
r_i	0.8100	0.8100	0.8100	0.8100	0.8100	0.8100	0.4225	0.4225	0.4225	0.4225	0.4225	0.4225
θ_i	2.0944	2.0944	2.0944	2.0944	2.0944	2.0944	1.5708	1.5708	1.5708	1.5708	1.5708	1.5708
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)

Table 5: Settings for the (a) FIR filter, (b) IIR (ARMA) filter, and (c) 8-point DCT under multirate operation.

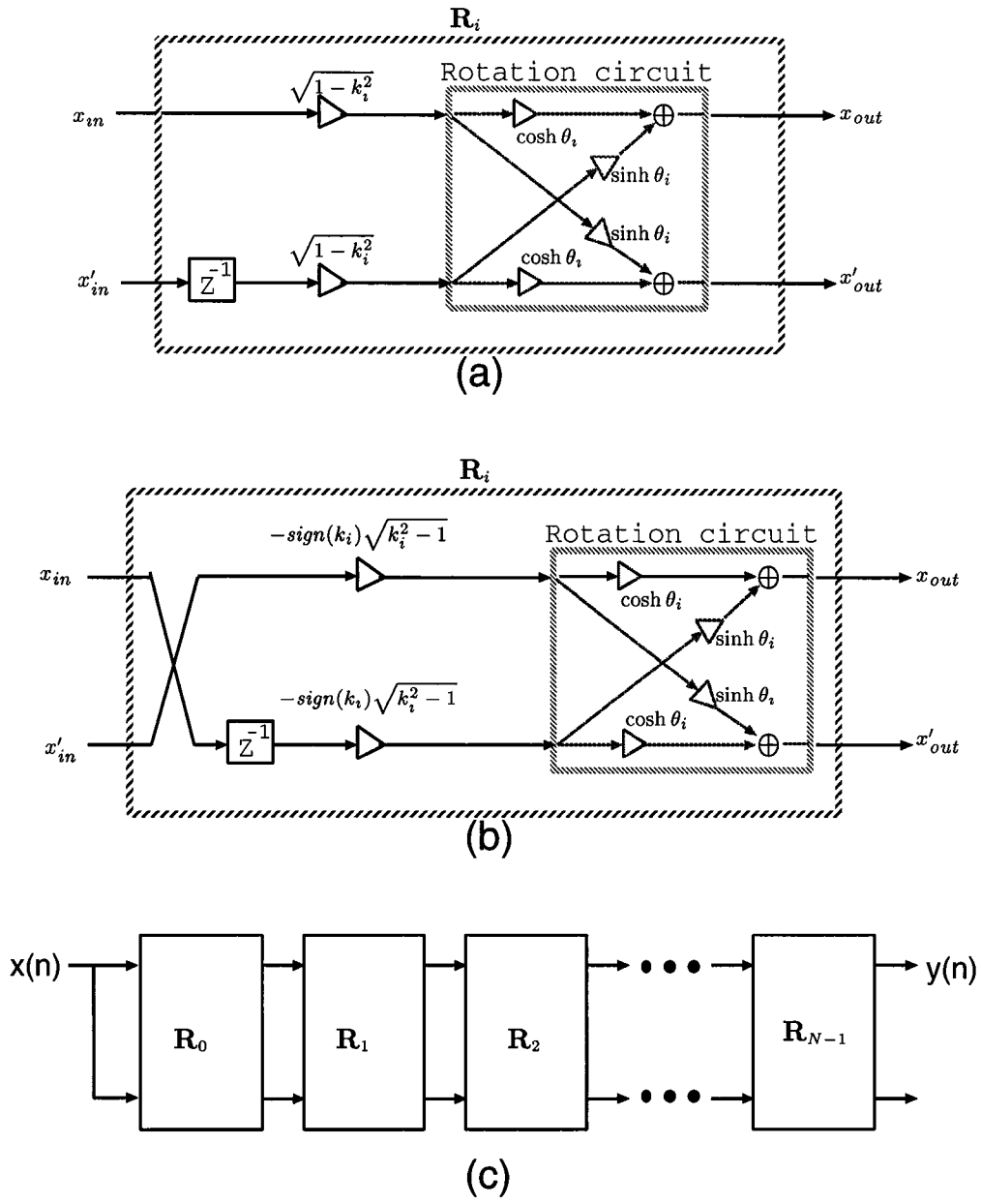


Figure 1: (a) Basic lattice filter section with $|k_i| < 1$. (b) Basic lattice filter section with $|k_i| > 1$. (c) Lattice FIR structure.

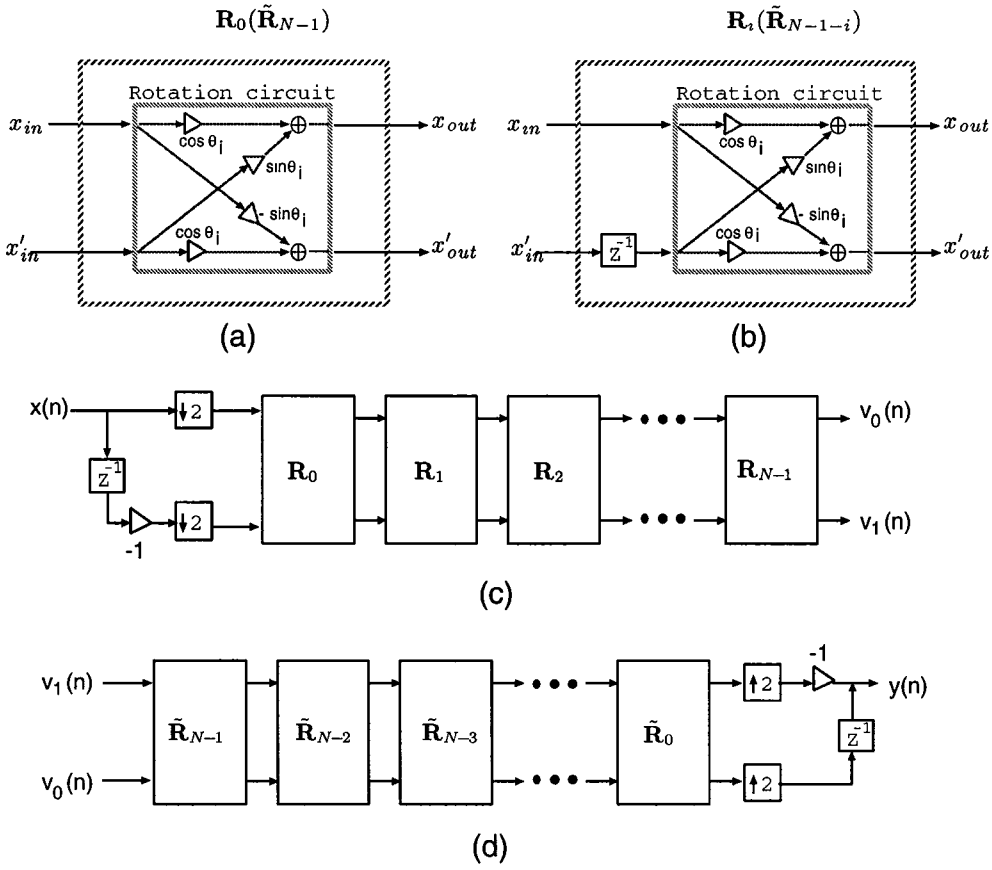


Figure 2: The two-channel paraunitary QMF lattice: (a)(b) Basic lattice section. (c) The analysis bank. (d) The synthesis bank.

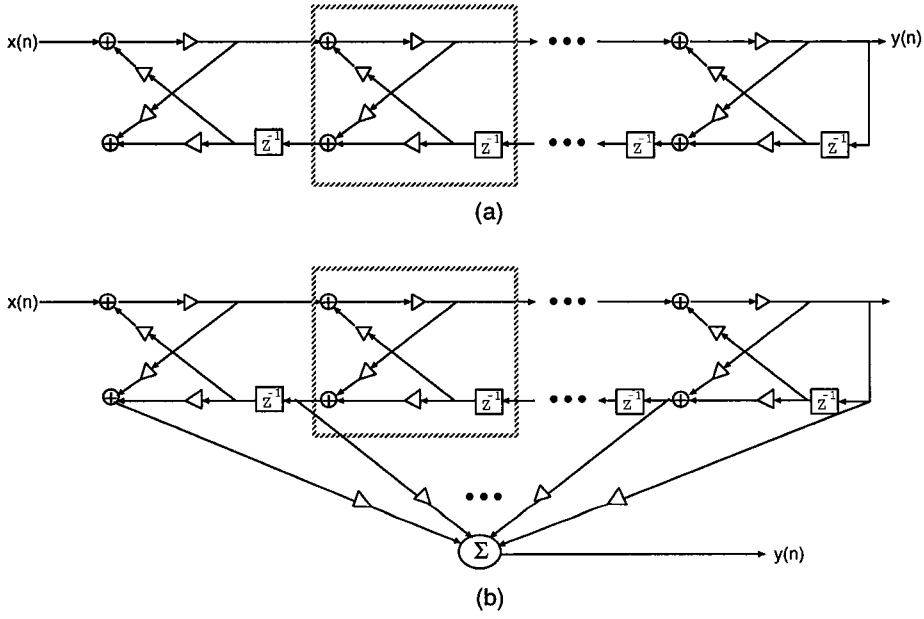


Figure 3: (a) All-pole IIR lattice. (b) General IIR (ARMA) lattice.

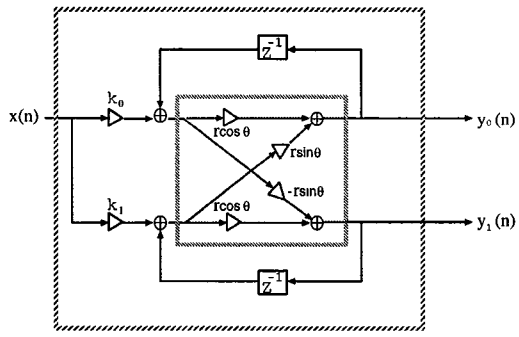


Figure 4: Second-order IIR lattice architecture.

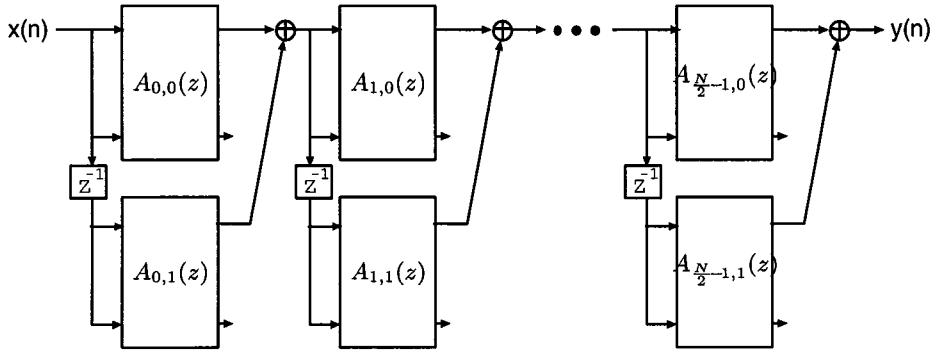


Figure 5: IIR (ARMA) structure based on the second-order IIR lattice module.

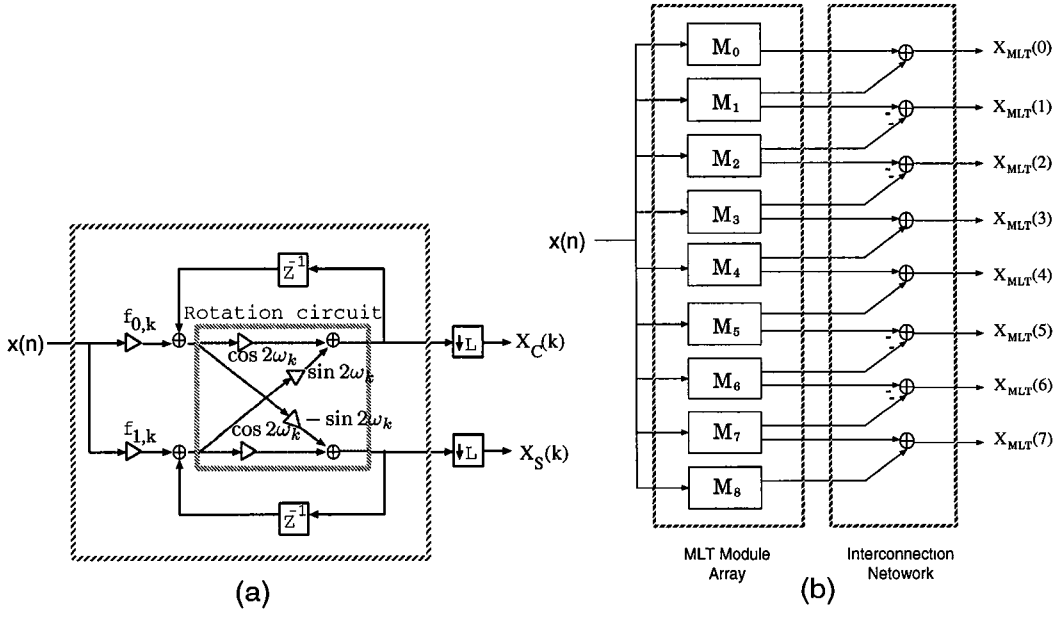


Figure 6: SIPO MLT architecture: (a) Rotation-based module for the dual generation of $X_C(k)$ and $X_S(k)$, where the downsampling operation $\downarrow L$ at the right end denotes that we pick up the values of $X_C(k)$ and $X_S(k)$ at the L^{th} clock cycle and ignore all the previous intermediate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

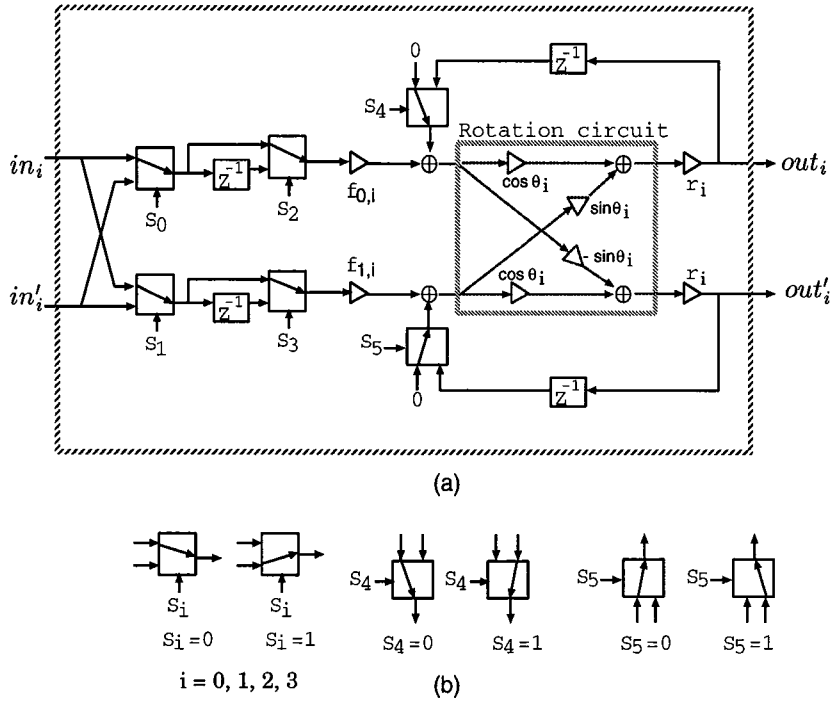


Figure 7: (a) Programmable module for the FIR/QMF/IIR/DT. (b) Switches used in the module.

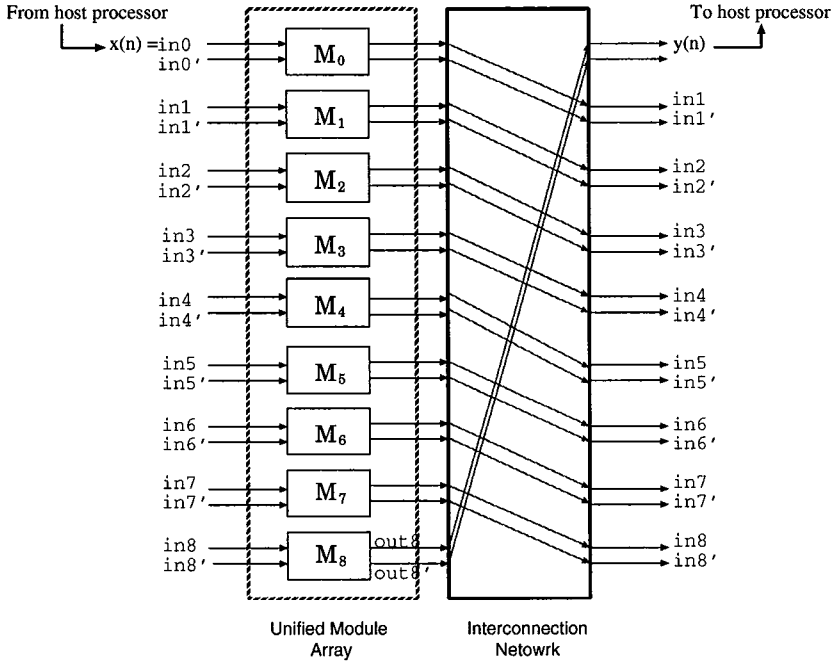


Figure 8: Overall architecture for FIR filtering.

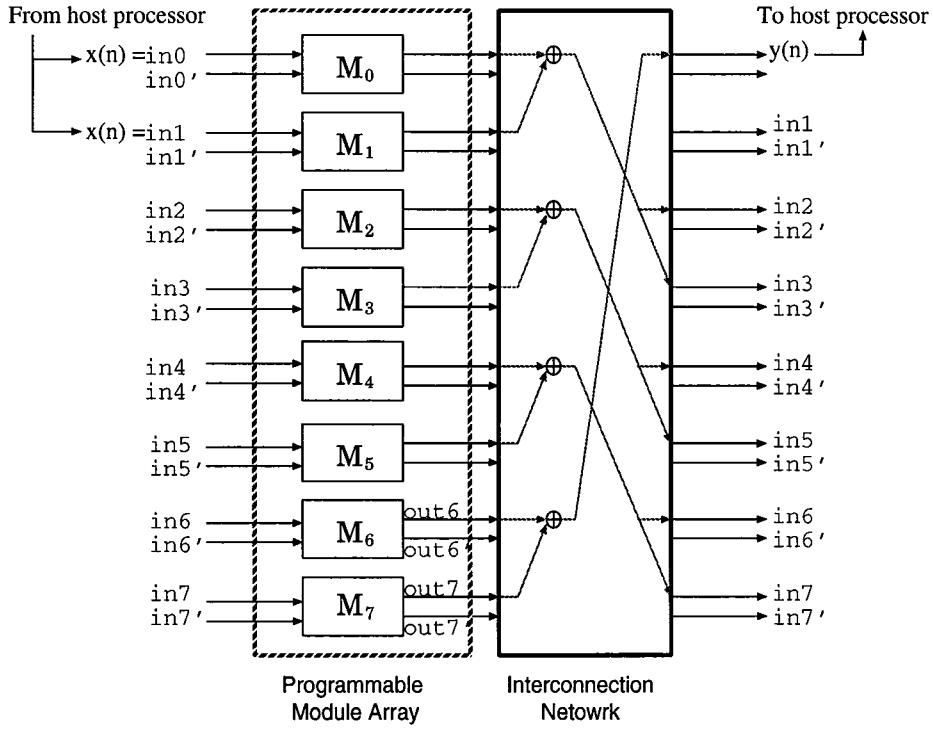


Figure 9: Overall architecture for IIR (ARMA) filtering.

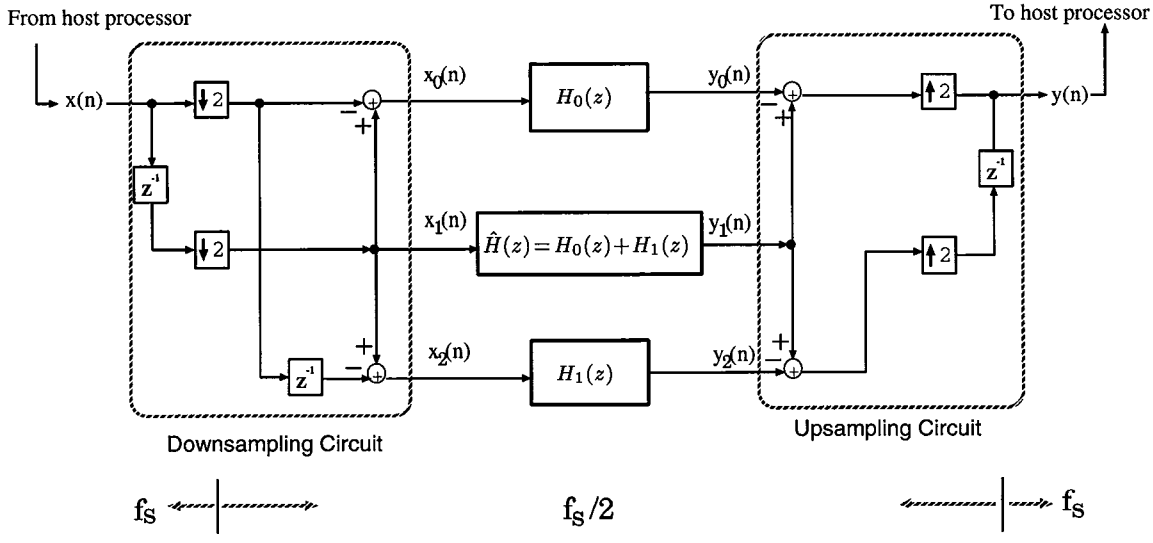
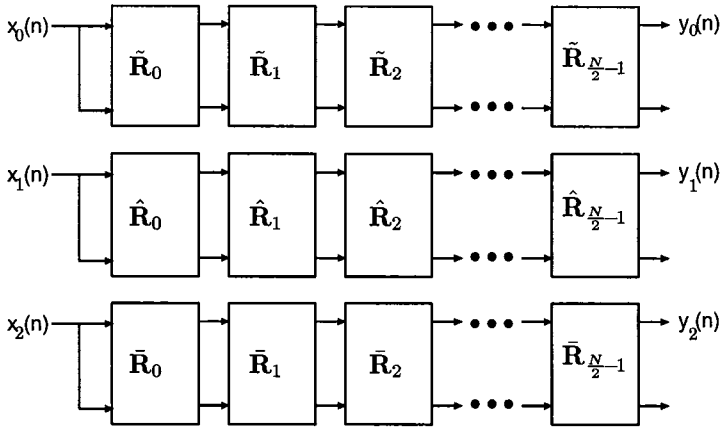
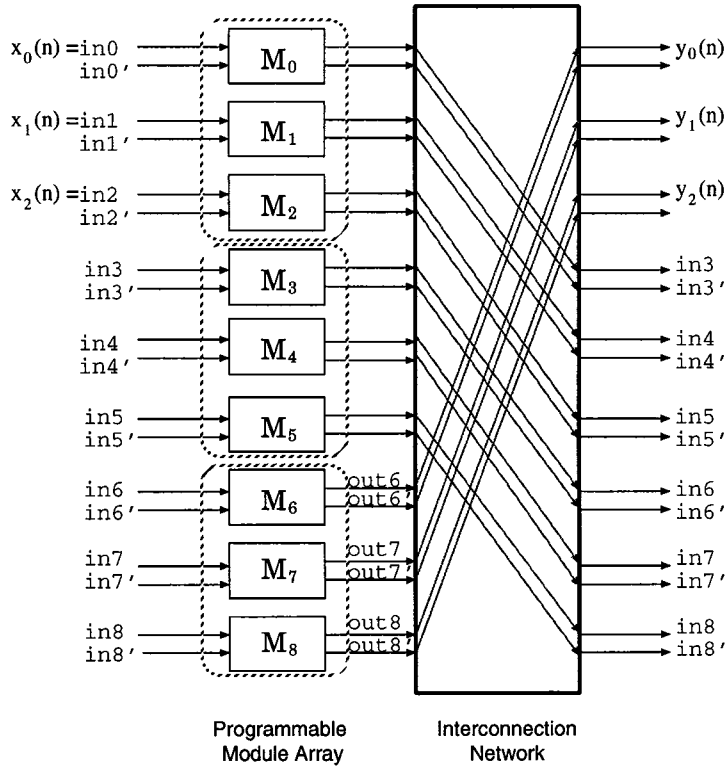


Figure 10: Multirate filtering architecture, where f_s is the data sample rate.

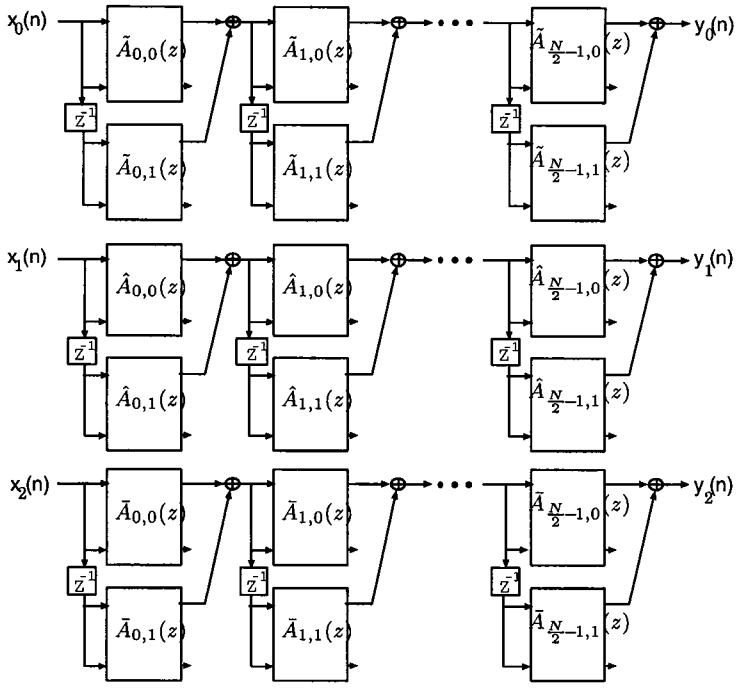


(a)

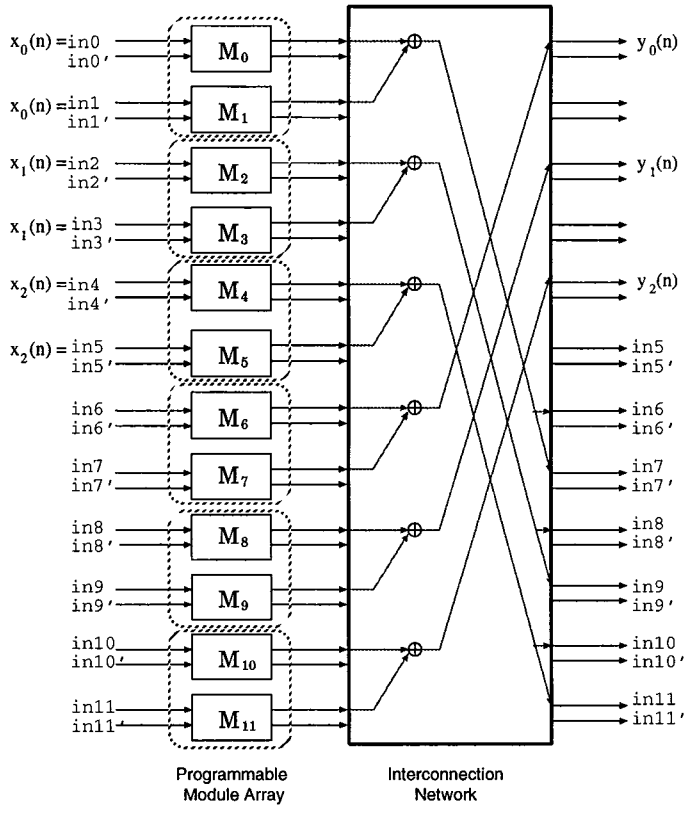


(b)

Figure 11: (a) Multirate FIR based on the lattice structure. (b) Mapping part (a) to the co-processor architecture: The figure demostrates the multirate $6th$-order FIR architecture using 9 programmable modules.

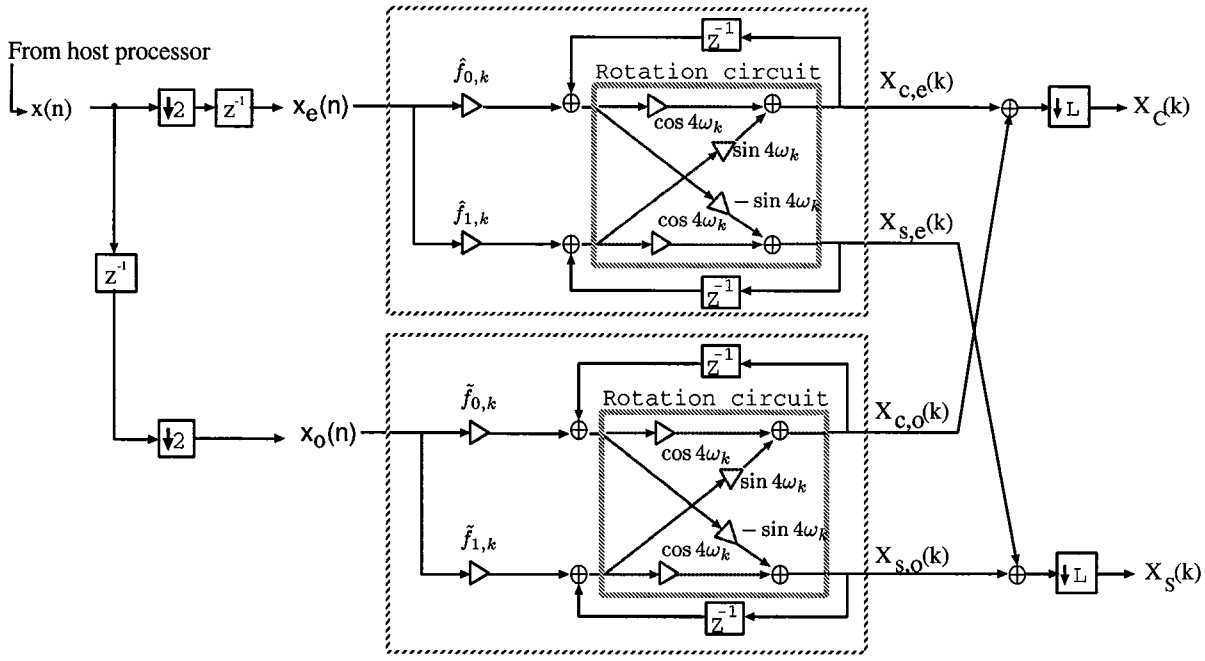


(a)

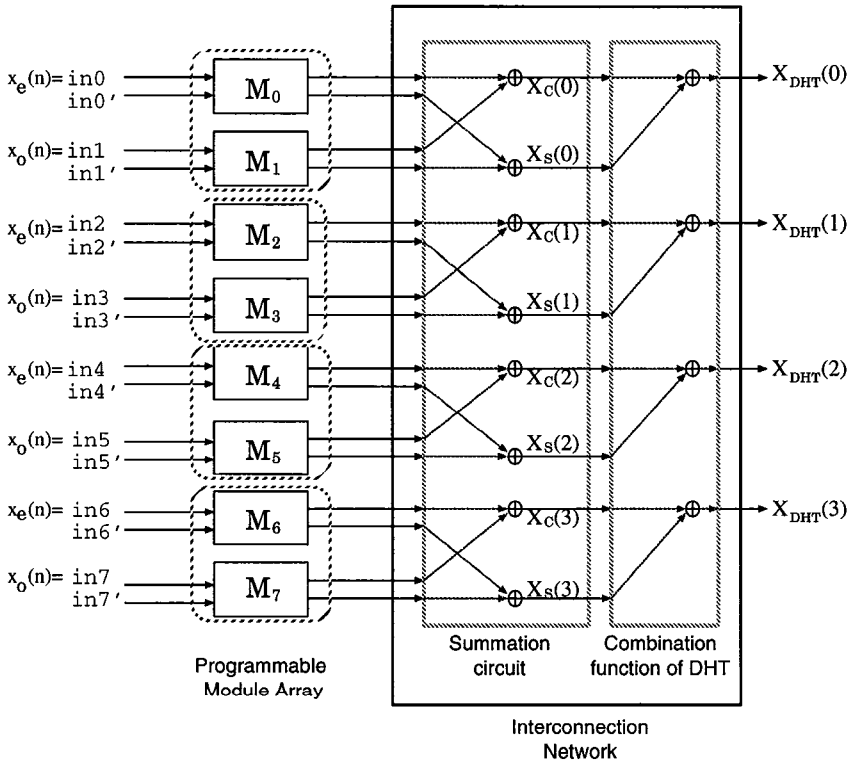


(b)

Figure 12: (a) Multirate IIR based on the lattice structure discussed in Section 2.3. (b) Mapping part (a) to the co-processor architecture: The figure demostrates the multirate $4th$-order IIR architecture using 12 programmable modules.



(a)



(b)

Figure 13: (a) Multirate architecture for the dual generation of $X_C(k)$ and $X_S(k)$. (b) Multirate 4-point DHT architecture based on 8 programmable modules.

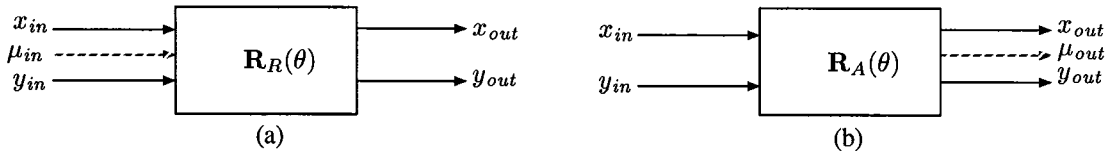


Figure 14: (a) CORDIC in *vector rotation mode.* (b) CORDIC in *angle accumulation* mode.

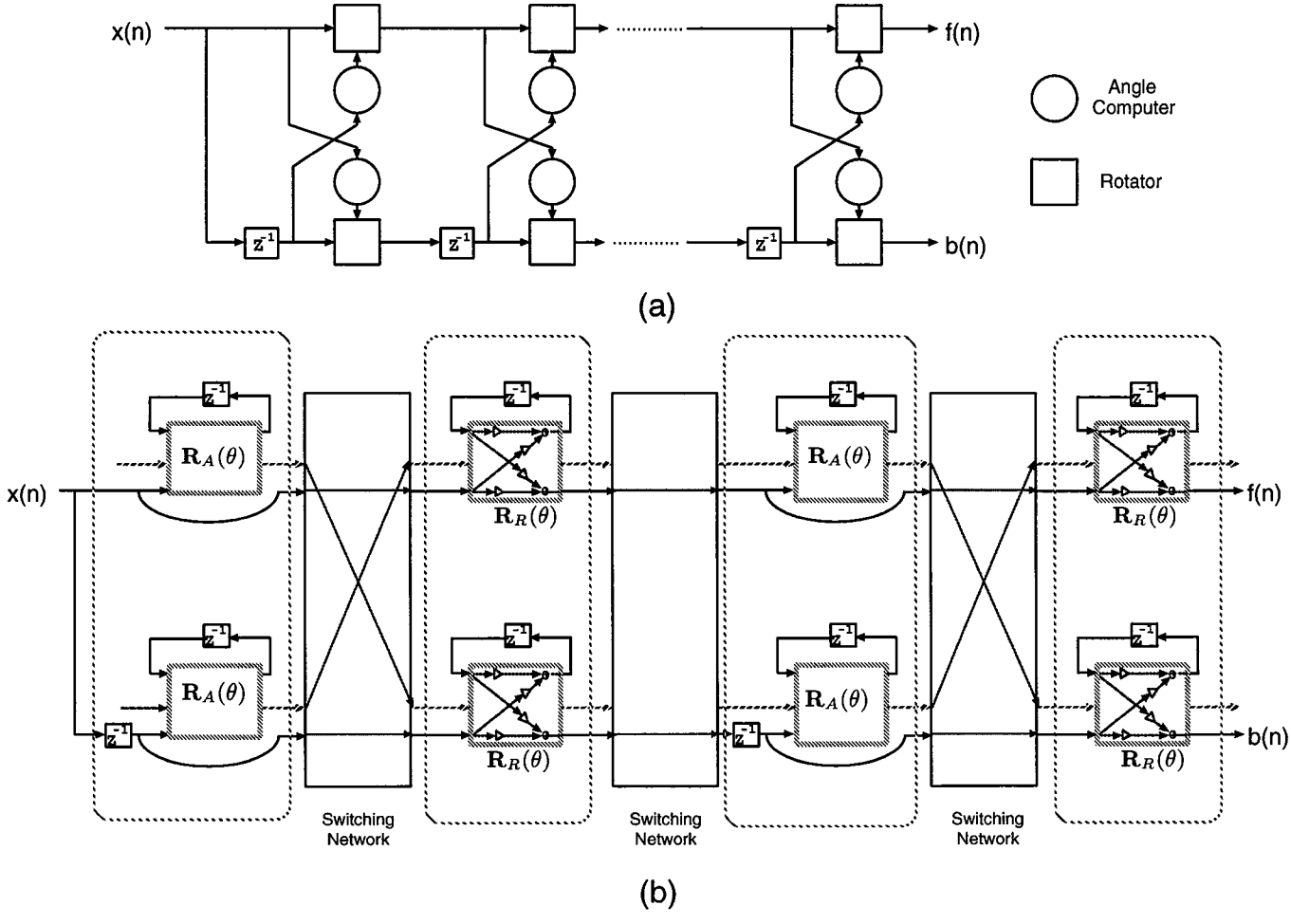


Figure 15: (a) QRD-LSL structure. (b) Realizing the QRD-LSL using the CORDIC processor.

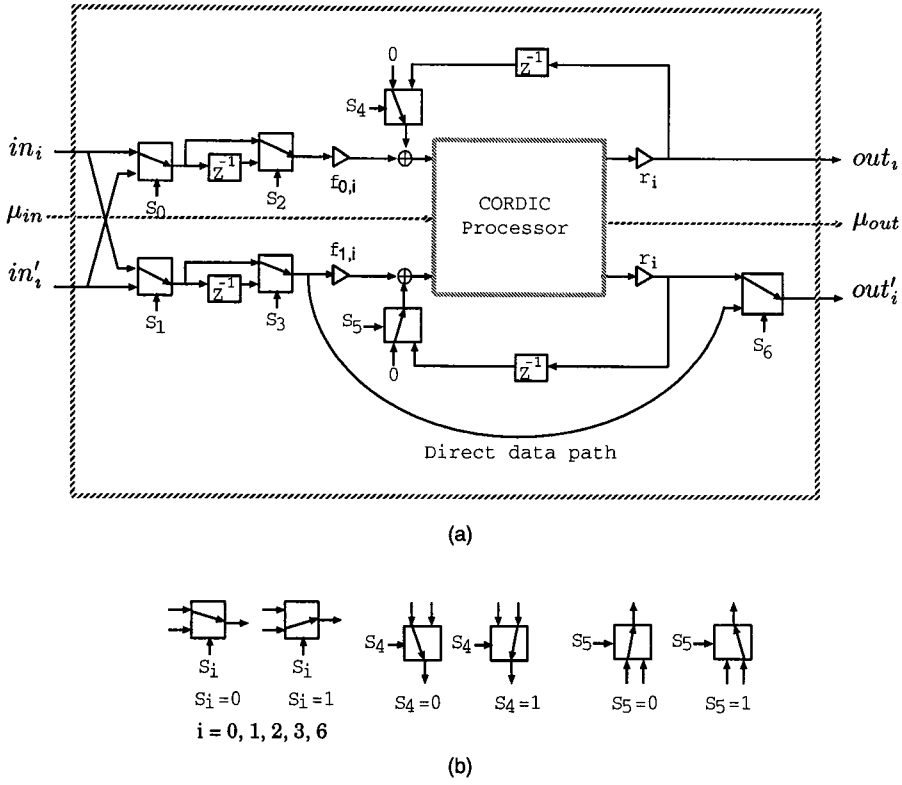


Figure 16: (a) New programmable module with the QRD-LSL feature. (b) Switches used in the module.

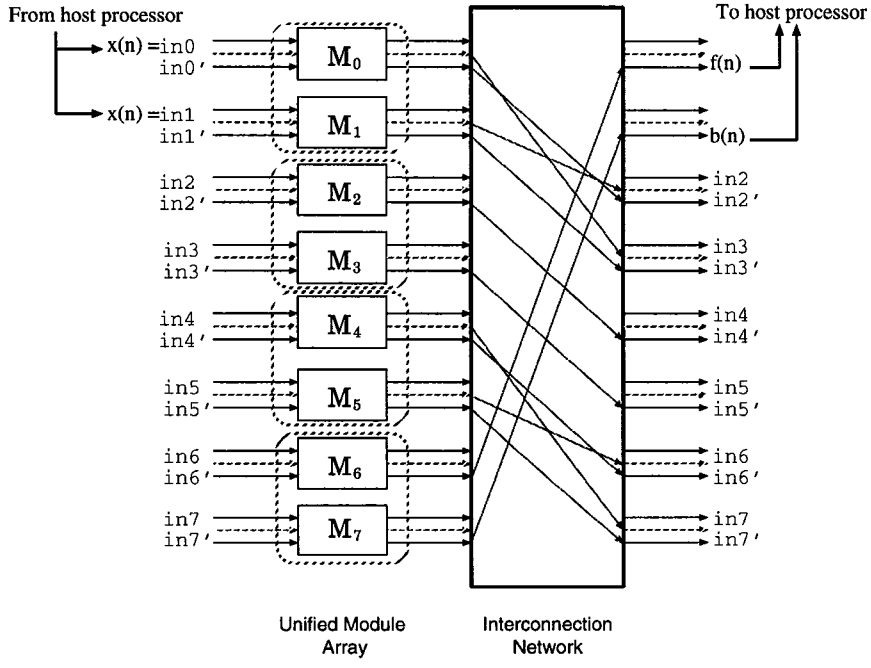


Figure 17: Realizing the QRD-LSL using the programmable video co-processor.