# A Power-Scalable Reconfigurable FFT/IFFT IC Based on a Multi-Processor Ring

Guichang Zhong, Fan Xu, *Member, IEEE*, and Alan N. Willson, Jr., *Fellow, IEEE*

***Abstract*—A single-chip reconfigurable FFT/IFFT processor that employs a ring-structured multiprocessor architecture is presented. Multi-level reconfigurability is realized by dynamically allocating computation resources needed by specific applications. The processor IC was fabricated in 0.25-$\mu$m CMOS. It performs 8-point to 4096-point complex FFT/IFFT with power-consumption scalability and provides useful trade-offs between algorithm flexibility, implementation complexity and energy efficiency.**

***Index Terms*—ASIC, cached-FFT, digital signal processor (DSP), fast Fourier transform (FFT), inverse FFT (IFFT), multi-processor, programmable, reconfigurable, scalable.**

## I. Introduction

IN THE DOMAIN of signal processing for communications, the current trends toward rapidly evolving standards and formats, and toward algorithms adaptive to dynamic factors in the environment, require programmable solutions that possess both algorithm flexibility and low implementation complexity. In addition, low power dissipation is critical in meeting portability requirements. Whereas designs using microprocessors and FPGAs provide implementation flexibility at the expense of higher power dissipation, compared to ASIC solutions [1], reconfigurable architectures have demonstrated better trade-offs between algorithm flexibility, implementation complexity, and energy efficiency [2].

Many recent communication standards propose OFDM or OFDMA as the primary modulation method. The fast Fourier transform (FFT), essential for such modulation, is both computation-intensive and data-exchange-intensive. Thus, an FFT processor's effectiveness plays a key role in optimizing system performance. It is desirable for a processor to perform flexible-size FFTs, thereby facilitating software upgradability when different formats and changing standards must be accommodated. For instance, in some applications, especially digital communications, digital signal processors (DSPs) are incorporated along with a RISC processor, with the DSP handling the data and the RISC handling the protocols. In such a system, when the DSP is merely an FFT processor for intensive data processing, its function in the system can be upgraded to a new standard with only software changes, provided that the FFT processor can accommodate flexible-size FFTs.

Also, it is highly desirable, in a practical system, that a processor perform FFTs with significantly scalable power dissipation across FFT sizes, as a means of keeping power dissipation to a minimum. To minimize power dissipation, an OFDM modulator might choose the smallest FFT that a channel allows and only switch to a larger one when the channel condition deteriorates [3]. In a specific system, an FFT processor could be required to meet certain performance specifications for the worst-case scenario—a maximum processing capability of 2048 points while consuming no more than 200 mW, for example. Then, two variable-size FFT processors may consume 200 mW when performing a 2048-point FFT, but for a 64-point FFT, one may consume a constant 200 mW power while the other consumes 100 mW. Obviously, when the system needs both 64-point and 2048-point FFTs, it is preferable to adopt the FFT processor having scalable power consumption for an overall minimization of power consumption.

While general-purpose DSPs can realize variable-length FFTs, their power dissipation can be relatively high and may not vary significantly with FFT size. Various dedicated FFT-processor implementations have been reported in [4]. Processors using pipeline architectures employ a series of butterfly processors, followed by angle rotations [5]. As opposed to pipeline processors containing multiple cascaded butterfly processors, processors having only a single butterfly processor are refered to as "nonpipeline" processors. Employing a cache between the butterfly processor and the main memory (a so-called "cached-memory architecture") on a nonpipeline processor has proved effective for increasing speed and energy efficiency since smaller memories are faster and require less energy per access. A custom-designed processor in [4] exhibits low power and high performance, however it lacks reconfigurability for variable-size FFTs; it also lacks flexibility in power dissipation. Although the flexibility in pipeline processors makes them potential candidates for reconfigurable FFT computation, they are typically less energy efficient than processors using cached-memory architectures, and usually employ many stages, resulting in greater hardware complexity.

The single-chip reconfigurable FFT/inverse FFT (IFFT)[1] processor presented here provides a solution midway between dedicated ASICs and software programmable general-purpose DSPs. This solution also happens to be midway between a nonpipelined and a pipelined implementation. It performs

Manuscript received January 18, 2005; revised August 4, 2005. This work was supported by Analog Devices, Broadcom Corporation, Maxim Integrated Products, and Rockwell Scientific through UC MICRO Grants 02-085, 03-094, and 04-101. TSMC provided support by fabricating the chip.

G. Zhong and A. N. Willson, Jr. are with the Electrical Engineering Department, University of California, Los Angeles, CA 90095 USA (e-mail: zhong@icsl.ucla.edu; willson@icsl.ucla.edu).

F. Xu is with Broadcom Corporation, Irvine, CA 92618 USA (e-mail: fxu@broadcom.com).

Digital Object Identifier 10.1109/JSSC.2005.862344

[1]Due to the strong similarities between the algorithmic processing in the FFT and IFFT, only the FFT algorithm and its implementation is discussed in detail in this paper.

8-point to 4096-point complex FFT/IFFTs with power-scalable features. The power scalability is due to the algorithm and reconfigurable architecture; it is not achieved by trading off power for arithmetic precision by means of variable data bitwidth or a variable number of filter taps [6].

## II. A Reconfigurable Architecture Based on Small-Size FFT Processors

### A. An FFT Decomposition Based on Small-Size FFTs

The $N$-point discrete Fourier transform (DFT) of a sequence $x(n)$ is defined as

$$X(k) = \sum_{n=0}^{N-1} x(n) W_N^{kn}, \quad 0 \le k \le N-1 \tag{1}$$

where $W_N = e^{-j2\pi/N}$. It can be decomposed into successively smaller DFTs and multiplications with phase factors [7]. For example, for an $N$-point DFT, let $N$ be factored as $N = r_1 r_2 r_3$ and let

$$n = n_2 r_2 r_3 + n_1 r_3 + n_0 \quad \begin{cases} n_0 = 0, 1, \ldots, r_3 - 1 \\ n_1 = 0, 1, \ldots, r_2 - 1 \\ n_2 = 0, 1, \ldots, r_1 - 1 \end{cases}$$

and

$$k = k_2 r_2 r_1 + k_1 r_1 + k_0 \quad \begin{cases} k_0 = 0, 1, \ldots, r_1 - 1 \\ k_1 = 0, 1, \ldots, r_2 - 1 \\ k_2 = 0, 1, \ldots, r_3 - 1. \end{cases}$$

Then the $N$-point DFT $X(k) = X(k_2, k_1, k_0)$ can be expressed in terms of $x(n) = x(n_2, n_1, n_0)$ as

$$\begin{aligned} X(k) &= \sum_{n_0=0}^{r_3-1} \sum_{n_1=0}^{r_2-1} \sum_{n_2=0}^{r_1-1} x(n_2, n_1, n_0) W_N^{(r_3 r_2 n_2 + r_3 n_1 + n_0)k} \\ &= \sum_{n_0=0}^{r_3-1} \sum_{n_1=0}^{r_2-1} F(k_0, n_1, n_0) W_{r_2}^{k_1 n_1} W_N^{(r_1 k_1 + k_0) n_0} W_{r_3}^{k_2 n_0} \\ &= \sum_{n_0=0}^{r_3-1} G(k_0, k_1, n_0) W_{r_3}^{k_2 n_0} \end{aligned} \tag{2}$$

where

$$F(k_0, n_1, n_0) = \left( \sum_{n_2=0}^{r_1-1} x(n_2, n_1, n_0) W_{r_1}^{k_0 n_2} \right) W_{r_1 r_2}^{k_0 n_1}$$

$$G(k_0, k_1, n_0) = \left( \sum_{n_1=0}^{r_2-1} F(k_0, n_1, n_0) W_{r_2}^{k_1 n_1} \right) W_N^{(r_1 k_1 + k_0) n_0}.$$

Hence, the large-size $N$-point DFT is subdivided into three passes of small-size DFTs. First, $F(k_0, n_1, n_0)$ is computed by an $r_1$-point FFT followed by a multiplication with the phase factor $W_{r_1 r_2}^{k_0 n_1}$. Second, an $r_2$-point FFT followed by a multiplication with $W_N^{(r_1 k_1 + k_0) n_0}$ realizes $G(k_0, k_1, n_0)$. Finally, an $r_3$-point FFT realizes $\sum_{n_0=0}^{r_3-1} G(k_0, k_1, n_0) W_{r_3}^{k_2 n_0}$.

The sequential character of the three operations lends itself to a logical pipeline structure with the output of one operation being an input to the next. Higher radix FFT algorithms are known to be more efficient than the classical radix-2 approach [8], and it can be more efficient to implement higher radix FFT algorithms on multiprocessor systems.

### B. A Reconfigurable Architecture

Based on the decomposition (2), the IC architecture employed here consists of two computation components: 1) a highly efficient small-size FFT ("atom-FFT") processor, mainly comprising a multi-processor ring, and 2) an angle rotator that performs phase-factor multiplications. The complete FFT processor (see Fig. 1) is integrated on a single chip and has these major blocks: a four-processor ring, a four-point FFT co-processor, an angle rotator, a memory interface and memory (RAM), address generators, a block-floating-point (BFP) unit, and built-in self-test (BIST).

The proposed architecture, being a hybrid of a pipelined and a nonpipelined processor, is similar to a cached-memory structure but the computation components are internally pipelined/paralleled. The multi-processor system can be programmed in a manner such that data flow around the ring, either clockwise or counter-clockwise or in both directions, while the processors simultaneously perform the necessary arithmetic operations. This is fundamentally different from a pipeline architecture where data flow is completely feed-forward.

This processor can be reconfigured, consistent with application and data-flow needs. The configuration process, employing configuration controllers and configurable networks (mainly multiplexers and I/O ports), occurs at two levels, the first concerning a flexible selection of small-size FFT computations and angle rotators, and, at a lower level, by datapath reconfiguration within components. At a higher level, the configuration controller uses configuration registers, providing the configurable-network-select signals that configure the processor for a given algorithm and address sequences for memories. This is a hardwired control strategy, avoiding the continuous fetch, decode and execution of instructions on each processor cycle. At a lower level, the configurable processing units are partitioned to allow optimization of the computational datapath by maximizing the hardware sharing while meeting the computational requirements. Also, the control logic required to configure the signal flow for each algorithm is minimized. Since the processor ring is programmable, it is less dependent on hardware to realize reconfiguration.

### C. Processor Ring

The ring-structured multiprocessor [9] balances its functionality, for a set of signal processing tasks, between ASIC implementations and the use of general-purpose DSPs. Unlike a DSP, it does not accommodate "all signal processing applications" but rather, realizes certain types of algorithms, such as digital filtering and small-size FFTs. The processor ring, however, in contrast to a typical full-custom IC, is not confined to one specific filter or FFT and retains much of the reconfigurability and programmability of a DSP. It has proved effective in realizing digital filters, delayed LMS algorithms, interpolation and decimation, fixed-point blind beamforming, a fixed-point adaptive

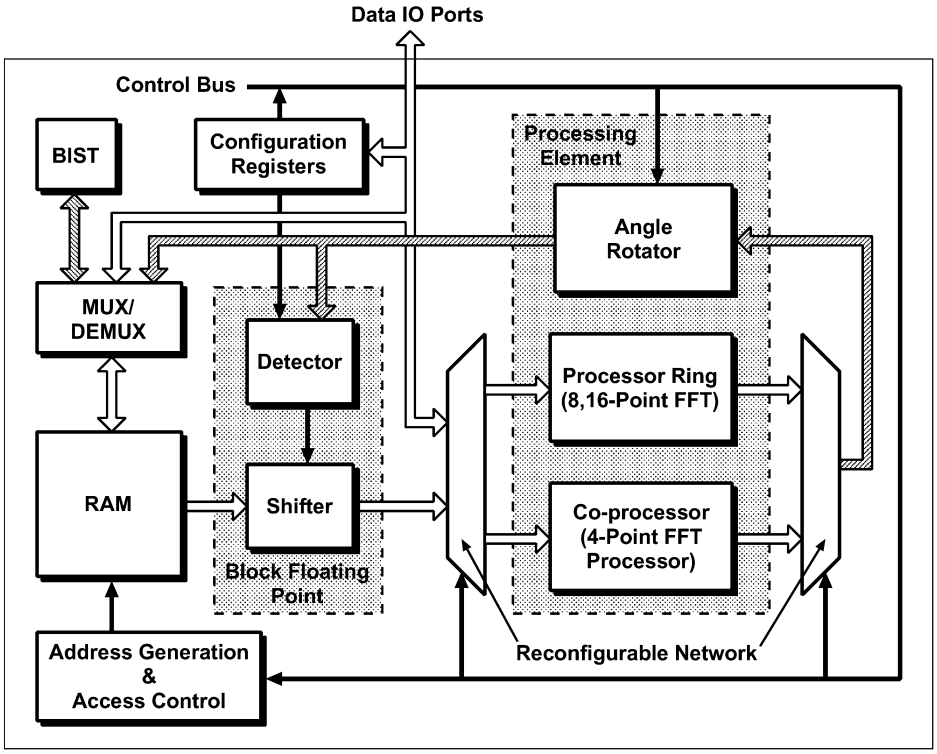



Fig. 1. Single-chip FFT processor structure.

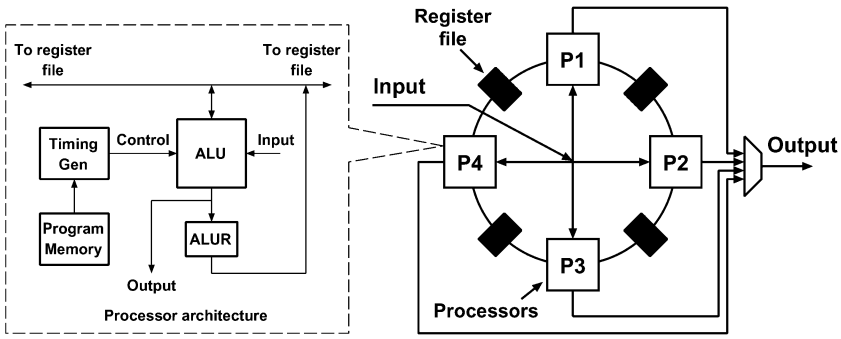



Fig. 2. Processor ring.

eigenvector algorithm, small-size FFTs, and other computations [9]–[13].

*1) Processor Ring Structure:* On a single chip, four processors are arranged in a ring topology, communicating with one another through on-chip dual-ported register files placed between adjacent pairs of processors, as shown in Fig. 2. This system's ring topology avoids the usual overhead needed to access off-chip RAMs, avoids memory-access contentions, and requires no bus arbitration hardware. In fact, no penalty, in comparison to bus-type architectures, has yet been observed when programming the processor ring for a variety of DSP applications.

The processor ring employs parallel (reduced instruction set) processors executing very long instruction words (VLIW). A typical instruction for any of the four processors specifies up to four separate operations: one *multiply*, one *data-move*, and two *add/subtracts*. Each processor's ALU, consisting of a multiplier, a RAM to store coefficients, and two adder/subtractors, as shown in Fig. 3, is pipelined and parallelized so that multiple instructions can execute simultaneously. Thus, the single-chip four-processor ring can execute 12 simultaneous computations per cycle: [4 processors $\times$ (1 multiply $+$ 2 add/subtract, simultaneously, per cycle, per processor)]. The flexible ALU datapaths also endow the processor with significant reconfigurability. The program memory acts not only as a source of program instructions, but also as a configuration controller. It configures the flow of input and output data through the computation datapath without instruction decoding. This removes the control logic complexity associated with dynamic changes in program control flow and instruction fetching. The input to an ALU can be from any of eight different sources, and the ALU outputs can be sent to multiple destinations at the same time. The computational datapath can be configured by the instructions to improve energy efficiency. For example, the multiplier path is inactive when executing instructions such as *move* and *add/subtract*. The primary (feed-forward)

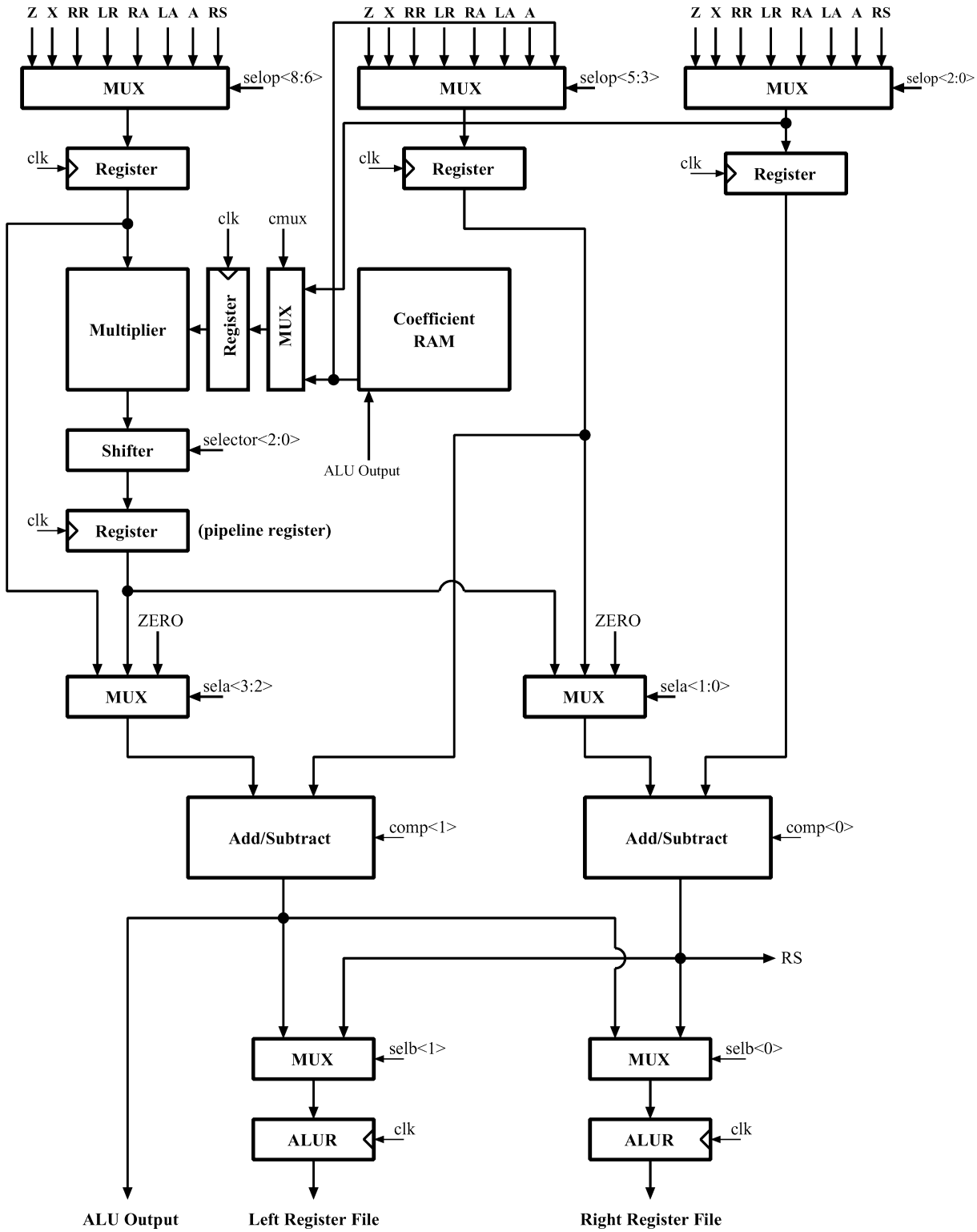



Fig. 3. ALU of a single processor of the four-processor ring.

FFT operations are multiply-accumulate (MAC) operations, which can be implemented in a highly efficient manner on the multi-processor ring.

*2) Small-Size FFTs on the Processor Ring:* The butterfly operation of a radix-2 decimate-in-time (DIT) FFT is represented as

$$X_m = X_{m-1} + W_N^k Y_{m-1}$$
$$Y_m = X_{m-1} - W_N^k Y_{m-1}$$

where $X_m = X_m^r + jX_m^i, Y_m = Y_m^r + jY_m^i$ and $W_N^k = W_r^k + jW_i^k$. The operation consists of four real multiplications and eight real additions:

$$X_m^r = X_{m-1}^r + \left(Y_{m-1}^r W_r^k - Y_{m-1}^i W_i^k\right)$$
$$Y_m^r = X_{m-1}^r - \left(Y_{m-1}^r W_r^k - Y_{m-1}^i W_i^k\right)$$
$$X_m^i = X_{m-1}^i + \left(Y_{m-1}^i W_r^k + Y_{m-1}^r W_i^k\right)$$
$$Y_m^i = X_{m-1}^i - \left(Y_{m-1}^i W_r^k + Y_{m-1}^r W_i^k\right)$$

which can be completed in a single clock cycle on the four-processor ring since it has four multipliers and eight adders. The (equivalent computation speed) program for a *two*-step butterfly on just *two* processors is shown in Table I, where processor P4 computes the real parts of $X_m$ and $Y_m$ while P3 computes imaginary parts. The multiplication $Y^r_{m-1}W^k_r$ occurs on P4 during program step 1 while a previous butterfly's final addition and subtraction are still being performed. The result is stored in the ALU's pipeline register $M^r_1$ located between its multiplier and its adder, for immediate use in further processing on the same processor. During step 2, the $M^r_1$ value is then used in the addition and subtraction that form $A^r_1$ and $A^r_2$ on the same processor and, simultaneously, $Y^i_{m-1}W^k_i$ is computed and stored in the pipeline register $M^r_2$ (same register as $M^r_1$) for use in the next step. The ALU step 2 outputs ($A^r_1$ and $A^r_2$) are used as ALU inputs during step 3 to form the final results $X^r_m$ and $Y^r_m$ of the present butterfly. Simultaneously, the next butterfly's $M^r_1$ can be computed, resulting in a continuous rate of two steps for each butterfly, on average. The $X^r_m$ and $Y^r_m$ results can be sent to external outputs or latched in the ALUR (operand result register) and written to a register file. Similar processing simultaneously occurs on P3, where imaginary parts are computed.

Since two butterflies can be processed at once, by using all four processors, the efficiency equals "one butterfly per clock cycle," with inter-processor data exchanges significantly reduced by using the Table I implementation. In particular, when programming 8- and 16-point FFTs on the processor ring, 12 clock cycles are needed for the 8-point FFT and 31 clock cycles suffice for the 16-point FFT.

*3) Example (8-Point FFT):* Fig. 4(a) shows the Signal Flow Graph (SFG) of an 8-point DIT FFT. Only two of the twelve butterflies need nontrivial complex multiplications (i.e., with $W^1_8$ and $W^3_8$). This 8-point FFT consists of three layers of butterflies, butterflies ①②③④ being the first layer, butterflies ⑤⑥⑦⑧ being the second, and so on. Figs. 4(b) and (c) show the implementation of the 8-point FFT on the processor ring, where it is pipelined into two stages, Stage I and Stage II, the input and the first two layers of butterflies being Stage I and the last butterfly layer and output being Stage II. The pipelining is realized automatically by assigning different processors to the stages: P1 and P2 realize Stage I while P3 and P4 deal with Stage II.

The real parts are computed on P1 and P4 and the imaginary parts, on P2 and P3. Variables named "$r$" (with added subscripts and superscripts) represent registers storing a number's real part, while imaginary parts are stored in registers labeled "$i$." Different superscripts indicate corresponding butterfly layers: "0" for the input data, "1" for the outputs of the first butterfly layer, "2" for the outputs of the second butterfly layer, and "3" for the outputs of the third layer. Variables having the same subscripts indicate variables on the same horizontal line in the SFG of Fig. 4(a). Variables having the same name but different accents such as "*" or "#", indicate that the same number is stored in different register file locations. For instance, row 10 in the program table realizes butterfly ⑧ on P1 and P2, where all inputs (outputs from the first layer) have a superscript "1". The output $r^2_5$ from P1 will be used by P3 and P4 for performing butterfly ⑩, hence the value of $r^2_5$ is sequentially passed on to $r^{2*}_5$ (by P1, on step 1 of the next pass through the program) then to $r^{2\#}_5$ (by P4 on step 3 of that second pass). $r^{2\#}_5$ is located in the register file between P3 and P4.

TABLE I
IMPLEMENTING BUTTERFLY BY TWO PROCESSORS

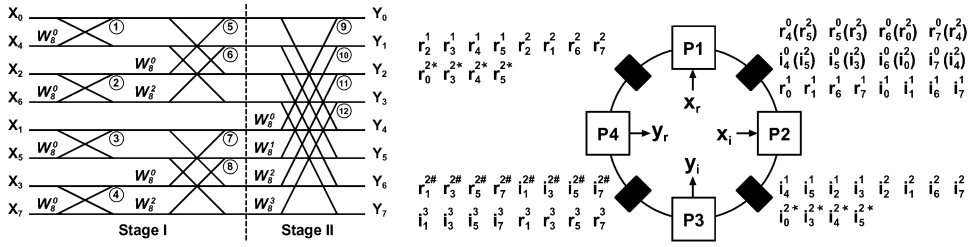
| | P1 | P2 | P3 | P4 |
|---|---|---|---|---|
| 1 | ... | ... | ... | ... |
| | | | ... | ... |
| | | | $M^i_1 \leftarrow Y^i_{m-1}W^k_r$ | $M^r_1 \leftarrow Y^r_{m-1}W^k_r$ |
| 2 | ... | ... | $A^i_1 \leftarrow X^i_{m-1} + M^i_1$ | $A^r_1 \leftarrow X^r_{m-1} + M^r_1$ |
| | | | $A^i_2 \leftarrow X^i_{m-1} - M^i_1$ | $A^r_2 \leftarrow X^r_{m-1} - M^r_1$ |
| | | | $M^i_2 \leftarrow Y^r_{m-1}W^k_i$ | $M^r_2 \leftarrow Y^i_{m-1}W^k_i$ |
| 3 | ... | ... | $Y^i_m \leftarrow A^i_2 - M^i_2$ | $Y^r_m \leftarrow A^r_2 + M^r_2$ |
| | | | $X^i_m \leftarrow A^i_1 + M^i_2$ | $X^r_m \leftarrow A^r_1 - M^r_2$ |
| | | | ... | ... |

In the program, a right arrow indicates a clockwise-directed output and a left arrow indicates a counterclockwise-directed output. For instance, the step 10 right arrow in $r^1_5 + i^1_7 \rightarrow r^2_5$ indicates that the addition result from P1 is sent clockwise to a register between P1 and P2. Different variables sharing the same physical memory locations are identified in Fig. 4(b) by use of parentheses, $r^0_4(r^2_5)$ for instance. Values of $W^1_8 = W^1_r + jW^1_i$ and $W^3_8 = W^3_r + jW^3_i$ were read from the coefficient memories.

The Fig. 4(c) program yields a complete (12-butterfly) 8-point FFT in 12 clock cycles, hence "one butterfly per clock cycle," even including data I/O. Sequential data I/O, which could contribute significantly to the computation time, is scheduled to overlap the computation, halving the data I/O time. Caches between main memory and processor ring rearrange the input data in an order of $x(4), x(5), x(6), x(7), x(0), x(1), x(2), x(3)$ to facilitate data processing on the processor ring. The processor ring is programed such that the output sequence is $y(0), y(2), y(1), y(3), y(4), y(6), y(5), y(7)$. It begins being produced on step 5 of the second pass through the program. Output storage addresses are generated by a 3-bit counter, and the natural order of the output sequence can generate the required sequence $\{0, 2, 1, 3, 4, \ldots\}$ by simply swapping the two LSBs of the 3-bit address, thus simplifying the traditional bit-reversal operations.

### *D. Four-Point Co-Processor*

Since it would be inefficient to reprogram the processor ring repeatedly for different size atom-FFTs (because processing would have to stop and wait for program memories to be refreshed), a dedicated (nonprogrammable) four-point FFT processor assists the processor ring for certain transform lengths whose decompositions involve two different sized atoms. (See the various decompositions in Table II.) Fig. 5(a) illustrates the SFG of a 4-point DIT radix-2 FFT, where the trivial multiplication by $-j$ can be done by real–imaginary interchanging and sign inversion. The architecture of [5] is adopted for the four-point FFT processor, as shown in Fig. 5(b). The operations of the 4-point FFT processor are summarized in




(a) (b)

| | P1 | P2 | P3 | P4 |
|---|---|---|---|---|
| 1 | $x_r(4) \rightarrow r_4^0$ | $i_4^0 \leftarrow x_i(4)$ | $i_1^2 \rightarrow i_1^{2\#}$ | $r_1^{2\#} \leftarrow r_1^2$ |
| | $r_5^{2*} \leftarrow r_5^2$ | $i_5^2 \rightarrow i_5^{2*}$ | $y_i(4) \leftarrow i_1^3$ | $y_r(4) \leftarrow r_1^3$ |
| 2 | $x_r(5) \rightarrow r_5^0$ | $i_5^0 \leftarrow x_i(5)$ | $i_7^{2\#} \leftarrow i_7^2$ | $r_7^{2\#} \leftarrow r_7^2$ |
| | $r_3^{2*} \leftarrow r_3^2$ | $i_3^2 \rightarrow i_3^{2*}$ | $y_i(6) \leftarrow i_3^3$ | $y_r(6) \leftarrow r_3^3$ |
| 3 | $x_r(6) \rightarrow r_6^0$ | $i_6^0 \leftarrow x_i(6)$ | $i_5^{2*} \rightarrow i_5^{2\#}$ | $r_5^{2\#} \leftarrow r_5^{2*}$ |
| | $r_0^{2*} \leftarrow r_0^2$ | $i_0^2 \rightarrow i_0^{2*}$ | $y_i(5) \leftarrow i_5^3$ | $y_r(5) \leftarrow r_5^3$ |
| 4 | $x_r(7) \rightarrow r_7^0$ | $i_7^0 \leftarrow x_i(7)$ | $i_3^{2*} \rightarrow i_3^{2\#}$ | $r_3^{2\#} \leftarrow r_3^{2*}$ |
| | $r_4^{2*} \leftarrow r_4^2$ | $i_4^2 \rightarrow i_4^{2*}$ | $y_i(7) \leftarrow i_7^3$ | $y_r(7) \leftarrow r_7^3$ |
| 5 | $x_r(0) + r_4^0 \rightarrow r_0^1$ | $i_0^1 \leftarrow x_i(0) + i_4^0$ | $i_0^{2*} - i_1^{2\#} \rightarrow i_1^3$ | $r_1^3 \leftarrow r_0^{2*} - r_1^{2\#}$ |
| | $r_4^1 \leftarrow x_r(0) - r_4^0$ | $x_i(0) - i_4^0 \rightarrow i_4^1$ | $y_i(0) \leftarrow i_0^{2*} + i_1^{2\#}$ | $y_r(0) \leftarrow r_0^{2*} + r_1^{2\#}$ |
| 6 | $x_r(1) + r_5^0 \rightarrow r_1^1$ | $i_1^1 \leftarrow x_i(1) + i_5^0$ | $i_2^2 + r_3^{2\#} \rightarrow i_3^3$ | $r_3^3 \leftarrow r_2^2 - i_3^{2\#}$ |
| | $r_5^1 \leftarrow x_r(1) - r_5^0$ | $x_i(1) - i_5^0 \rightarrow i_5^1$ | $A_1^i \leftarrow i_2^2 - r_3^{2\#}$ | $A_1^r \leftarrow r_2^2 + i_3^{2\#}$ |
| 7 | $r_2^1 \leftarrow x_r(2) + r_6^0$ | $x_i(2) + i_6^0 \rightarrow i_2^1$ | $y_i(2) \leftarrow A_1^i$ | $y_r(2) \leftarrow A_1^r$ |
| | $x_r(2) - r_6^0 \rightarrow r_6^1$ | $i_6^1 \leftarrow x_i(2) - i_6^0$ | | |
| 8 | $r_3^1 \leftarrow x_r(3) + r_7^0$ | $x_i(3) + i_7^0 \rightarrow i_3^1$ | $M_1^i \leftarrow i_5^{2\#} W_r^1$ | $M_1^r \leftarrow r_5^{2\#} W_r^1$ |
| | $x_r(3) - r_7^0 \rightarrow r_7^1$ | $i_7^1 \leftarrow x_i(3) - i_7^0$ | | |
| 9 | $r_0^1 + r_2^1 \rightarrow r_0^2$ | $i_0^2 \leftarrow i_0^1 + i_2^1$ | $A_1^i \leftarrow i_4^{2*} + M_1^i$ | $A_1^r \leftarrow r_4^{2*} + M_1^r$ |
| | $r_2^2 \leftarrow r_0^1 - r_2^1$ | $i_0^1 - i_2^1 \rightarrow i_2^2$ | $A_2^i \leftarrow i_4^{2*} - M_1^i$ | $A_2^r \leftarrow r_4^{2*} - M_1^r$ |
| | | | $M_2^i \leftarrow r_5^{2\#} W_i^1$ | $M_2^r \leftarrow i_5^{2\#} W_i^1$ |
| 10 | $r_5^1 + i_7^1 \rightarrow r_5^2$ | $i_5^2 \leftarrow i_5^1 - r_7^1$ | $A_2^i - M_2^i \rightarrow i_5^3$ | $r_5^3 \leftarrow A_2^r + M_2^r$ |
| | $r_7^2 \leftarrow r_5^1 - i_7^1$ | $i_5^1 + r_7^1 \rightarrow i_7^2$ | $y_i(1) \leftarrow A_1^i + M_2^i$ | $y_r(1) \leftarrow A_1^r - M_2^r$ |
| | | | $M_1^i \leftarrow i_7^{2\#} W_r^3$ | $M_1^r \leftarrow r_7^{2\#} W_r^3$ |
| 11 | $r_4^1 + i_6^1 \rightarrow r_4^2$ | $i_4^2 \leftarrow i_4^1 - r_6^1$ | $A_1^i \leftarrow i_6^2 + M_1^i$ | $A_1^r \leftarrow r_6^2 + M_1^r$ |
| | $r_6^2 \leftarrow r_4^1 - i_6^1$ | $i_4^1 + r_6^1 \rightarrow i_6^2$ | $A_2^i \leftarrow i_6^2 - M_1^i$ | $A_2^r \leftarrow r_6^2 - M_1^r$ |
| | | | $M_2^i \leftarrow r_7^{2\#} W_i^3$ | $M_2^r \leftarrow i_7^{2\#} W_i^3$ |
| 12 | $r_1^2 \leftarrow r_1^1 + r_3^1$ | $i_1^1 + i_3^1 \rightarrow i_1^2$ | $A_2^i - M_2^i \rightarrow i_7^3$ | $r_7^3 \leftarrow A_2^r + M_2^r$ |
| | $r_1^1 - r_3^1 \rightarrow r_3^2$ | $i_3^2 \leftarrow i_1^1 - i_3^1$ | $y_i(3) \leftarrow A_1^i + M_2^i$ | $y_r(3) \leftarrow A_1^r - M_2^r$ |

(c)

Fig. 4. Eight-point FFT: (a) DIT flow graph; (b) processor ring implementation; (c) processor ring program.

Table III, where each stage is in 50% utilization, equivalent to only one active butterfly consuming power in each clock cycle.

### *E. Angle Rotator*

The angle rotator efficiently computes $(X + jY) = (X_0 + jY_0)e^{j\phi}$ by rotating the complex data-point $(X_0, Y_0)$ about the origin, through an input angle $\phi$ to produce $(X, Y)$ where

$$X = X_0 \cos\phi - Y_0 \sin\phi$$
$$Y = Y_0 \cos\phi + X_0 \sin\phi. \quad (3)$$

This could be done by Coordinate Rotation Digital Computer (CORDIC) processors [14] or processors such as [15] and [16]. Such algorithms accomplish the rotation through successive subrotations, with each subrotation stage input being the previous stage output. This, however, would involve increased latency due to iterations. An alternative implementation multiplies $X_0$ and $Y_0$ with pre-computed sine/cosine values stored in a ROM, which achieves lower latency. However, since the ROM size grows exponentially with the precision of $\phi$, a rather large ROM would be required to achieve accurate results. Several methods to reduce the lookup table size have been presented in recent years [17], [18]. CORDIC can achieve good

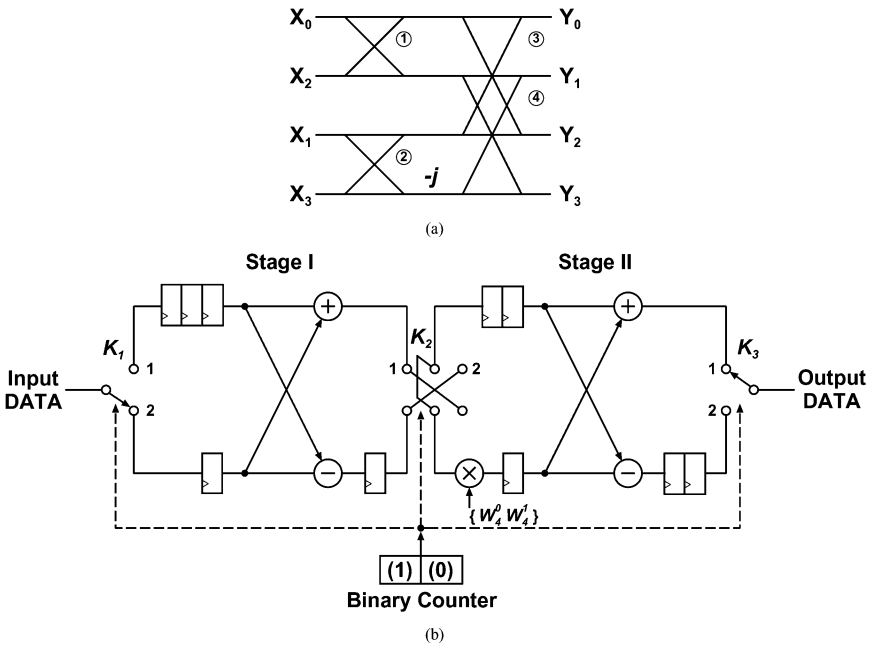



Fig. 5. Four-point co-processor: (a) DIT FFT flow graph; (b) pipeline structure for 4-point FFT.

TABLE II
POSSIBLE DECOMPOSITIONS OF 32-4096 POINT FFTS

| Size of FFT | 32 | 64 | 128 | 256 | 512 | 1024 | 2048 | 4096 |
|---|---|---|---|---|---|---|---|---|
| Possible decomposition | 8×4 | 8×8 | 8×4 ×4 | 8×8 ×4 | 8×8 ×8 | 16×16 ×4 | 8×8 ×8×4 | 8×8 ×8×8 |

TABLE III
OPERATIONS OF THE 4-POINT FFT PROCESSOR

| Counter | Switch Position | | | Stage | Stage | Twiddle | Input | Output |
|---|---|---|---|---|---|---|---|---|
| Value | $K_1$ | $K_2$ | $K_3$ | I | II | Factor | | |
| 00 | 1 | 2 | 2 | ② | idle | $W_4^0$ | $X_0$ | $Y_3$ |
| 01 | 1 | 1 | 1 | idle | ③ | $W_4^1$ | $X_1$ | $Y_0$ |
| 10 | 2 | 2 | 1 | idle | ④ | $W_4^0$ | $X_2$ | $Y_1$ |
| 11 | 2 | 1 | 2 | ① | idle | $W_4^0$ | $X_3$ | $Y_2$ |

reconfigurability, but only at the cost of a complex controller and relatively long latency, and the direct table-lookup method has poor reconfigurability. In the present FFT processor, a reconfigurable angle rotator architecture provides a compromise between the CORDIC and direct table lookup schemes, achieving higher energy efficiency without loss of accuracy and speed. In a manner similar to that employed in [19], the lookup table for a 12-bit angle control word $\bar{\phi}$ (where $\phi = 2\pi\bar{\phi}$) is reduced from $2^{12} = 4096$ locations to $2^5 + 1 = 33$ locations, a reduction factor of 124.

The angle control word $\bar{\phi} = \bar{\phi}_M + \bar{\phi}_L$ becomes $\bar{\theta} = \bar{\theta}_M + \bar{\theta}_L$ after a mapping (Table IV) of $\phi$ into the first octant $[0, \pi/4]$. (See [18] and [19] for mapping details.) Here

$$\bar{\theta}_M = d_1 2^{-1} + \cdots + d_M 2^{-M}$$
$$\bar{\theta}_L = d_{M+1} 2^{-M-1} + \cdots + d_B 2^{-B}.$$

Since the angle control word $\bar{\phi}$ is relatively short (12 bits), so is $\bar{\theta}_L$ (4 bits). Moreover, not all bits are always active. Therefore, the architecture shown in Fig. 6 is used, wherein a coarse stage is followed by four fixed angle sub-rotators. Compared to two-stage DDS-type angle rotators [18], [19], a smaller area and an 85% savings in power is achieved for the fine rotation stages. This is due to the sine/cosine values being fixed for each stage, hence requiring no approximation of these values, as is needed in [18] and [19]. Furthermore, since the sine values and (1− cosine) values are small, by using carry-save addition (CSA) and sub-expression sharing techniques, no general purpose multipliers are necessary and IC area is significantly reduced. A fine stage, rotating by $\pi/1024$ radians, is shown in Fig. 7. A coarse angle-rotation stage involves multiplications of input data $X_0$ and $Y_0$ by the $\cos\theta_M$ and $\sin\theta_M$ values. It is implemented by a complex multiplier and adders, plus a 33-word × 35-bit ROM table, as in [19]. More details are given in [20], where it is also shown that full-accuracy in the datapath outputs is achieved.

This scheme also benefits the angle rotator in speed. Compared to the adder tree stages in a general purpose multiplier—the Booth multiplier used in [19] for instance—the number of multiplierless CSA stages is smaller and needs no Booth multiplier recoding circuit. This yields a shorter critical path in the sub-rotators. The smaller area also results in shorter routing wires, hence less RC delay in the critical path.

This angle rotator is more flexible and reconfigurable than those in [19]. It is configured by selecting coarse-stage or sub-rotator outputs, and simultaneously shutting down unneeded sub-rotators. No sub-rotator is activated unless the "resolution" of the angle is greater than $\pi/256$ (i.e., for the longer FFTs); even then, a sub-rotator "stands by" if its control bit is zero, thus reducing power consumption. The datapath data-flow configuration network consists of a decoder and multiplexers. It can be configured for IFFT operations (wherein angles rotate in the opposite direction) by interchanging the inputs $X_0$ and $Y_0$, and interchanging the outputs. These operations, easily integrated with the mapping operations of Table IV, require no additional logic circuits.

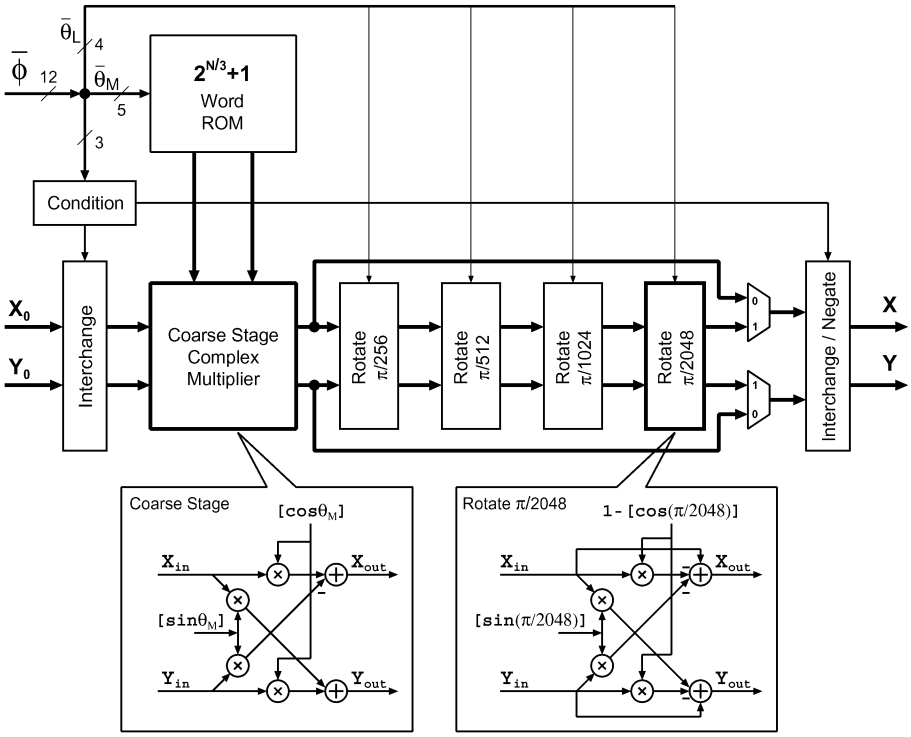



Fig. 6. Reconfigurable angle-rotator architecture.

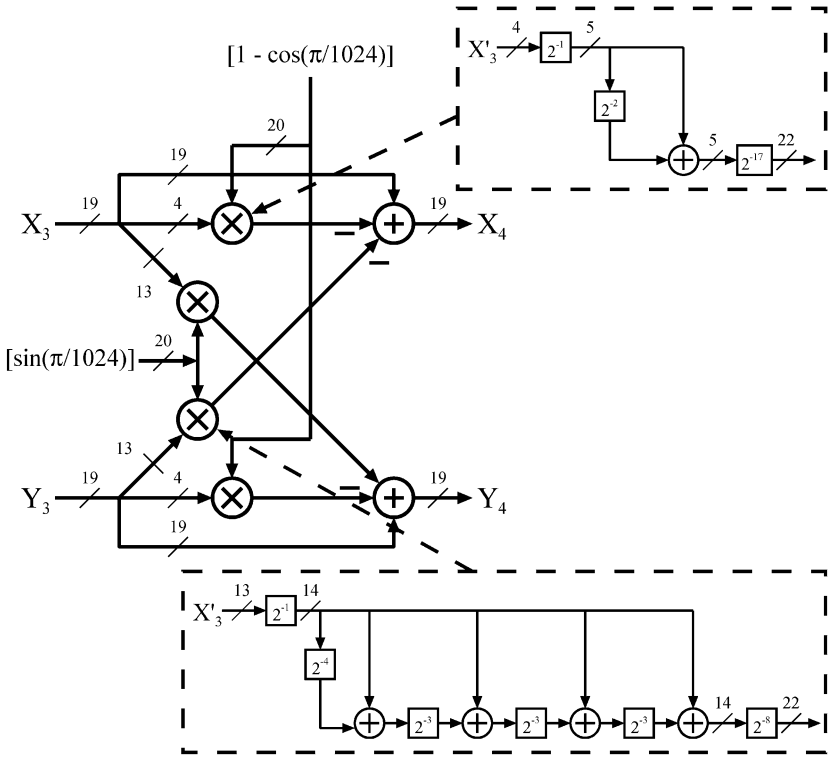



Fig. 7. A sub-rotator for rotation of $\pi/1024$.

### *F. Memory Hierarchy*

The memory system, organized in a cached-memory architecture, is distinguished from [4] by its hierarchical arrangement and efficiency due to a higher radix algorithm. As shown in Fig. 8, memories are organized in a four-level hierarchy: main memory storing input and output data; caches between main memory and processor ring; register files between individual processors within the ring; and instruction and coefficient memories within each processor.

On-chip dual-port SRAM is provided for main memory on the present prototype chip to avoid off-chip memory-transfer bottlenecks. It can be replaced with off-chip memory for suitable practical applications. The main memory is organized as three banks of 2048-word by 32-bit dual-ported SRAM arrays.

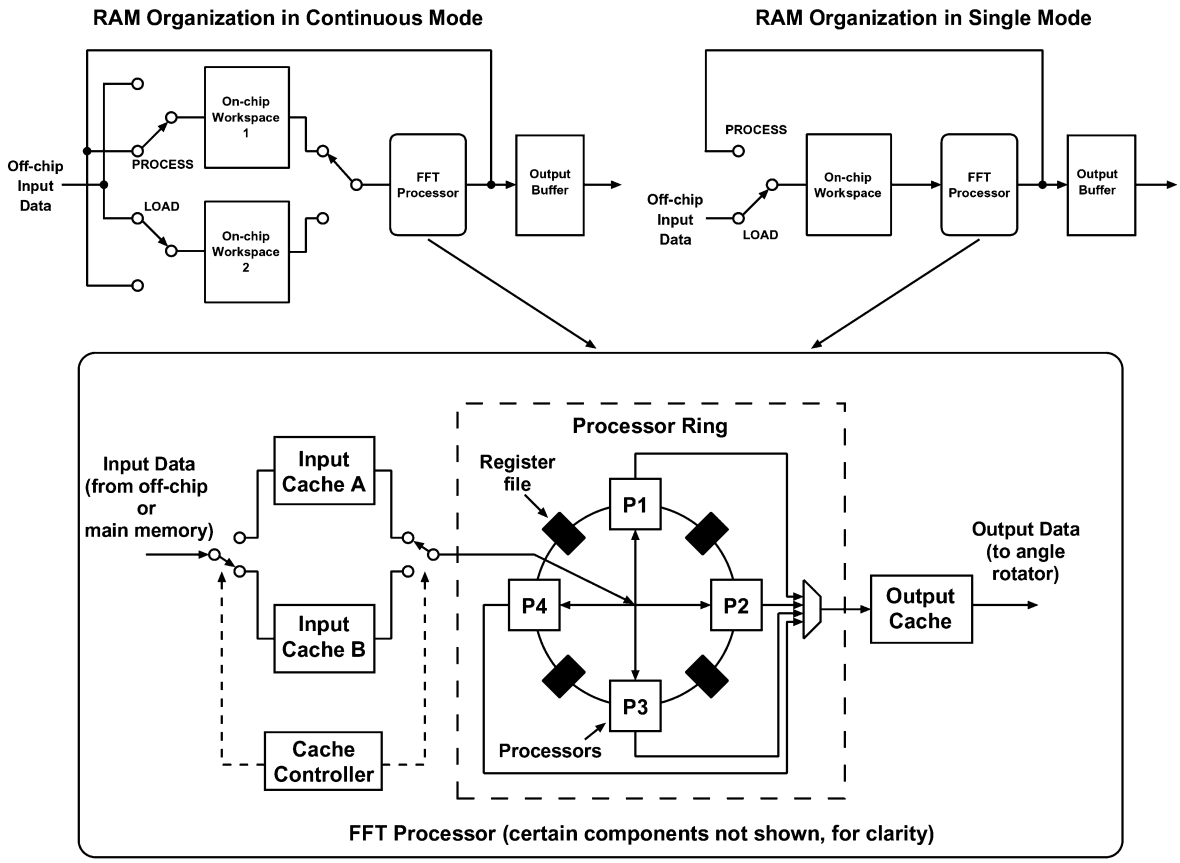



Fig. 8. Memory hierarchy of the FFT processor.

TABLE IV
MAPPING FROM $[0, 2\pi)$ TO $[0, \pi/4]$

| Octant | Swap Inputs | Complement $\bar{\theta}$ | Swap Outputs | Negate Outputs | |
|---|---|---|---|---|---|
| | | | | $X$ | $Y$ |
| 000 | | | | | |
| 001 | √ | √ | | √ | |
| 010 | | | √ | √ | |
| 011 | √ | √ | √ | √ | √ |
| 100 | | | | √ | √ |
| 101 | √ | √ | | | √ |
| 110 | | | √ | | √ |
| 111 | √ | √ | √ | | |

Improving on general memory hierarchy caches, those in the FFT processor reduce access latency and processor-memory traffic and benefit FFT operations; the input caches reorder inputs for more efficient processor ring use, and the output cache performs bit-reversal reordering for the atom-FFTs, thus avoiding or simplifying higher level bit-reversal computations. A dual set of caches allow new data to be loaded from main memory while prior inputs are being processed. Thus, no processor ever waits for data. Register files between processors simplify the inter-processor communication protocols and reduce the inter-processor data exchange. The three caches and register files consist of 32-word by 24-bit arrays using eight transistors per cell.

Each processor has an instruction memory holding up to forty 54-bit wide instructions, and a coefficient memory with up to 32 16-bit coefficients. Both are single-port register files using six-transistor cells. The program memory and coefficient memory have proved adequate to implement atom-FFTs up to 16 points, as well as other DSP applications [12].

The memory system can be reconfigured, depending on the operation modes: "continuous" or "single." In continuous mode, the RAM organization allows simultaneous input of new data and the transforming of data already stored in RAM, and outputting of previous results. In single mode, the IC is configured with separate load, transform and output operations. Eight-point to 2048-point FFTs can be transformed in both modes; a 4096-point FFT can only be programmed in single mode due to the on-chip SRAM's capacity. A memory interface circuit controls the reconfiguration, data input/output, and data exchanges between main memory and processors.

### *G. Address Generation Unit*

There are three address generators in the FFT processor: input and output address generators, providing correct RAM addresses during data input and output, and an angle-rotator address generator providing angle-rotator control words. The generators are similar in principle; we discuss the input address generator. Unlike radix-2 processors, this FFT processor's input address generator does not perform simple bit-reverse operations but a switching operation in blocks of two, three, or four bits, depending on the atom-FFT size; the block-switching pattern differs in different computation stages.

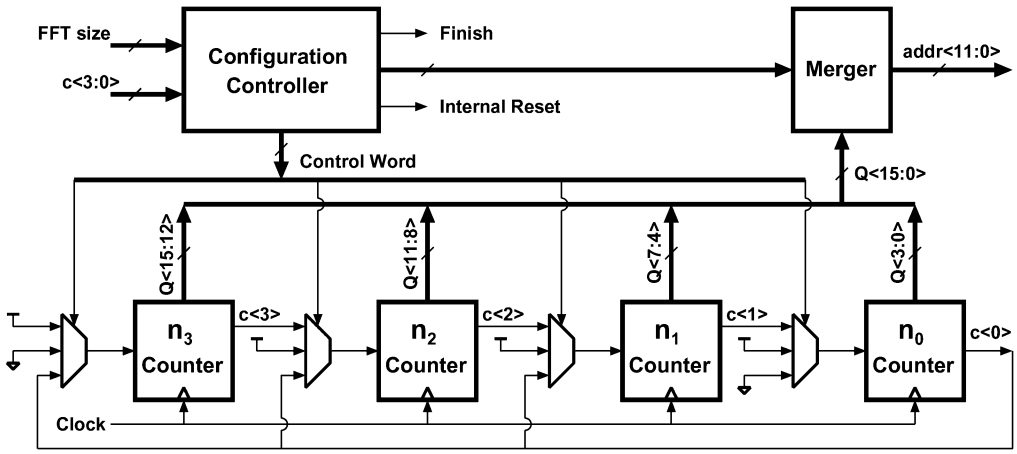



Fig. 9. Address generator.

A 2K FFT, for example, has $N$ factored as $N = r_1r_2r_3r_4$, with $r_1 = r_2 = r_3 = 8$ and $r_4 = 4$. The input address is $n = (r_2r_3r_4)n_3 + (r_3r_4)n_2 + r_4n_1 + n_0$, where $n_0 \in \{0,1,2,3\}, n_1, n_2, n_3 \in \{0,1,\ldots,6,7\}$. By this decomposition, the FFT is split into four passes each equivalent to small-size FFT operations on a four-dimensional array indexed by $(n_3, n_2, n_1, n_0)$. A straightforward address-computation method would use multipliers and adders, having larger area and higher power consumption. Alternatively, the indexing can be treated as four nested loops, each navigating through the values of a specific variable. The looping sequence changes from one pass to another and from one size FFT to another, and it can be arranged in a "circular" manner without changing the overall FFT result. An effective multiplier-free auto-reconfigured scheme is achieved (Fig. 9). Four counters are arranged in a loop, each capable of configuration as a variable-width counter from one to four bits. For different FFT sizes, from 8 to 4096 points, the whole counter is configured with widths from 3 to 14 bits. During each pass of a specific-sized FFT computation, each counter can be configured as an "initial" counter performing the innermost loop, feeding its carry signal to the following counter (its next outer loop), and so on. The controller configures the address pattern by selecting counter inputs or enabling/disabling specific counters. For example, in the first pass of a 2K FFT performing 256 8-point FFTs, the initial counter $n_3$ counts the three LSBs every clock cycle, $n_2$ counts the next higher three LSBs every 8 clock cycles, $n_1$ counts the next three bits every 64 clock cycles, and $n_0$ counts the two MSBs every 256 cycles. Thus, configured, the Merger (Fig. 9) rearranges the counter output and forms the final addresses according to FFT size and the pass that is running.

By using in-place operations, the output address generator is similarly implemented, differing only in that the merger of the output address generator combines the counter output in the form $(r_3r_2r_1)n_0 + (r_2r_1)n_1 + r_1n_2 + n_3$ so that outputs are stored in natural order.

### H. Block Floating Point

Block floating point (BFP) operations significantly improve numerical performance of FFT processors [21]. In theory, using separate BFP blocks for the small-FFT processor and angle rotator provides better performance. However, simulations on 64-, 256- and 1024-point FFTs have shown that the signal-to-quantization-noise ratio (SQNR) improvement, compared to a one-BFP solution, is very limited [12]. Shifting data before writing to memory (output scaling) seems to yield better numerical performance, but it severely complicates the circuit. Thus, only one BFP block is used here, and the BFP shifter is at the FFT's input port, i.e., data are not shifted until they are read into the processor for the next stage of computation (input scaling).

The BFP block consists of a BFP detector and a BFP shifter. The detector picks out the largest output (in magnitude) of one pass, and records its number of MSB zeros as the "block-scale factor" used by the shifter for input scaling. The detector is composed of a datapath and an FSM controller, where the datapath includes priority encoders and exponent accumulators (comparators). For each input data point $X + jY$ sent to the detector, the priority encoder detects and records the number of MSB zero bits, i.e., the scale factor. The accumulator searches for the minimum scale factor using comparators. The controller generates a set of control signals, grouped as a word that controls the detector datapath, and checks the status signals from the detector datapath to determine the next state and control signals generated. The controller waits until the input data are available (after a latency time), then enables the detector datapath for a proper time period, next latches the output, and finally resets to the initial state after all data are processed.

## III. Prototype Implementation

The FFT processor was implemented using a standard-cell-based design methodology and the 0.25-$\mu$m Artisan Components library plus full-custom cache and register file blocks. The chip was fabricated in TSMC 0.25-$\mu$m 5-level-metal CMOS. Fig. 10 shows the chip microphotograph and Table V summarizes its overall characteristics.

### A. High-Level Design

Matlab was used for algorithm-level simulation and verification. Logic design was done using a Cadence Verilog-XL simulator. The processor ring, angle rotator, and address generators were the most demanding parts of the design. To achieve

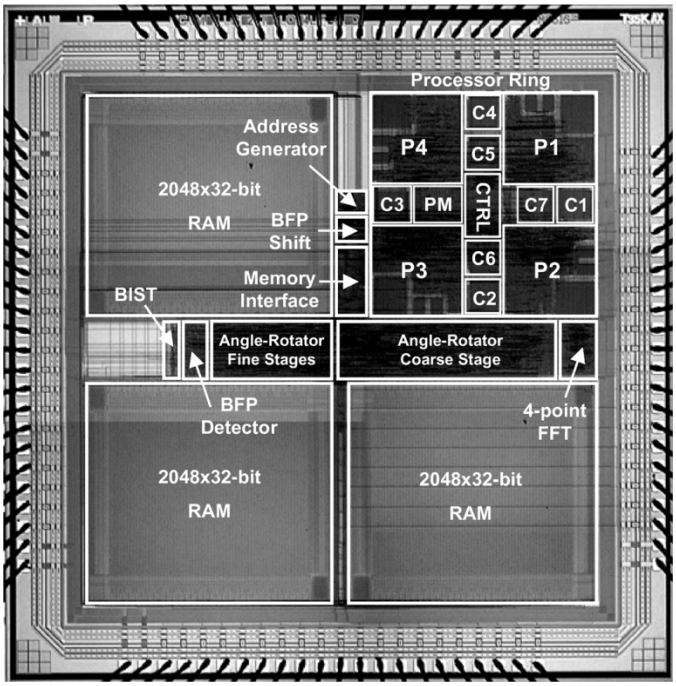



Fig. 10. Chip microphotograph.

TABLE V
CHIP CHARACTERISTICS

| | |
|---|---|
| Technology | TSMC 0.25-$\mu$m 5-level-metal CMOS |
| Instruction format | VLIW |
| Die size | $4.28mm \times 4.28mm$ |
| Core size | $3.38mm \times 3.38mm$ |
| Transistor count | 2,265,732 |
| Port buswidth | 16-bit |
| Internal buswidth | 24-bit |
| Program memory size | 4×(40-word×54-bit) |
| Coefficient memory size | 4×(32-word×16-bit) |
| Cache memory size | 7×(32-word×24-bit) |
| Main memory size | 3×(2048-word×32-bit) |
| Maximum clock rate | 200 MHz (@2.5 V) |

a compact and fast implementation, their design was based on gate-level structural descriptions. The BFP design and other control logic was based on behavioral descriptions and synthesized. The 16-bit multiplier (24-bit output) in the ALU uses Booth encoding and a Wallace tree adder array without internal pipelining. The complex multiplier in the angle rotator coarse stage was implemented by two-stage pipelined Booth multipliers; complex multiplications in the fine rotation stages were implemented multiplier-free.

### B. Full-Custom Cell Design

Cache memories and register files in the processor system require high speed and low power consumption; performance is primarily affected by cache memories and register files. Fully custom caches and register files were therefore designed, allowing a trade-off between speed and power consumption, while the main data memory was generated by a RAM compiler.

### C. Place and Route

With a satisfactory Verilog netlist from the pre-layout verification, the final place and route (P&R) was performed. The timing information from P&R was back-annotated to the netlist for post-layout simulation. The P&R iteration continued until timing and functionality in post-layout simulation met system specifications, and the layout passed DRC and LVS checks.

The prototype chip's floorplan is shown in Fig. 10, where the logic part (approximately 0.52 million transistors) occupies approximately 1/4 of the core area, while SRAM blocks occupy the remaining 3/4 of the core area. When using the FFT/IFFT processor in a system-on-chip application, this memory can likely be provided by other on-chip shared memory. Then, the FFT processor's area could more correctly be assessed as 1/4 the Fig. 10 area, i.e., 1.7 mm × 1.7 mm.

For such a large-scale ASIC, a major floorplanning concern is the clock distribution network, a key factor in achieving high speed and low power dissipation. A manually designed clock tree was tuned for low clock skew. Another concern is to provide equal path lengths to each processor for input and output and minimizing the path length between a processor and its two adjacent register file blocks. The four-processor ring has its input and output blocks and controller centrally located, within the IC's upper-right corner, and processors and register files are placed symmetrically around them. The ALU, program memory, and coefficient memory in each individual processor are placed to minimize path delays between them.

## IV. IC TESTING AND RESULTS

### A. Testing Strategy

Functionality, speed, and power were tested using an HP82000 IC tester and on-chip BIST features. The tester verified the logic functions of the FFT processor. On-chip BIST was used to measure maximum operating frequency and power consumption. A custom printed circuit board was designed so logic analysis systems could program the FFT processor and a signal generator could provide a high-frequency clock signal through an SMA connector on the PCB.

### B. Results and Comparison

With a real sinewave as the input of a 1K FFT, the SQNR is measured as 61.23 dB, as is a 72.67 dB spurious-free dynamic range (SFDR). The measured power dissipation when the present processor is programmed for various FFT sizes, is given in Fig. 11(a), and shows a significant scalable-power-dissipation feature with varying FFT size. This reconfigurable FFT processor, thus, can outperform those whose power consumption does not scale with FFT size. One processor recently reported [22], for example, consumes 545 mW for a 64-point FFT at 65 MHz and 574 mW for a 2048-point FFT at 60 MHz, exhibiting limited scaling in power consumption. For the present processor ring based chip, a 64-point FFT consumes about 30% less power than a 2048-point FFT at the same data rate. The power dissipation of the processor in [23] equals 300 mW for a 2K FFT at 20 MHz and, according to [4], it is 300 mW for a 1K FFT at 20 MHz as well (even though it scales to 600 mW for an 8K FFT).

TABLE VI
COMPARING TO INDUSTRY BENCHMARKS OF PROGRAMMABLE FIXED-POINT DSPS FOR 1K FFT

| | Processor ring based FFT Processor | Texas Instruments TMS320C64x [25] | Analog Devices Sharc [26] | Texas Instruments TMS320C62x [27] |
|---|---|---|---|---|
| Clock cycles | 5280 | 6002 | 9200 | 12972 |
| Normalized clock cycles | 1 | 1.14 | 1.74 | 2.46 |

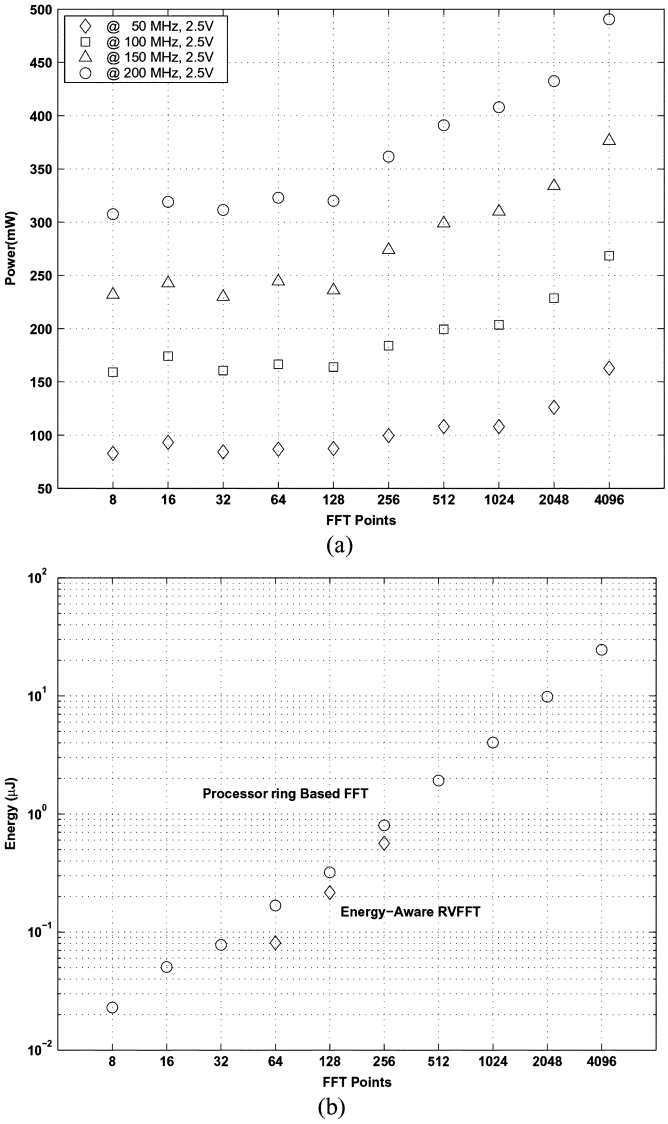



Fig. 11. (a) Power consumption versus FFT size. (b) Comparing the energy-scalability.

Wang *et al.* [24] have proposed energy scalability, a relative quantity, as an energy-efficiency criterion. An energy-aware real-valued FFT (RVFFT) [24] showed better energy-scalability characteristics than a nonscalable architecture when performing RVFFTs having $N = 128$, 256 and 512 points. The RVFFT is implemented in 0.18-$\mu$m CMOS, while the processor ring based FFT processor uses 0.25-$\mu$m CMOS and its FFT is a full complex-FFT, not a real-valued FFT. Energy can be normalized to the 0.18-$\mu$m technology with the following relationship:

$$\text{Normalized Energy} = \frac{\text{Energy}}{(0.25/0.18)(2.5/1.8)^2}$$

since the dynamic energy dissipation is proportional to $CV^2$ and $C$ scales approximately as 0.25/0.18. The energy dissipated by the RVFFT for $N = 128$, 256 and 512 is compared with that of the complex-FFT for $N = 64$, 128 and 256. Fig. 11(b) shows that the processor ring based FFT processor has comparable energy scalability to that of the energy-aware RVFFT, even though it is designed to perform a much wider range (8 point to 4096 point) of complex FFTs.

It is appropriate to comment on the use of power scalability as opposed to energy scalability as a design criterion. In situations wherein individual FFT operations occur relatively infrequently, energy scalability may be the better low-power assessment. The energy scalability discussed in [24] is based on the energy dissipated for a single FFT run, i.e., $E = P \times T_{\text{FFT}}$, where $T_{\text{FFT}}$ is the time required to perform a specific size FFT. (Notice that, even for programmable processors in which $P$ does not scale with FFT size, "scalability" in $E$ can still be achieved simply because $T_{\text{FFT}}$ scales with FFT size.) Where an FFT processor is running continuously, however, the scalability of $P$ with FFT size seems a more appropriate design criterion. That is, it can be more meaningful to consider the scalability of the power consumption [as in Fig. 11(a)] rather than the energy dissipated in a single FFT computation [as in Fig. 11(b)].

When comparing the performance of a processor such as the one presented here,[2] versus the computation of an FFT on a general purpose DSP, it is more insightful to remove hardware and IC-technology-specific variations, and focus directly on relative architectural merits. This processor, for example, can compute a 1K FFT (an industry benchmark) in 5280 clock cycles—taking approximately 26.4 $\mu$s at the maximum clock rate of 200 MHz, with a supply voltage of 2.5 V. Table VI shows that, using clock cycles as a comparison criterion, this processor outperforms various well-known fixed-point DSPs, in a 1K FFT operation.

## V. CONCLUSION

A reconfigurable FFT/IFFT processor based on a multi-processor ring that achieves significantly scalable power-consumption with varying (8 point to 4096 point) FFT size has been presented. By decomposing a large FFT into small-size FFTs, the use of a multi-level reconfigurable architecture and highly energy-efficient processing units provides flexible choices of computation resources under varying FFT sizes, thus reducing implementation complexity and improving energy efficiency. Compared to state-of-the-art FFT processors, the trade-off between algorithm flexibility, implementation complexity, and energy efficiency achieved by this processor makes it an ideal

[2]The processor is mainly designed as an FFT processor; however, it also inherits the full functionality of its four-processor ring predecessor. It is fully functional for digital filtering, blind beamforming [13], etc.

candidate for systems requiring power-dissipation scalability in highly changing environments, and for systems requiring multi-standard and multi-algorithm capability.

## ACKNOWLEDGMENT

The authors would like to thank the anonymous reviewers of this paper for their helpful and constructive comments that have substantially improved the paper's exposition.

## REFERENCES

[1] N. Tredennick and B. Shimamoto, "Go reconfigure," *IEEE Spectrum*, vol. 40, no. 12, pp. 26–40, Dec. 2003.
[2] J. M. Rabaey, "Reconfigurable processing: The solution to low-power programmable DSP," in *Proc. ICASSP*, 1997, pp. 275–278.
[3] N. Zhang and R. W. Brodersen, "Architectural evaluation of flexible digital signal processing for wireless receivers," in *Conf. Rec. 34th Asilomar Conf. Signals, Systems and Computers*, vol. 1, 2000, pp. 78–83.
[4] B. M. Baas, "A low-power, high-performance, 1024-point FFT processor," *IEEE J. Solid-State Circuits*, vol. 34, no. 3, pp. 380–387, Mar. 1999.
[5] S. He and M. Torkelson, "Design and implementation of a 1024-point pipeline FFT processor," in *Proc. CICC*, 1998, pp. 131–134.
[6] R. Amirtharajah, T. Xanthopoulos, and A. Chandrakasan, "Power scalable processing using distributed arithmetic," in *Proc. ISLPED*, Aug. 1999, pp. 170–175.
[7] J. W. Cooley and J. W. Tukey, "An algorithm for the machine calculation of complex Fourier series," *Math. Comput.*, vol. 19, pp. 297–301, 1965.
[8] G. D. Bergland, "A fast Fourier transform algorithm using base 8 iterations," *Math. Comput.*, vol. 22, pp. 275–279, 1968.
[9] A. Kwentus, M. Werter, and A. N. Willson Jr., "A programmable digital filter IC employing multiple processors on a single chip," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 2, no. 2, pp. 231–243, Jun. 1992.
[10] M. J. Werter and A. N. Willson Jr., "Automated programming of digital filters for parallel processing implementation," *IEEE Trans. Circuits Syst. II: Analog Digit. Signal Process.*, vol. 41, no. 4, pp. 285–294, Apr. 1994.
[11] D. D. Richard III, "A ring processor for digital filter applications," Master's thesis, Univ. California, Los Angeles, CA, 1999.
[12] F. Xu, "On designing high-performance signal processing algorithms for a ring-structured multiprocessor," Ph.D. dissertation, Univ. California, Los Angeles, CA, 2001.
[13] F. Xu, G. Zhong, and A. N. Willson Jr., "Analysis and VLSI realization of a blind beamforming algorithm," *J. VLSI Signal Process.*, vol. 40, no. 2, pp. 159–174, Jun. 2005.
[14] A. Ahn, S. Nahm, and W. Sung, "VLSI design of a CORDIC-based derotator," in *Proc. ISCAS*, 1998, pp. 449–452.
[15] A. Madisetti, A. Y. Kwentus, and A. N. Willson Jr., "A 100-MHz, 16-b, direct digital frequency synthesizer with a 100-dBc spurious-free dynamic range," *IEEE J. Solid-State Circuits*, vol. 34, no. 8, pp. 1034–1043, Aug. 1999.
[16] D. DeCaro, E. Napoli, and A. G. M. Strollo, "Direct digital frequency synthesizer using high-order polynomial approximation," in *IEEE ISSCC Dig. Tech. Papers*, vol. 1, 2002, pp. 134–135.
[17] H. T. Nicolas and H. Samueli, "An analysis of the output spectrum of direct digital frequency synthesizer performance in the presence of finite word length effects," in *Proc. 41st. Annu. Symp. Frequency Control*, 1987, pp. 495–502.
[18] D. Fu, "Efficient synchronization architectures for multimedia communications," Ph.D. dissertation, Univ. California, Los Angeles, CA, 2000.
[19] A. Torosyan, D. Fu, and A. N. Willson Jr., "A 300 MHz quadrature direct digital synthesizer/mixer in 0.25-$\mu$m CMOS," *IEEE J. Solid-State Circuits*, vol. 38, no. 6, pp. 875–887, Jun. 2003.
[20] G. Zhong, F. Xu, D. Fu, and A. N. Willson Jr., "An energy-efficient reconfigurable angle-rotator architecture," in *Proc. ISCAS*, vol. 3, 2004, pp. 661–664.
[21] W. W. Smith and J. M. Smith, *Handbook of Real-Time Fast Fourier Transforms*. Piscataway, NJ: IEEE Press, 1995.
[22] J. Kuo, C. Wen, and A. Wu, "Implementation of a programmable 64–2048-point FFT/IFFT processor for OFDM-based communication systems," in *Proc. ISCAS*, vol. 2, May 2003, pp. 121–124.
[23] E. Bidet, D. Castelain, C. Joanblanq, and P. Senn, "A fast single-chip implementation of 8192 compex point FFT," *IEEE J. Solid-State Circuits*, vol. 30, no. 3, pp. 300–305, Mar. 1995.
[24] A. Wang and A. Chandrakasan, "Energy-aware architecture for a real-valued FFT implementation," in *Proc. ISLPED*, Aug. 2003, pp. 360–365.
[25] FFT Benchmarks, Texas Instruments Inc. [Online]. Available: http://www.ti.com/sc/docs/products/dsp/c6000/benchmarks/64x.htm#fft
[26] Analog Devices. [Online]. Available: http://www.analog.com/processors/processors/sharc/benchmarks.html
[27] FFT Benchmarks, Texas Instruments Inc. [Online]. Available: http://www.ti.com/sc/docs/products/dsp/c6000/benchmarks/62x.htm#fft

**Guichang Zhong** received the B.S. degree from Xi'an Jiaotong University, China, in 1996 and the M.S. degree from the Institute of Microelectronics of Chinese Academy of Sciences, Beijing, China, in 2000, both in electrical engineering. He is currently working toward the Ph.D. degree in integrated circuits and systems at the University of California, Los Angeles.

His present research interests are in high-performance VLSI digital signal processor design, with an emphasis on reconfigurable and energy-efficient architectures.

**Fan Xu** (S'98–M'01) received the B.S. and M.S. degrees in electronics engineering from Tsinghua University, Beijing, China, in 1993 and 1996, respectively, and the Ph.D. degree in electrical engineering from the University of California, Los Angeles, in 2001. His Ph.D. research focused on algorithm design and analysis for digital signal processors and eigenvector estimation architectures.

In 1997, he held an internship at Bell Labs, Lucent Technologies, Holmdel, NJ, where he worked on equalization algorithms for cellular systems. He joined Broadcom Corporation, Irvine, CA, in 2001. His research interests include VLSI signal processing, customized digital signal processors, efficient hardware architectures for adaptive signal processing and high-performance VLSI design.

**Alan N. Willson, Jr.** (S'66–M'67–SM'73–F'78) received the B.E.E. degree from the Georgia Institute of Technology, Atlanta, in 1961, and the M.S. and Ph.D. degrees from Syracuse University, Syracuse, NY, in 1965 and 1967, respectively.

From 1961 to 1964, he was with IBM, Poughkeepsie, NY. He was an Instructor in electrical engineering at Syracuse University from 1965 to 1967. From 1967 to 1973, he was a Member of the Technical Staff at Bell Laboratories, Murray Hill, NJ. Since 1973, he has been on the faculty of the University of California, Los Angeles (UCLA), where he is Professor of Engineering and Applied Science in the Electrical Engineering Department. In addition, he served the UCLA School of Engineering and Applied Science as Assistant Dean for Graduate Studies from 1977 through 1981 and as Associate Dean of Engineering from 1987 through 2001. He has been engaged in research concerning computer-aided circuit analysis and design, the stability of distributed circuits, properties of nonlinear networks, theory of active circuits, digital signal processing, analog circuit fault diagnosis, and integrated circuits for signal processing. He is the editor of *Nonlinear Networks: Theory and Analysis* (IEEE Press, 1974). In 1991, he founded Pentomics, Inc.

Dr. Willson is a member of Eta Kappa Nu, Sigma Xi, Tau Beta Pi, the Society for Industrial and Applied Mathematics, and the American Society for Engineering Education. From 1977 to 1979, he served as Editor of the IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS. In 1980, he was General Chairman of the 14th Asilomar Conference on Circuits, Systems, and Computers. During 1984, he served as President of the IEEE Circuits and Systems Society. He was the recipient of the 1978 and 1994 Guillemin–Cauer Awards of the IEEE Circuits and Systems Society, the 1982 George Westinghouse Award of the American Society for Engineering Education, the 1982 Distinguished Faculty Award of the UCLA Engineering Alumni Association, the 1984 Myril B. Reed Best Paper Award of the Midwest Symposium on Circuits and Systems, the 1985 and 1994 W. R. G. Baker Awards of the IEEE, the 2000 Technical Achievement Award, and the 2003 Mac Van Valkenburg Award of the IEEE Circuits and Systems Society.