A Variable-Radix Digit-Serial Design Methodology and its Application to the Discrete Cosine Transform

M. P. Leong and Philip H. W. Leong

***Abstract*—A variable-radix digit-serial design methodology and its application to the implementation of a systolic structure for computing the discrete cosine transform is presented. Based on the parameters supplied by a user, different fixed-point designs can be derived from a single floating-point description where trade-offs among quantization effects, throughput, latency, and area can be addressed. The resulting hardware implementations have variables of different wordlengths and operators of different radices. This design methodology enables efficient exploration of a complex design space to determine the most suitable implementation for a particular application.**

***Index Terms*—Design automation, design methodology, discrete cosine transforms, field-programmable gate arrays (FPGAs), fixed-point arithmetic.**

I. Introduction

THE discrete cosine transform (DCT), proposed by Ahmed *et al.* in 1974 [1], has become an increasingly important tool for image and video signal processing applications due to its utility and its adoption in standards such as Joint Photographic Experts Group (JPEG), Moving Picture Experts Group (MPEG), and CCITT H.261 [2]–[4]. Compared with other orthogonal transforms, the performance of the DCT is very similar to the optimal Karhunen–Loeve transform (KLT) for highly correlated data [5], [6]. Its prominence makes hardware implementations important, particularly in high performance and low-power applications.

In this paper, we propose a variable-radix digit-serial design methodology and its application to the implementation of the DCT. This methodology allows many different designs to be derived from a single algorithmic description. In particular, trade-offs among quantization effects, throughput, latency, and area can be addressed and a large and complex design space can be explored to determine the most suitable implementation for a particular application. The resulting hardware implementation is quite different from those produced by human designers since each variable may have a different wordlength and each operator may have a different radix, a task which would be too tedious for most designers. Such a design process poses two main challenges: appropriate wordlengths must be determined for each variable used in the DCT and the radix must be selected for each of the operators subject to user's requirements for throughput, latency, and area [7]. To address the former problem, an optimization based method for deriving a fixed point implementation via simulation was used [8]–[10]. The latter issue was addressed by using an automatic synthesis approach. The entire design process is automated via a high-level compilation tool called fp. This tool translates a fp algorithmic description of the DCT (based on an algorithm by Liu *et al.* [11]) into a fixed-point, variable-radix, digit-serial dataflow architecture. The resulting synthesizable VHDL code is then used to produce a bitstream for a field programmable gate array (FPGA).

We have successfully generated a series of DCT implementations with different radices and wordlengths from a single description. These implementations offer almost continuous trade-offs among performance, area, latency, and throughput. As an example, the DCT implementations were applied to 2-D image coding, from which the signal-to-noise ratios (SNRs) produced by the implementations are visualized.

The rest of the paper is organized as follows. In Section II, background concerning the DCT algorithm, floating-point to fixed-point translation and digit-serial computation are reviewed. In Section III, a description of the design environment and its design flow are given. In Section IV, synthesis and measured performance results are presented. This is followed in Section V with a discussion of the experimental results and conclusions are drawn in Section VI.

II. Background

A. Hardware Implementations of the DCT

Previously reported hardware implementations of the DCT in VLSI technology can be divided into two main categories: high-throughput and low-power. High-throughput designs include the 100 million samples per second, 100-MHz VLSI implementation in 0.8 μm CMOS technology reported by Uramoto *et al.* [12] and the 150×10^6 samples per second, 150-MHz VLSI implementation in 0.3 μm CMOS technology reported by Kuroda *et al.* [13]. An example of a low-power design is the VLSI implementation in 0.6 μm CMOS technology by Xanthopoulos and Chandrakasan [14], which dissipates 4.38 mW at 14 MHz and 1.56 V and has a maximum performance of 43×10^6 samples per second at 43 MHz. Hunter and McCanny proposed a system which takes parameters such as point size and coefficient wordlengths and generates an efficient design for VLSI synthesis [15].

FPGA devices have also been used to implement the DCT. A Xilinx XC6200 series-based implementation was reported by Trainor *et al.* [16], which achieves a performance of 15.36×10^6 pixels/sec for 2-D DCT image coding. Bergmann and Chung reported an implementation on a Xilinx XC4010 FPGA [17] which achieves 6.21×10^6 pixels/s for 2-D DCT image coding using the Fast DCT algorithm and 12.44×10^6 pixels/sec using

Manuscript received August 26, 2000; revised July 2, 2001; revised January 5, 2002.

The authors are with the Department of Computer Science and Engineering, The Chinese University of Hong Kong, Shatin, N.T., Hong Kong (e-mail: mpleong@clc.cuhk.edu.hk).

Digital Object Identifier 10.1109/TVLSI.2003.811099

distributed arithmetic. Kropp *et al.* proposed a generator for pipelined multipliers on FPGAs and applied their work on the DCT [18]. A 2-D DCT processor on an Altera FLEX10K100 FPGA was reported by Mohd–Yusof, Suleiman and Aspar [19], which achieves a throughput of 5.53×10^6 pixels/s at a clock rate of 11 MHz. Naviner *et al.* reported a high accuracy implementation of the DCT on an Altera FLEX10K50 FPGA [20].

There are three main strategies which have been employed in high-performance implementations of the DCT. The number of arithmetic operations, particularly multiplications, can be reduced by converting the DCT to skew-circular convolutions [21] or a direct sum of matrices [22]; parallelism can be maximized by systolic structures [23]–[25] or distributed arithmetic [17], [26]–[28], and, finally, the hardware resources required for arithmetic operations can be reduced by applying appropriate approximations such as replacing multiplications by sequences of shift-and-adds [29], [30].

Liu *et al.* [11] proposed an approach which utilizes all three of the above strategies by transforming the DCT computation into a systolic computation involving discrete moments (DM). The bulk of computation is performed using addition. Specifically, the algorithm permits minimal area hardware implementations with reasonable accuracy and the systolic structure maps well to the adder and register-rich architecture of an FPGA. This algorithm centers around the relationship between the DCT and DM and a brief description follows.

The N-point DCT is defined by

$$X(k) = c_k \sum_{n=0}^{N-1} x(n) \cos \frac{\pi(2n+1)k}{2N}, \quad 0 \le k \le N-1 \quad (1)$$

where c_k equals $1/\sqrt{N}$ if $k = 0$, $\sqrt{2/N}$ otherwise. Define $S(k,i)$, $s(k,i)$ and $x_{k,i}$ $(i,k = 0,1,2,\ldots,N-1)$ by (2), shown at the bottom of the page. By substituting $x_{k,i}$ into (1), for a given value of p, the N-point DCT can be transformed to

$$X(k) = c_k \left(x_{k,0} + \sum_{r=0}^{p} a_r m_{k,2r} \right) + R_p,\ 0 \le k \le N-1 \quad (3)$$

where

$$\begin{aligned} a_r &= \frac{(-1)^r \pi^{2r}}{(2N)^{2r}(2r)!},\ m_{k,2r} = \sum_{i=1}^{N-1} x_{k,i} i^{2r}, \\ R_p &= c_k \sum_{i=1}^{N-1} x_{k,i} \cos \frac{\xi_i + (2p+1)\pi}{2} \frac{(\frac{\pi i}{2N})^{2p+1}}{(2p+1)!} \\ 0 &\le \xi_i \le \frac{\pi i}{2N}. \end{aligned} \quad (4)$$

Equation (3) establishes the relationship between the DCT and DM. If the $2r$-order moments, $m_{k,2r}$, $(k = 0,1,\ldots,N-1; r = 0,1,\ldots,p)$ are available, each $X(k)$ $(k = 0,1,\ldots,N-1)$ can be readily computed. R_p is a Taylor remainder term, which rapidly converges to zero as p increases.

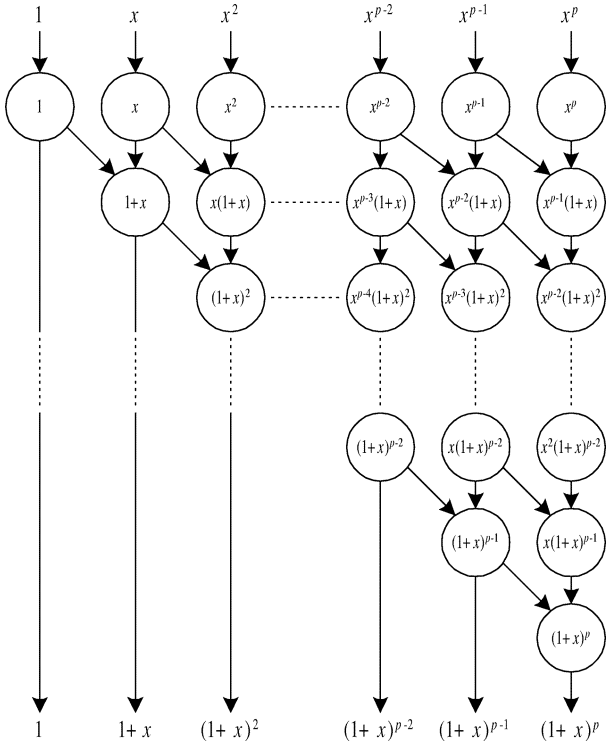


Fig. 1. The p-network with input vector $(1, x, x^2, \ldots, x^{p-1}, x^p)$, nodes represent additions.

As proposed by Liu *et al.* [11], to compute $m_{k,2r}$, a transformation known as F_p can be used. The transformation is conducted by an architecture called the p-network which resembles a Pascal's triangle. Fig. 1 illustrates a p-network for the transformation

$$\begin{aligned} &F_p(1, x, x^2, \ldots, x^{p-1}, x^p) \\ &\quad = \left(1, (1+x), (1+x)^2, \ldots, (1+x)^{p-1}, (1+x)^p\right). \end{aligned}$$

By recursive application of F_p, the p-order moment, $m_p(a_n, a_{n-1}, a_{n-2}, \ldots, a_2, a_1)$, can be computed

$$\begin{aligned} m_p =& F_p\left(F_p \cdots \left(F_p\left(F_p\left(F_p\left(\mathbf{a}_n\right) + \mathbf{a}_{n-1}\right) + \mathbf{a}_{n-2}\right)\right.\right. \\ &\left.\left. + \cdots + \mathbf{a}_2\right) + \mathbf{a}_1\right) \\ =& F_p^{n-1}\left(\mathbf{a}_n\right) + F_p^{n-2}\left(\mathbf{a}_{n-1}\right) + F_p^{n-3}\left(\mathbf{a}_{n-2}\right) \\ &+ \cdots + F_p^2\left(\mathbf{a}_3\right) + F_p\left(\mathbf{a}_2\right) + \mathbf{a}_1 \\ =& \left(\sum_{i=1}^{n} a_i, \sum_{i=1}^{n} a_i i, \sum_{i=1}^{n} a_i i^2, \ldots, \sum_{i=1}^{n} a_i i^{p-1}, \sum_{i=1}^{n} a_i i^p\right). \end{aligned}$$

$$\begin{aligned} S(k,i) &= \left\{ j \middle| \cos \frac{\pi(2j+1)k}{2N} = \cos \frac{i\pi}{2N},\ 0 \le j \le N-1 \right\} \\ s(k,i) &= \left\{ j \middle| \cos \frac{\pi(2j+1)k}{2N} = -\cos \frac{i\pi}{2N},\ 0 \le j \le N-1 \right\} \\ x_{k,i} &= \begin{cases} \sum\limits_{j \in S(k,i)} x(j) - \sum\limits_{j \in s(k,i)} x(j), & \text{if } S(k,i) \cup s(k,i) \ne \emptyset \\ 0, & \text{otherwise.} \end{cases} \end{aligned} \quad (2)$$

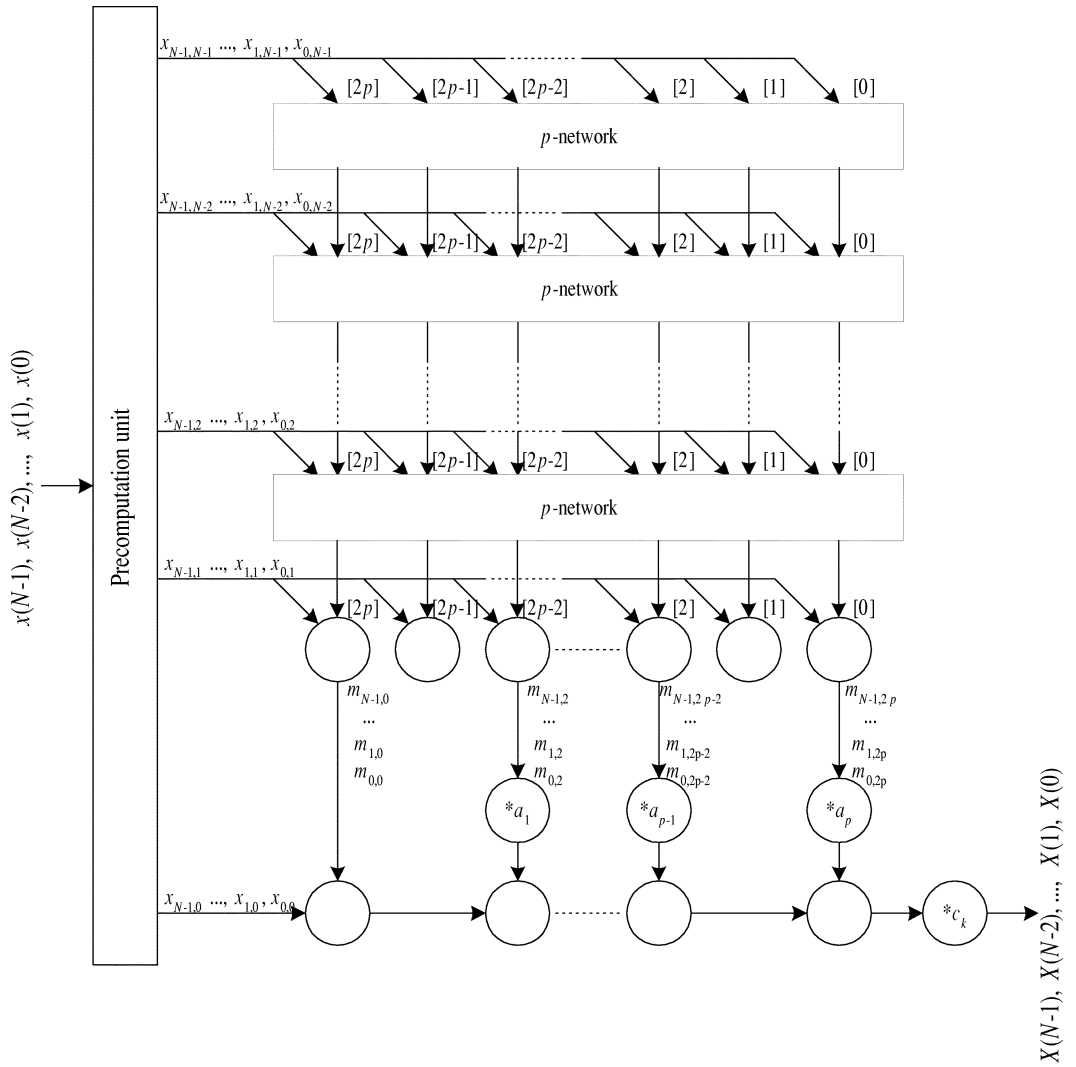


Fig. 2. The systolic array for 1-D N-point DCT, with p-network blocks as shown in Fig. 1 and blank nodes represent additions.

The relationship between the DCT and DM decomposes the computation of DCT into three steps.

1) Precomputation: $x_{k,i}$ $(i,k = 0,1,2,\ldots,N-1)$.
2) Computation of DM: $m_{k,2r}$ $(r = 0,1,2,\ldots,p;\ k = 0,1,2,\ldots,N-1)$.
3) Postcomputation: $X(k)$ $(k = 0,1,2,\ldots,N-1)$.

Computation of $x_{k,i}$ from x_k and $X(k)$ from $m_{k,2r}$ are done using (2). Computation of the DM is implemented using $N-1$ p-networks each with $2p+1$ adders at the top level and $2p+1$ adders at the output of the cascaded p-network. Fig. 2 shows the overall systolic structure for computing the 1-D DCT. Finally, $X(k)$ is computed from (3) where a_r and c_k are precomputed constants and hence its implementation can use a constant multiplier.

The structure offers a tradeoff between area and accuracy. The choice of p should be around two to four for a practical hardware implementation. Hardware requirements and the associated error term R_p for some typical values of p and N are listed in Table I. The numbers of adders and multipliers refer to the operators in the systolic array only and does not account for those in the precomputation and postcomputation units. R_p was calculated by assuming input values in the range of $[-8.0, 8.0)$.

TABLE I
THE NUMBER OF ADDERS AND MULTIPLIERS AND THE ASSOCIATED ERROR BOUND FOR DIFFERENT VALUES OF p AND N

p	N	Number of adders	Number of multipliers	R_p	$R_p/\max\|x\|$
2	4	37	4	1.80324	0.22541
2	8	97	4	2.55016	0.31877
2	16	217	4	3.60648	0.45081
2	32	457	4	5.10033	0.63754
3	4	66	5	0.10594	0.01324
3	8	178	5	0.14982	0.01873
3	16	402	5	0.21187	0.02648
3	32	850	5	0.29963	0.03745
4	4	103	6	0.00363	0.00045
4	8	283	6	0.00513	0.00064
4	16	643	6	0.00726	0.00091
4	32	1363	6	0.01027	0.00128

B. Floating to Fixed Point Translation

Fixed-point implementations are characterized by the assignment of a fixed wordlength and a fixed exponent to each variable. This is in contrast with a floating-point implementation in which wordlengths of variables remain fixed but the exponents

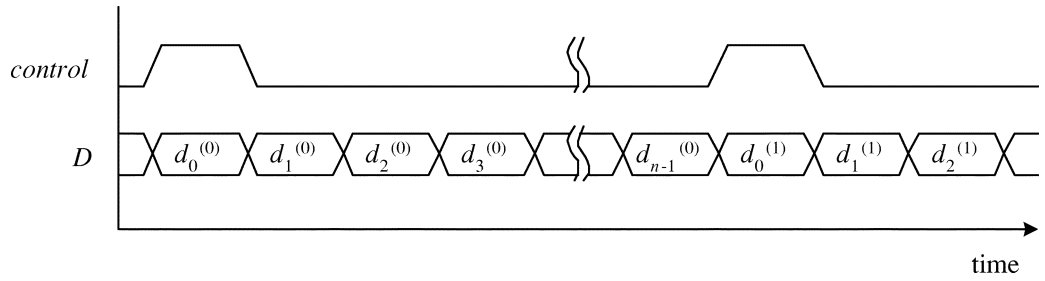


Fig. 3. Dataflow and control mechanisms in digit-serial architectures.

can change at runtime. Fixed-point hardware implementations are usually preferred over floating-point implementations since they are more favorable in terms of hardware resources, cost, design complexity, latency, and power consumption, especially for those designs having variables with small dynamic range.

The major difficulty in designing a fixed-point implementation arises from quantization effects. Quantization occurs naturally when a finite wordlength is used to represent a real number. A typical design process begins with a floating-point algorithmic description, each floating-point variable being translated to fixed-point. To ensure the results obtained are correct, designers observe the dynamic ranges and errors of variables and specify sufficient numbers of bits for each fixed-point variable. Such a translation process is tedious but worthwhile for gaining an area reduction over a floating-point implementation.

Commercial fixed-point design environments are mainly targeted for fixed-point digital signal processing (DSP) chips in which size of a word is fixed. An example of these environments is mentor graphics data flow language (DFL) [31]. These tools, however, may not be optimal for FPGA implementations in which there is no need for each variable to have the same wordlength and an unnecessarily long wordlength for some variables may result. Sung and Kum proposed an approach based on dataflow descriptions made using block diagrams [32]. However, the problem of converting an algorithm into such a description must be addressed.

Other fixed-point design environments include FRIDGE by Willems *et al.* [33], [34] and a fixed-point optimization utility by Kim *et al.* [9]. These design environments are based on general programming languages, such as C and C++. Both introduce new datatypes or classes on top of the programming language for simulating fixed-point arithmetic with configurable precision. Later, the authors proposed a framework also built on top of C++, which automatically translates arbitrary floating-point algorithmic description into synthesizable fixed-point VHDL [very high speed integrated circuit (VHSIC) hardware discription language] descriptions [10] and this approach was used in the work described here.

C. Digit-Serial Computation

In a bit-parallel architecture, a k-bit variable requires an k-bit wide datapath for its transmission and at any clock cycle all the n data bits appear on their associated wires to form a two's complement representation of the value. Bit-parallel arithmetic operators process the entire variable in a single clock cycle.

Instead of processing the entire word of a variable in a single cycle, digit-serial operations are carried out in multiple clock cycles, a digit being processed each cycle. As an example, Fig. 3 illustrates a digit-serial representation and its associated control signal. The control signal is set high when the first digit d_0 of the n-digit variable D is being transmitted. Bit-serial and bit-parallel architectures are considered as special cases of digit-serial architectures where the digit sizes are respectively one and the maximum wordlength among all variables.

Digit-serial systems offer area reduction and higher clock rate compared with bit-parallel architectures at the expense of increased latency and perhaps decreased throughput [7], [35], [36].

III. Design Environment

The design environment consists of a compiler that translates algorithmic descriptions in infix form to a directed acyclic graph (DAG) representation in C++; a C++ library that models fixed-point arithmetic (specifically quantization and computation errors) under different wordlength configurations; an optimizer that takes sample input vectors, passes the values to the algorithm for simulation under different wordlength and radix configurations and minimizes area or errors at outputs; a variable-radix, variable-wordlength module library that provides area, latency, and throughput estimations to the optimizer and generates VHDL descriptions of the modules; and a VHDL generator that takes wordlengths and radices determined by the optimizer and calls module library to generate VHDL description of the whole design [10].

Fig. 4 illustrates the design flow. The primary input taken by the system is the algorithmic description written in the C programming language. The input is processed by an infix expression compiler to generate a DAG representing the dataflow of the algorithm. The DAG description is compiled along with two libraries, namely the fixed-point library and the optimizer library, to generate an executable for wordlength and radix optimization. To perform optimization, the executable simulates the algorithm with the sample input vectors, varying wordlengths, and radices to reduce a cost function which incorporates quantization error, area, latency, and throughput measures. As output, the executable produces the configuration of radices and wordlengths that minimizes the cost function while satisfying user-specified constraints. The VHDL description generator takes the configuration and requests the module library for VHDL descriptions of the modules being used in the design. At this step, proper control circuitry is

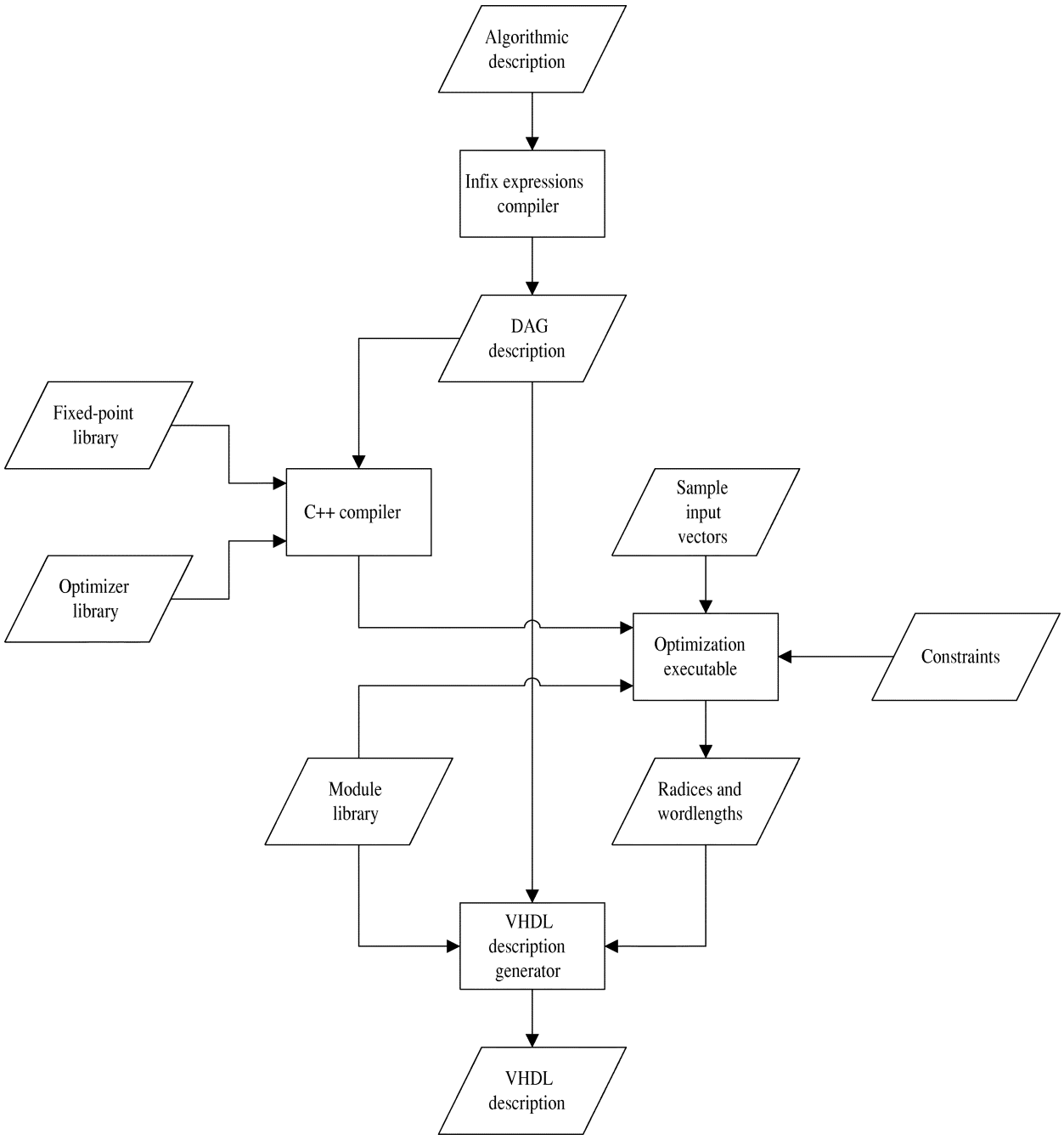


Fig. 4. Design flow of the fixed-point design environment.

also inserted into the design. The overall output of the design flow is a fixed-point VHDL description that implements the algorithm. As the final step, the description is simulated and/or synthesized using commercial CAD software.

A. Compilation

The design environment takes a set of infix expressions as input. The syntax used is a subset of the C programming language which does not contain any transfer of control statements.

The input is parsed and converted to a DAG using standard compilation techniques. As an example, let a and b be inputs and c and d be outputs. Then the sequence of infix expressions $c = a + b$, $d = a \times b$, $c = c \times a$, $d = d + b$ is translated to the DAG shown in Fig. 5.

The output of the compiler is a C++ function that instantiates operators as objects and defines the connections between them. As an example, the compiler would generate the following segment of C++ code for the DAG in Fig. 5

```
FIn fIn0("fIn0", "a")
FIn fIn1("fIn1", "b")
FAdd fAdd0("fAdd0", "fIn0", "fIn1")
FMul fMul0("fMul0", "fIn0", "fIn1")
FMul fMul1("fMul1", "fMul0", "fIn0")
FAdd fAdd1("fAdd1", "fAdd0", "fIn1")
FOut fOut0("fOut0", "fMul1", "c")
FOut fOut1("fOut1", "fAdd1", "d")
```

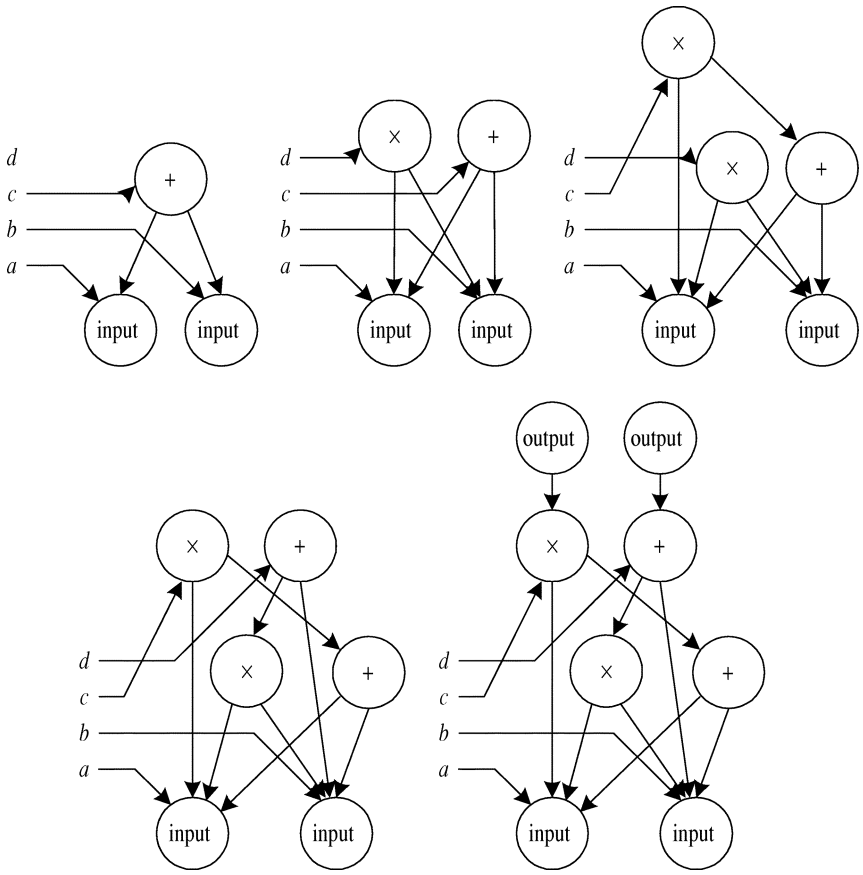


Fig. 5. Example of building a DAG from infix expressions.

where `FIn`, `FAdd`, `FMul` and `FOut` are C++ classes that model the operators (inputs, addition, multiplication and outputs, respectively). The first argument of each constructor is a unique operator name. For `FAdd` and `FMul`, the subsequent arguments are operands. The second argument for `FOut` specifies the output operator and for `FIn` and `FOut`, the last argument specifies the port name.

B. Variable Radix and Wordlength Architecture

In traditional digit-serial implementations, variables have a common fixed radix or digit size. In general, the wordlength of a variable is the product of its number of digits and its radix. When combining the traditional digit-serial architecture and the variable wordlength technique, a number of problems arise.

- If the deviation of variable wordlengths is large, hardware resources may not be fully utilized in all clock cycles. This is because arithmetic operations on a variable with a large number of digits requires more cycles to complete, but operators on variables with smaller number of digits require less cycles, resulting in idle cycles.
- The throughput of a system is limited by the variable with the largest number of digits. In a conventional digit-serial system, variables with higher precisions require larger wordlength and hence more digits. To improve precision by increasing the wordlength of the variable, either the number of digits or radix must be increased. Increasing the number of digits would reduce the throughput, while increasing the radix would result in large area overhead. This is because in a traditional digit-serial architecture, the same radix is used for all variables, so all other variables would also have increased radices and area requirements.

To overcome these problems, the implementations generated by fp employ a novel architecture where each variable can have a different wordlength and radix. The representation of a variable can be of an arbitrary number of bits and is parameterized as a triplet (f, w, d), where f is the fractional wordlength, w is the total wordlength and d is the radix. All variables have the same number of digits which is specified by a global parameter n. In general, $w \geq 0$, f is an integer and $d \geq \lceil w/n \rceil$. The least significant bit of a variable has a partial value of 2^{-f}. When $d > w/n$, sign extension is applied to get the most significant bits of the most significant digit. The representation of a variable with precision format (5,22,8) and $n = 3$ is the equation shown at the bottom of the next page.

The subscripts of b refer to their exponents in base 2. For instance, if the most significant bit of digit 1 is one, its partial value is $2^{10} = 1024$. Since the variable has 22 bits but 24 bits (3 digits $\times 8$ bits) are required in the digit-serial implementation, two sign-extension bits are inserted in the most significant digit. To summarize our approach:

- the number of digits n is common to all variables throughout the system; this ensures all arithmetic operators have the same throughput and maximal resource utilization is guaranteed;

- since all operators have the same number of digits, they take the same number of cycles to complete and, hence, no bottlenecks can occur;
- in contrast to the traditional digit-serial architecture, radices of individual variables can be increased or decreased independently and, hence, a more cost-effective tradeoff between performance and resource requirements can be attained;
- a traditional digit-serial architecture is a special case of this approach where the number of digits and radix are the same for all variables in the system.

Similar to traditional digit-serial architectures, every variable in the variable-radix variable-wordlength architecture has an associated control signal. The control signal is asserted when the first digit of the variable is being transmitted over the associated data wires. Bit-parallel and bit-serial implementations are two special cases, the former case having $n = 1$ and the latter having n equal to the maximum wordlength.

C. Fixed-Point Library

As mentioned in Section III-A, algorithmic descriptions are first translated into C++ code, which instantiates objects to model the fixed-point arithmetic operations. The fixed-point library consists of a collection of C++ classes, each of which models an operator. Each fixed-point arithmetic object contains an arithmetic core for the real fixed-point computation, which has N_i inputs and N_o outputs. For instance, an addition object has $N_i = 2$ and $N_o = 1$.

Each of the inputs or outputs contains information regarding precision format, runtime and worst-case error statistics and range estimation. Their precision formats are controlled by the arithmetic core associated with it. Each arithmetic core has a number of public accessible parameters. The optimizer can, hence, control the precision of the fixed-point arithmetic and the precision formats of the inputs or outputs by modifying these parameters. The inputs and outputs are respectively connected to outputs and inputs of other fixed-point arithmetic objects. The overall inputs and outputs of the design are defined by two special classes of nodes, where $N_i = 0$ and $N_o = 0$ respectively.

When the parameters of a fixed-point arithmetic object are modified, the arithmetic precision of the core and the precision formats associated with the inputs and outputs are updated. Worst-case error analysis and range estimation is subsequently performed. After the parameters have been changed, the fixed-point arithmetic object can carry out its computation based on the precisions determined. In a computation cycle, the following steps are performed.

1) Inputs take values from the outputs to which they connect and are truncated or extended if the precision formats between two connected ports are not the same.
2) Arithmetic core performs the required computation and the results are copied to the outputs.
3) Output is truncated or extended if its precision format differs from the output of the arithmetic core.
4) Inputs and outputs record runtime range and error statistics.

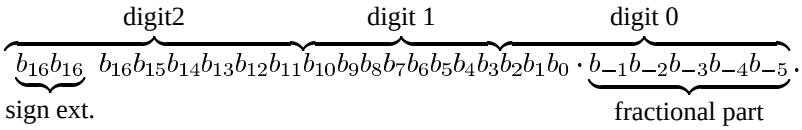
D. Optimization

The objective of the optimization process is to minimize area or errors at the outputs of a design while satisfying user-specified constraints. The optimizer tries different parameters that control the precisions of arithmetic objects, monitors the errors at outputs and estimates the area, evaluates a cost, and finds out the set of parameters that minimizes the cost. The optimizer used the downhill simplex method of Nelder and Mead [37]. This method does not require the computation of derivatives and is suitable for a multidimensional discontinuous search space. Other optimization techniques may lead to a more efficient and accurate search.

In a single pass of the optimization process, the following operations are carried out.

1) Optimizer proposes a set of parameters and the fixed-point arithmetic operators, in turn, modify the number of fractional bits in their inputs and outputs.
2) Sample input vector is taken as the input to the design.
3) Arithmetic operators perform their computation using precisions based on the specified parameters and the number of fractional bits specified for their associated inputs.
4) Wordlengths of all fixed-point variables are computed so that they do not overflow with respect to the runtime ranges.
5) Runtime error statistics and ranges are extracted.
6) If there exist variables with zero wordlengths, the variables and associated operators are trimmed.
7) Latency and throughput are computed and stage latches are inserted where appropriate.
8) Area is estimated.
9) Constraint satisfaction is checked and the cost function is computed if the constraints were satisfied.
10) Optimizer saves the parameters.

As mentioned in Section III-B, the implementation generated by the tool has variable radix. To facilitate arithmetic operations, the digit sizes of the operands must match. For addition and subtraction operations, the fractional precision must also match. To convert a data value from one format to another, conversion modules are automatically generated and inserted between operators whenever necessary. Conversion modules introduce area and latency overhead, so the optimizer takes into account the overheads contributed by the conversion modules.



In most cases, arithmetic modules require all operands to enter at the same time. The process of time alignment is applied when the latencies of the operators are different. During this process, stage latches are inserted such that operands have the same latency after passing through these shift registers. As the latencies of the modules are known, fp can compute the required number of stage latches for every arithmetic module input. Note that the cost function takes into account the area used by stage latches.

E. Module Library

The module library serves two purposes. First, it accepts enquiries from the optimizer and returns area, latency and throughput measures of individual modules and second, it takes requests from the VHDL description generator and generates synthesizable VHDL descriptions.

The modules inside the library are organized hierarchically, with every module dedicated to one arithmetic operation. This hierarchical design enables complicated modules to be implemented using other modules. In addition to the standard arithmetic operations, the module library also has two special subclasses, namely the conversion module and the stage latch module, as described in Section III-D.

F. VHDL Generation

As inputs, the VHDL description generator takes the DAG representation from the compiler and the radices and wordlengths configuration determined by the optimization process. The generator calls the module library accordingly, requesting a VHDL description of each module. It also inserts appropriate sequencing circuits to control module operation. As output, the generator emits VHDL description for simulation and synthesis purposes.

IV. Results

A program was developed to generate sets of infix expressions that describe the DCT algorithm described in Section II-A with different moment order p and number of points N. The algorithm was originally targeted for VLSI where regularity is of utmost importance. However, for VHDL synthesized FPGA implementations, regularity is less important, hence a simplification of the p-network at the top level can be applied. Since the inputs to the p-network at the top level are identical, the outputs of the first level and subsequent adders are the same value as their neighboring adders. Therefore, the removal of all adders except those along the diagonal can be applied. For all of the experiments, $p = 3$ and $N = 8$ was used. The systolic structure thus generated requires 157 adders and five constant multipliers. The sample input vector consists of 500 entries, 200 of which were extracted from image data (normalized to the range $[0.0, 1.0)$), another 200 were the DCT of image data (in the range $[-8.0, 8.0)$) and the remaining were randomly generated (in the range $[-8.0, 8.0)$). We included the 200 entries of transformed image data because for 2-D image coding experiments, the DCT implementation is used twice to encode the input images.

The software generates 35 000 lines of VHDL code from the 95-line description of the DCT algorithm and the average runtime was approximately 1600 s on a 450-MHz Intel Pentium III machine. The synthesis and implementation tools used were Synopsys FPGA Compiler and Xilinx Alliance, respectively. The designs were tested on an Annapolis "Wildstar" Reconfigurable Computing Engine, a PCI64 board containing three Xilinx Virtex XCV1000-6 FPGAs [38]. All results presented in this section were obtained experimentally from a single FPGA on the "Wildstar" board.

TABLE II
EIGHT-DIGIT DCT IMPLEMENTATIONS OBTAINED BY AREA MINIMIZATION WITH DIFFERENT OUTPUT MEAN ERROR CONSTRAINTS

Mean error constraint ($\times 10^{-2}$)	0.1953	0.3906	1.5625	6.2500
Error at output				
Maximum ($\times 10^{-3}$)	0.856	1.672	6.751	22.224
Mean ($\times 10^{-3}$)	0.195	0.219	0.832	2.883
SNR ($\times 10^{2}$)	2.502	1.343	0.579	0.180
Digit size				
Average	2.94	2.64	1.70	1.61
Maximum (rounded up)	7	5	4	4
Minimum (rounded up)	4	2	1	1
Wordlength				
Average	21.24	18.65	14.49	14.28
Maximum	83	42	36	28
Minimum	11	9	8	7
Trimmed operators (ADD/MUL)	0/0	0/0	117/2	117/2
Amount of resources (slices)	1595	1167	691	540
Clock rate (MHz)	69.15	95.56	96.20	122.01
Throughput (cycles)	8	8	8	8
Latency (cycles)	111	85	68	51
Performance ($\times 10^{6}$)	8.64	11.95	12.02	15.25
Performance/area ($\times 10^{3}$)	5.42	10.24	17.40	28.24

Using this approach, a set of 70 implementations of the DCT of different radices and wordlengths were generated from a single description. These implementations had areas ranging from 366 to 4013 Virtex slices, with throughputs between 1.62×10^6 DCT operations per second and 120.50×10^6 DCT operations per second. Applying the DCT implementations to 2-D image coding, the resultant images had SNRs between 19.67 and 44.52 dB.

A. Area Minimization

In these experiments, error constraints were specified and the optimization objective was to minimize the implementation area. In our experiments, mean error constraint were specified. Other types of error constraints supported by the system include maximum error constraints, SNR constraints, and worst-case analysis error constraints.

The results in Table II were obtained by setting n to eight (digit-serial) and varying the error constraints. Results in Table III were obtained by setting the error constraint to $1/256 = 3.9063 \times 10^{-3}$ and varying the number of digits n.

Performance is defined as the number of DCT operations per second and the performance to area ratio indicates the area efficiency. In the tables, trimmed operators refer to those with zero output wordlength and could thus be removed. Operators

TABLE III
DIGIT-SERIAL DCT IMPLEMENTATIONS WITH DIFFERENT DIGIT SIZES OBTAINED BY AREA MINIMIZATION WITH OUTPUT MEAN ERROR CONSTRAINT SET TO 3.9063×10^{-3}

Number of digits, n	1	2	4	6	8	12	16	24	32	48
Error at output										
Maximum ($\times 10^{-3}$)	20.277	18.271	40.161	40.161	16.725	40.161	40.161	36.255	8.558	16.724
Mean ($\times 10^{-3}$)	3.726	2.303	3.621	3.057	2.190	3.324	3.074	3.344	2.241	2.243
SNR ($\times 10^{2}$)	1.301	1.955	1.189	1.278	2.069	1.250	1.359	1.294	2.235	2.037
Digit size										
Average	12.29	7.59	4.02	3.44	2.64	2.14	1.76	1.29	1.08	1.00
Maximum (rounded up)	39	15	9	7	5	3	3	2	2	1
Minimum (rounded up)	6	4	3	2	2	1	1	1	1	1
Wordlength										
Average	21.86	20.33	19.66	19.76	18.65	20.76	22.17	22.67	20.58	21.61
Maximum	39	53	60	48	42	39	42	48	49	48
Minimum	6	9	9	9	9	9	10	9	9	10
Trimmed operators (ADD/MUL)	0/0	0/0	0/0	0/0	0/0	0/0	0/0	0/0	0/0	0/0
Amount of resources (slices)	1769	2146	1553	1373	1167	1188	1369	1049	618	771
Clock rate (MHz)	66.18	64.30	55.85	74.11	95.56	89.60	92.40	96.76	113.11	107.95
Throughput (cycles)	1	2	4	6	8	12	16	24	32	48
Latency (cycles)	46	58	69	83	85	101	104	125	83	94
Performance ($\times 10^{6}$)	66.18	32.15	13.96	12.35	11.95	7.47	5.78	4.03	3.54	2.25
Performance/area ($\times 10^{3}$)	37.41	14.98	8.99	8.90	10.24	6.29	4.22	3.84	5.72	2.92

which only have outputs connect to trimmed operators are also trimmed. Note that the "error at output" entry in the tables refers to fixed-point quantization and its propagated error only and does not include the effects of R_p.

The number of trimmed operators in the implementations in Table III are zero, because to achieve the required output error, all the operators must be preserved. When the error constraint was relieved, some operators could be trimmed which resulted in significant area reduction.

Fig. 6 shows the number of slices (proportional to hardware resource requirements), frequencies, performance and performance to area ratios versus the number of digits under different error constraints. It can be seen from the graphs that the relationship between the number of digits n and area is not straightforward and these graphs can be used to determine the minimal area implementation satisfying certain error and throughput requirements. To more accurately locate the optimal point, both error and throughput constraints should be specified. Table IV shows a series of implementations with a mean error constraint of $1/256 = 3.9063 \times 10^{-3}$ and varying throughput constraints.

B. Image Coding

To further evaluate the digit-serial DCT implementation, we applied the DCT to the coding of several benchmark images. The experimental framework is shown in Fig. 7. Due to the separability of the DCT, the 2-D DCT can be computed using the row-column method. The 1-D DCT was first applied to the input images using the DCT core. The results were then tranposed and another 1-D DCT using the same hardware was applied, followed by another transposition. In this experiment, only the fixed-point 1-D DCTs are performed in hardware and transposition are carried out in software. The 2-D DCT results from the hardware implementation were then correlated with a software fp implementation and the results are shown in Table V. We chose Airfield and applied the inverse DCT on both the hardware and software DCT results and the images are shown in Fig. 8.

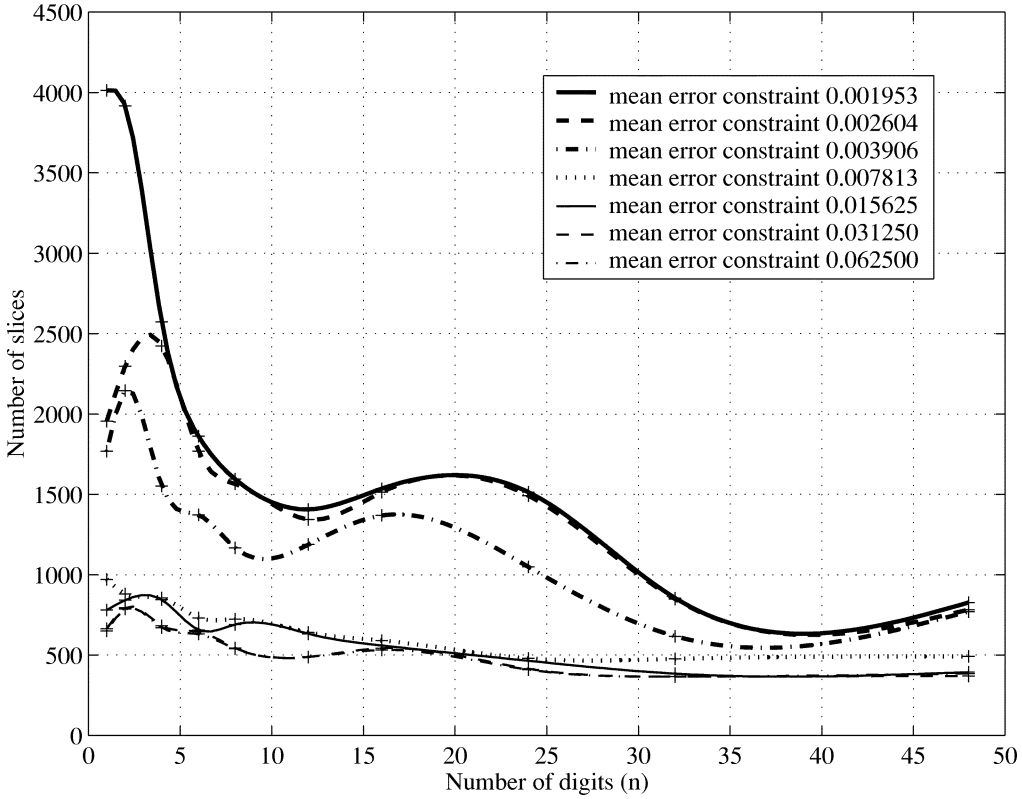
In the table, simulation SNR refers to the SNR based on simulation of sample input vectors; actual SNR refers to the SNR based on comparison of images obtained from fp and fixed-point implementations and is measured in decibels. Simulation SNR and actual SNR were measured separately because during simulation (Section IV-A) the errors contributed by R_p were neglected, whereas errors contributed by R_p are included in the actual SNR. Note that the image coding experiments require two 1-D DCTs and hence have twice the fixed-point errors of the 1-D case.

V. DISCUSSION

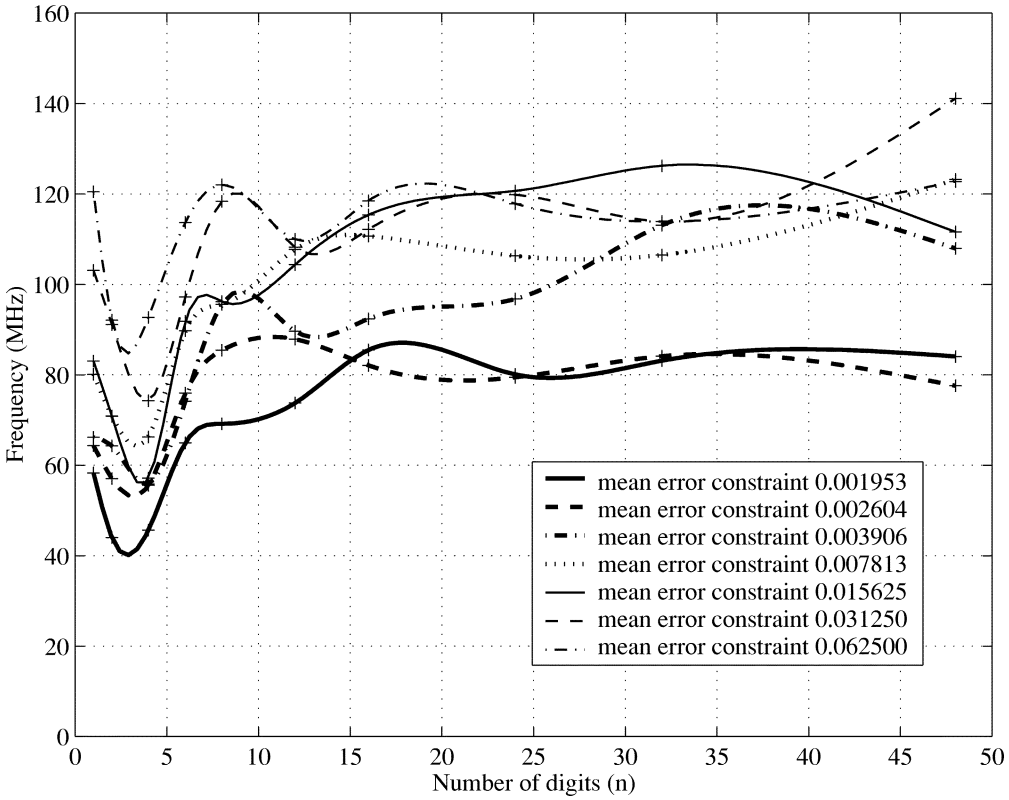
In general, implementations employing variable-radix variable-wordlength architectures have similar properties to traditional digit-serial implementations. For instance, the effects of increasing number of digits n are reduced area, increased clock rate and increased latency. However, the plots in Section IV show that relationships among area, frequency, performance and area efficiency are complex. We analyzed the data in an attempt to understand the cause of such behavior.

A. Tradeoff Between Area and Number of Digits

In Fig. 6(a) a general trend of decreasing area with increasing n can be observed. The insertion of conversion modules for variable format conversions resulted in area overhead when the number of digits n was increased from 1 (bit-parallel). Therefore, for $n = 2$, resource requirements are usually greater than



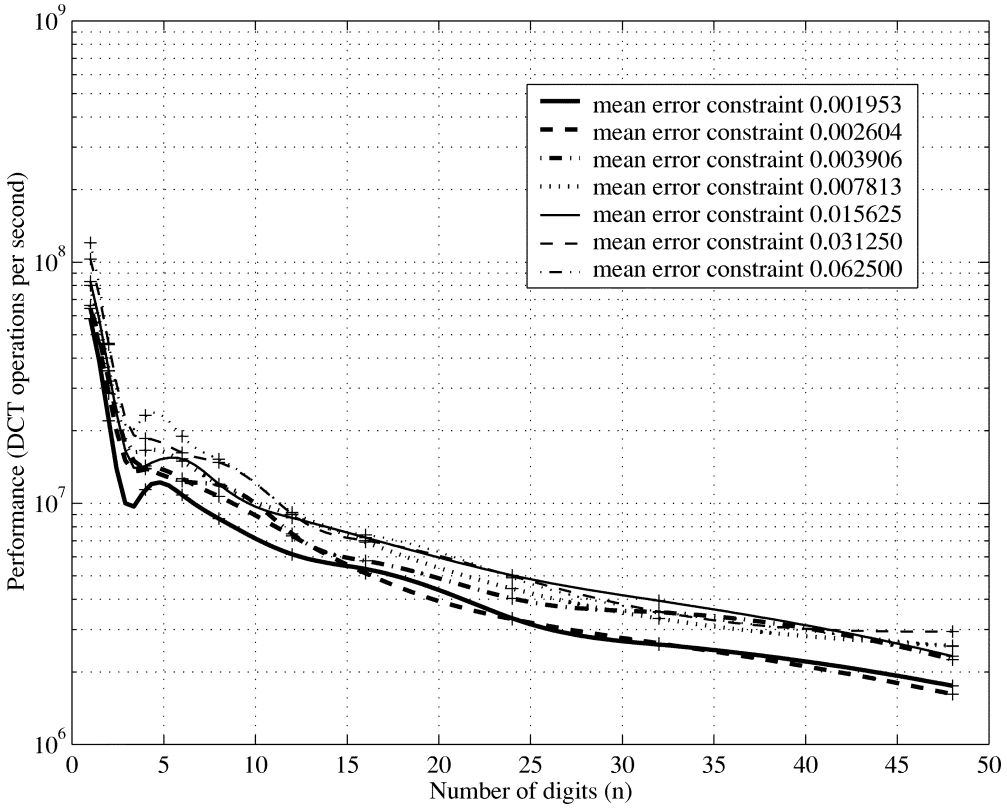
(a)



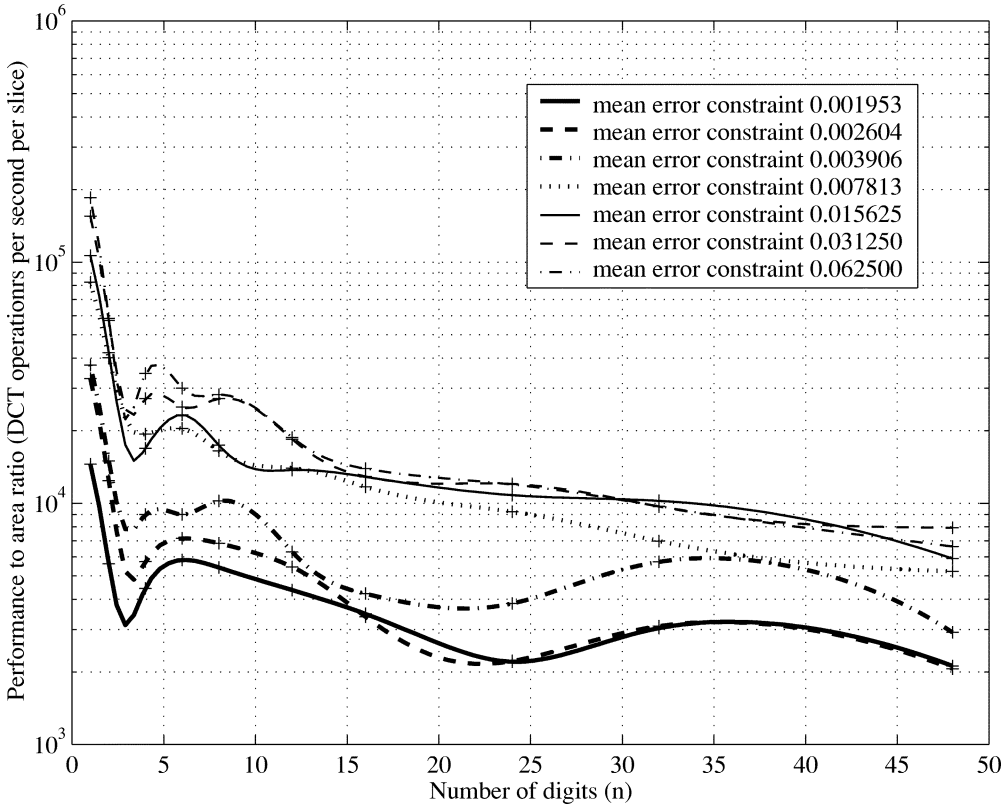
(b)

Fig. 6. Plots of (a) number of slices and (b) frequencies.

$n = 1$. To achieve a net decrease in resource requirements, n must be further increased so that area reduction achieved by folding operators into a lower radix can compensate for the conversion module overhead.



(c)



(d)

Fig. 6. (Continued.) Plots of (c) performances and (d) performance to area ratios against the number of digits under different error constraints.

When the implementation was bit-serial, the decrease in area is significant since conversion modules can be implemented solely with registers and do not require multiplexers. As can be seen from Fig. 6(a), when the specified n is larger than

TABLE IV
DCT IMPLEMENTATIONS OBTAINED BY AREA MINIMIZATION UNDER DIFFERENT THROUGHPUT CONSTRAINTS AND OUTPUT MEAN ERROR CONSTRAINT 3.0963×10^{-3}

Throughput constraint (cycles)	8	16	32	64
Error at output				
Maximum ($\times 10^{-3}$)	16.724	16.370	22.349	16.370
Mean ($\times 10^{-3}$)	2.190	2.552	3.087	3.046
SNR ($\times 10^{2}$)	2.069	1.918	1.433	1.554
Digit size				
Average	2.64	2.64	1.08	1.00
Maximum (rounded up)	5	3	2	1
Minimum (rounded up)	2	1	1	1
Wordlength				
Average	18.65	22.52	21.12	19.57
Maximum	42	42	44	39
Minimum	9	8	8	9
Trimmed operators (ADD/MUL)	0/0	0/0	0/0	0/0
Amount of resources (slices)	1167	1158	606	599
Clock rate (MHz)	95.56	88.09	111.98	120.53
Throughput (cycles)	8	14	30	39
Latency (cycles)	85	99	84	84
Performance ($\times 10^{6}$)	11.95	6.29	3.73	3.09
Performance/area ($\times 10^{3}$)	10.24	5.43	6.16	5.16

that required for a bit-serial implementation, area increases because unnecessarily long stage latches are inserted into the circuit.

B. Tradeoff Between Clock Rate and Number of Digits

We observed from Fig. 6(b) that the maximum clock frequency of the resulting implementations generally increases with increasing number of digits n. An exception is when $2 \leq n \leq 4$, a drop of maximum frequency is noticed.

Digit-serial architectures with smaller radix often offer higher clock rates due to shorter ripple-carry chains. The variable-radix variable-wordlength architecture has a similar property but it can be seen in Fig. 6(b) to be a weak function of n. Since is that conversion modules become more complicated with increased n, requiring more control circuitry and becoming a bottleneck. Several optimization techniques such as pipeline stage insertion have been implemented, but it is still difficult to establish a relationship between these two parameters as in a traditional digit-serial implementation. The drop in frequency when n is between 2 and 4 is mainly caused by the increased area of conversion modules. Normally for FPGA implementations, routing produces most of the delay in the critical path. Area has a large effect on consuming routing resources, so it affects the maximum frequency.

C. Tradeoff Between Performance and Number of Digits

The variable-radix variable-wordlength architecture has a property that the throughput is controlled by the number of digits n. A variable of n digits takes at least n cycles for its data bits to pass through the associated data wires since it is multiplexed in time.

The overall performance (DCT operations per second) is determined by two factors, the number of clock cycles to complete a DCT operation (or equivalently, the maximum number of cycles between adjacent inputs or outputs) and the maximum clock rate of the implementation. As can be seen from Fig. 6(c), the first factor has the dominant effect on performance, hence it is observed that the performance drops with increased number of digits. The dip in the performance curves in Fig. 6(c) around $2 \leq n \leq 4$ can be explained by the corresponding drop in the frequency discussed in Section V-A.

D. Tradeoff Between Area Efficiency and Number of Digits

Fig. 6(d) shows a plot of area efficiency (DCT operations per second per slice) versus the number of digits. The curves, compared with previous ones, contain more local minima and maxima. This is because area efficiency is determined by two conflicting factors, namely area requirement and performance.

Certainly better area efficiency implies a better implementation, but in practice, implementations that achieve a high area efficiency may not satisfy certain design constraints. For instance, implementations on the left side of the curves may not satisfy area constraints, whereas those on the right side may not satisfy throughput constraints. Therefore, one possible design approach is to analyze the curves and pick an implementation that offers the minimal area while satisfying all the design constraints.

E. Effects of Output Error Constraints

By allowing larger errors at the outputs, the control of fixed-point quantization and computation errors can be relaxed, hence, providing a larger space for area minimization. The curves in Fig. 6(a) shows the area requirements under different error constraints. When the mean error constraint is less than or equal to $1/128 = 7.8125 \times 10^{-3}$, there is a significant reduction in area due to trimming of operators. More precisely, when the error constraint was relaxed say to $1/64 = 1.5625 \times 10^{-2}$, 117 adders and two multipliers are trimmed. The removal of multipliers with small coefficients (for the above case, multipliers with coefficients a_p and a_{p-1}) does not affect the precision of output $X(k)$ with respect to the error constraint, as the partial sums they produce are relatively small. Trimming of these multipliers consequently allowed the removal of their preceding adders hence resulting in significant area reduction.

Trimming of operators not only results in area reduction but also in latency reduction. Since adders in the Pascal's triangles are trimmed, latency in the p-network is shortened, resulting in a significant reduction in the overall latency.

Moreover, the relaxed error constraints allow operators to have reduced precision and hence reduced wordlength, leading to an area reduction. Furthermore, higher maximum clock rates are obtained because operators are simplified. With smaller area and higher clock rate, implementations with relaxed error constraints thus have higher performance and better area efficiency. This can be observed in Fig. 6(b) and (c).

F. Optimization by Simulation

Our approach to optimize an implementation is based on the simulation of sample input vectors. Although the optimization

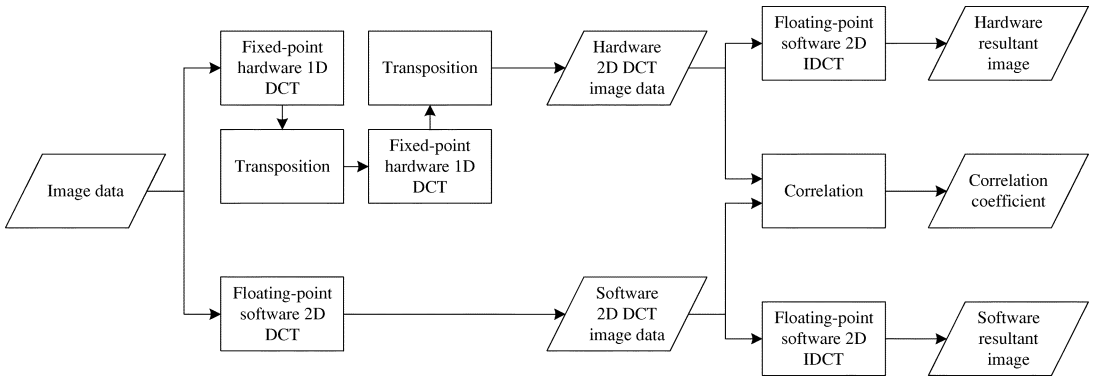


Fig. 7. Framework for image coding experiments.

Fig. 8. Resultant images obtained from software fp and hardware fixed-point DCT image coding, followed by fp inverse DCT. Results from various algorithm performances are shown. The number of slices is measured in a bit-parallel implementation. (simulation SNR, actual SNR, slices). (a) Floating-point DCT. (b) Fixed-point DCT (251.811, 44.550 dB, 4013). (c) Fixed-point DCT (66.271, 32.520 dB, 751). (d) Fixed-point DCT (20.155, 19.664 dB, 651).

procedure considers only runtime error analysis and does not take worst-case analysis into account, it is observed from the image coding experiments that the performance of the algorithm is practical for many applications.

In the experiments the runtime maximum and mean error are in the order of 10^{-2}, but analysis suggests that worst-case error could be up to the 10^{5}, seven orders of magnitude more than runtime error analysis. A worst-case analysis is very pessimistic

TABLE V
CORRELATIONS BETWEEN DCT RESULTS OBTAINED FROM HARDWARE FIXED-POINT AND SOFTWARE FLOATING-POINT IMPLEMENTATIONS UNDER VARIOUS ALGORITHM PERFORMANCES. OUTPUT SNR REFERS TO THE SNR AT THE OUTPUT BASED ON SIMULATION OF THE SAMPLE INPUT VECTOR

Mean error constraint	Output SNR	*Airfield*	*Airplane*	*Bridge*	*Couple*	*Crowd*	*Harbour*	*Lax*	*Lena*	*Man*	*Peppers*
0.001953	251.811	0.9998392	0.9997353	0.9998078	0.9997098	0.9997772	0.9996340	0.9995413	0.9997498	0.9997479	0.9998022
0.002604	212.835	0.9995402	0.9988759	0.9990135	0.9994490	0.9995638	0.9998601	0.9991282	0.9997892	0.9992854	0.9998324
0.003906	118.936	0.9995167	0.9992529	0.9992773	0.9993545	0.9994785	0.9991240	0.9989422	0.9995340	0.9994413	0.9995960
0.007813	66.271	0.9989709	0.9988271	0.9987446	0.9987056	0.9989653	0.9982936	0.9976151	0.9989699	0.9988836	0.9991613
0.015625	42.413	0.9948467	0.9906164	0.9939893	0.9912205	0.9931685	0.9879892	0.9858949	0.9927063	0.9923737	0.9944384
0.031250	24.577	0.9837934	0.9822585	0.9814566	0.9765656	0.9824283	0.9716344	0.9560052	0.9822122	0.9794762	0.9857037
0.062500	20.155	0.9831877	0.9714933	0.9807353	0.9704832	0.9773544	0.9615852	0.9555060	0.9737961	0.9739921	0.9792801

in practice and area requirements can be drastically decreased if optimization is based on runtime error analysis.

VI. CONCLUSION

In this paper, an implementation of a systolic structure for the computation of the DCT using an optimization approach which automatically translates fp algorithmic descriptions into hardware-efficient fixed-point implementations was described. Using a variable radix and wordlength architecture, an almost continuous tradeoffs in performance, area, latency and throughput was obtained, thus allowing designers to choose the most appropriate design for a given application.

Although this paper only addressed the implementation of the DCT, the variable-radix variable-wordlength methodology could be applied to other DSP systems, particularly those in which the tradeoff between area and performance is an important design issue.

REFERENCES

[1] N. Ahmed, T. Natarajan, and K. R. Rao, "Discrete cosine transform," *IEEE Trans. Comput.*, vol. 23, pp. 90–93, Jan. 1974.

[2] D. Le Gall, "MPEG: A video compression standard for multimedia applications," *Cammunications of the ACM (CACM)*, vol. 34, no. 4, pp. 46–58, Apr. 1991.

[3] M. Liou, "Overview of the $p\times$ 64 kbits/s video coding standard," *Communications of the ACM (CACM)*, vol. 34, no. 4, pp. 59–63, Apr. 1991.

[4] W. B. Pennebaker and J. L. Mitchell, *JPEG—Still Image Data Compression Standard*. New York: Van Nostrant Reinhold, 1993.

[5] R. M. Haralick, "A storage efficient way to implement the discrete cosine transform," *IEEE Trans. Comput.*, vol. 25, pp. 333–341, July 1976.

[6] B. G. Lee, "A new algorithm to compute the discrete cosine transform," *IEEE Trans. Acoustics, Speech, Signal Processing*, vol. ASSP-32, pp. 1243–1247, Dec. 1984.

[7] H. Suzuki, Y. N. Chang, and K. K. Parhi, "Performance tradeoffs in digit-serial DSP systems," in *Proc. 32nd Asilomar Conf. Signals, Systems, and Computers*, Pacific Grove, CA, Nov. 1998.

[8] W. Sung and K. I. Kum, "Simulation-based word-length optimization method for fixed-point digital signal processing systems," *IEEE Trans. Signal Processing*, pp. 3087–3090, Dec. 1995.

[9] S. Kim, K. I. Kum, and W. Sung, "Fixed-point optimization utility for C and C++ based digital signal processing programs," *IEEE Trans. Circuit Syst. II*, vol. 45, pp. 1455–1464, Nov. 1998.

[10] M. P. Leong, M. Y. Yeung, C. K. Yeung, C. W. Fu, P. A. Heng, and P. H. W. Leong, "Automatic floating to fixed point translation and its application to post-rendering 3-D warping," in *Proc. IEEE Symp. Field-Programmable Custom Computing Machines*, Napa Valley, CA, Apr. 1999, pp. 240–248.

[11] J. G. Liu, H. F. Li, F. H. Y. Chan, and F. K. Lam, "Fast discrete cosine transform via computation of moments," *J. VLSI Signal Processing*, vol. 19, pp. 257–268, 1998.

[12] S. Uramoto, Y. Inoue, A. Takabatake, J. Takeda, Y. Yamshita, H. Terane, and M. Yoshimoto, "A 100-MHz 2-D discrete cosine transform core processor," *IEEE J. Solid-State Circuits*, vol. 27, pp. 492–499, Apr. 1992.

[13] T. Kuroda, T. Fujita, S. Mita, T. Nagamatsu, S. Yoshioka, K. Suzuki, F. Sano, M. Norishima, M. Murota, M. Kako, M. Kinugawa, M. Kakumu, and T. Sakurai, "A 0.9 V, 150-MHz, 10-mW, 4 mm^2, 2-D discrete cosine transform core processor with variable threshold-voltage (VT) scheme," *IEEE J. Solid-State Circuits*, vol. 31, pp. 1770–1779, Nov. 1996.

[14] T. Xanthopoulos and A. P. Chandrakasan, "A low-power DCT core using adaptive bitwidth and arithmetic activity exploiting signal correlations and quantization," *IEEE J. Solid-State Circuits*, vol. 35, pp. 740–750, May 2000.

[15] J. Hunter and J. McCanny, "Discrete cosine transform generator for VLSI synthesis," in *Proc. IEEE Int. Conf. Acoustics, Speech, and Signal Processing*, vol. 5, Seattle, WA, May 1998, pp. 2997–3000.

[16] D. W. Trainor, J. P. Heron, and R. F. Woods, "Implementation of the 2-D DCT using a Xilinx XC6264 FPGA," in *Proc. IEEE Workshop Signal Processing Systems*, Leicester, U.K., Nov. 1997, pp. 541–550.

[17] N. W. Bergmann and Y. Y. Chung, "Efficient implementation of DCT-based video compression on custom computers," in *Proc. 31st Asilomar Conf. Signals, Systems and Computers*, vol. 2, Pacific Grove, CA, Nov. 1997, pp. 1532–1536.

[18] H. Kropp, C. Reuter, T. T. Do, and P. Pirsch, "A generator for pipelined multipliers on FPGA's," in *Proc. 9th Int. Conf. Signal Processing Applications and Technology*, Toronto, Canada, Sept. 1998, pp. 669–673.

[19] Z. Mohd-Yusof, I. Suleiman, and Z. Aspar, "Implementation of two dimensional forward DCT and inverse DCT using FPGA," in *Proc. Int. Conf. Electrical and Electronic Technology 2000*, vol. 3, Kuala Lumper, Malaysia, 2000, pp. 242–245.

[20] L. Naviner, C. Laurent, J. L. Danger, and A. Garcia-Garcia, "Efficient implementation for high accuracy DCT processor based on FPGA," in *Proc. 42nd Midwest Symp. Circuits and Systems*, vol. 1, Las Cruces, NM, Aug. 1999, pp. 508–511.

[21] W. Lee, "A new algorithm to compute the DCT and its inverse," *IEEE Trans. Signal Processing*, vol. 39, pp. 1305–1313, June 1991.

[22] E. Feig and S. Winograd, "Fast algorithms for the discrete cosine transform," *IEEE Trans. Signal Processing*, vol. 40, pp. 2174–2193, Sept. 1992.

[23] C. Chakrabarti and J. JáJá, "Systolic architectures for the computation of the discrete Hartley and the discrete cosine transforms based on prime factor decomposition," *IEEE Trans. Comput.*, vol. 39, pp. 1359–1368, Nov. 1990.

[24] L. W. Chang and M. C. Wu, "A unified systolic array for discrete cosine and sine transform," *IEEE Trans. Signal Processing*, vol. 39, pp. 192–194, Jan. 1991.

[25] Y. H. Hu and Z. Wu, "An efficient CORDIC array structure for the implementation of discrete cosine transform," *IEEE Trans. Signal Processing*, vol. 43, pp. 331–336, Jan. 1995.

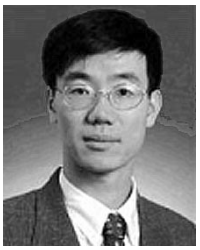
[26] Y. H. Chan and W. C. Siu, "On the realization of discrete cosine transform using the distributed arithmetic," *IEEE Trans. Circuits Syst.*, vol. 39, pp. 705–712, Sept. 1992.

[27] G. Fettweis, J. Chiu, and B. Fraenkel, "A low-complexity bit-serial DCT/IDCT architecture," in *Proc. IEEE Int. Conf. Communications*, vol. 1, May 1993, pp. 217–221.

[28] W. Pan, A. Shams, and M. A. Bayoumi, "NEDA: A new distributed arithmetic architecture and its application to one dimensional discrete cosine transform," in *Proc. IEEE Workshop Signal Processing Systems*, Washington, DC, Sept. 1999, pp. 159–168.
[29] K. Lenwchasatit and A. Ortega, "DCT computation based on variable complexity fast approximations," in *Proc. Int. Conf. Image Processing*, vol. 3, Chicago, Il, Oct. 1998, pp. 95–99.
[30] T. D. Tran, "The BinDCT: Fast multiplierless approximation of DCT," *IEEE Signal Processing Lett.*, vol. 7, pp. 145–149, June 2000.
[31] *DSP Station User's Manual*, Mentor Graphics Corp., 1995.
[32] W. Sung and K. I. Kum, "Word-length determination and scaling software for a signal flow block diagram," in *Proc. IEEE Int. Conf. Acoustics, Speech, and Signal Processing*, Adelaide, Australia, Apr. 1994, pp. 457–460.
[33] M. Willems, V. Bürsgens, H. Keding, T. Grötker, and H. Meyr, "System level fixed-point design based on an interpolative approach," in *Proc. 34th Design Automation Conf.*, Anaheim, CA, June 1997, pp. 293–298.
[34] M. Willems, V. Bürsgens, T. Grötker, and H. Meyr, "FRIDGE: An interactive code generation environment for HW/SW codesign," in *Proc. IEEE Int. Conf. Acoustics, Speech and Signal Processing*, Munich, Germany, Apr. 1997, pp. 287–290.
[35] K. K. Parhi and C. Wang, "Digit-serial DSP architectures," in *Proc. of the Int. Conf. Applications Specific Array Processors*, Princeton, NJ, Sept. 1990, pp. 341–351.
[36] R. Hartley and K. K. Parhi, *Digit-Serial Computation.* Norwell, MA: Kluwer, 1995.
[37] J. Nelder and R. Mead, "A simplex method for function minimization," *IEEE Comput.*, vol. C-7, pp. 308–313, 1965.
[38] *Wildstar Reference Manual*, Annapolis Micro Systems, Inc., Annapolis, MD, 2000.

M. P. Leong received the B.Eng. and Ph.D. degrees from The Chinese University of Hong Kong, in 1998 and 2001, respectively.

He is currently the System Manager of the Center for Large-Scale Computation with The Chinese University of Hong Kong. His research interests include network security, parallel computing, and field-programmable systems.



Philip H. W. Leong received the B.Sc., B.E., and Ph.D. degrees from the University of Sydney, Sydney, Australia, in 1986, 1988, and 1993, respectively.

In 1989, he was a Research Engineer with AWA Research Laboratory, Sydney, Australia. From 1990 to 1993, he was a Postgraduate Student and Research Assistant with the University of Sydney, where he worked on low-power analogue VLSI circuits for arrhythmia classification. In 1993, he was a Consultant to SGS Thomson Microelectronics, Milan, Italy. He was a Lecturer with the Department of Electrical Engineering, University of Sydney from 1994 to 1996. He is currently an Associate Professor with the Department of Computer Science and Engineering, Chinese University of Hong Kong, and the Director of the Custom Computing Laboratory there. He is the author of more than 50 technical papers and three patents. His research interests include reconfigurable computing, digital systems, parallel computing, cryptography, and signal processing.