A Quantitative Analysis of Reconfigurable Coprocessors for Multimedia Applications

Takashi Miyamori
System ULSI Engineering Laboratory
TOSHIBA Corporation, JAPAN
miyamori@sdel.toshiba.co.jp

Kunle Olukotun
Computer Systems Laboratory
Stanford University
kunle@ogun.stanford.edu

Abstract

Recently, computer architectures that combine a reconfigurable (or retargetable) coprocessor with a general-purpose microprocessor have been proposed. These architectures are designed to exploit large amounts of fine grain parallelism in applications. In this paper, we study the performance of the reconfigurable coprocessors on multimedia applications. We compare a Field Programmable Gate Array (FPGA) based reconfigurable coprocessor with the array processor called REMARC (Reconfigurable Multimedia Array Coprocessor). REMARC uses a 16-bit simple processor that is much larger than a Configurable Logic Block (CLB) of an FPGA. We have developed a simulator, a programming environment, and multimedia application programs to evaluate the performance of the two coprocessor architectures. The simulation results show that REMARC achieves speedups ranging from a factor of 2.3 to 7.3 on these applications. The FPGA coprocessor achieves similar performance improvements. However, the FPGA coprocessor needs more hardware area to achieve the same performance improvement as REMARC.

1 Introduction

As the use of multimedia applications increases, it becomes important to achieve high performance on algorithms such as video compression, decompression, and image processing with general-purpose microprocessors. This has motivated the recent addition of multimedia instructions to most general-purpose microprocessor ISAs [1–3]. These ISA extensions work by segmenting a conventional 64-bit datapath into four 16-bit or eight 8-bit datapaths. The multimedia instructions exploit fine grain SIMD parallelism by operating on four 16-bit or eight 8-bit data values. However, a 64-bit datapath limits the speedups to a factor of four or eight even though many multimedia applications have much more inherent parallelism.

Computer architectures that connect a reconfigurable coprocessor to a general-purpose microprocessor have been proposed [4–9]. The advantage of this approach is that the coprocessor can be reconfigured to improve the performance of a particular application. All of these proposed architectures use field programmable gate arrays (FPGAs) for the reconfigurable hardware. We refer to this coprocessor as an "*FPGA coprocessor*" in this paper. The FPGA architecture, which has narrow programmable logic blocks and programmable interconnection network, provides great flexibility for implementing application specific hardware. However, the rich programmable interconnection comes at the price of reduced operating frequency and logic density.

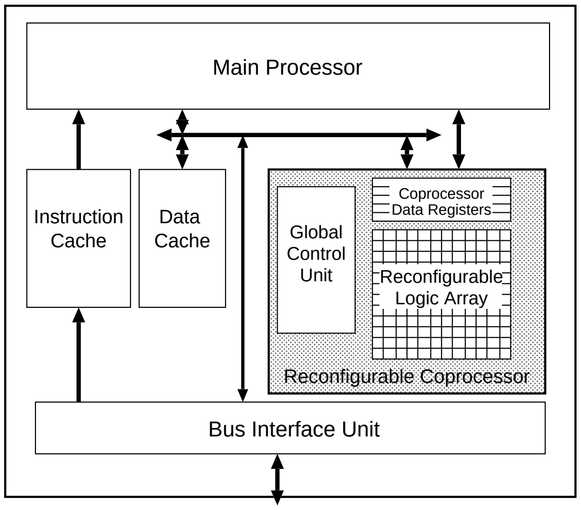
Array processors, such as general-purpose systolic array processors, wavefront array processors [10], PADDI-2 [11], and MATRIX [12], are other reconfigurable architectures. These processors have 8-bit or 16-bit datapaths and each programmable logic block has an 8 to 32-entry instruction RAM that makes it easy to support multiple functions. Because multimedia (or DSP) applications predominantly manipulate 8-bit or 16-bit data values, these architectures work very well on these applications. Recently, we proposed a new array processor architecture called REMARC (Reconfigurable Multimedia Array Coprocessor)[13]. REMARC is a reconfigurable coprocessor that is tightly coupled to a main RISC processor and consists of a global control unit and 64 16-bit simple processors called nano processors.

Both the FPGA coprocessor and REMARC are not limited to SIMD parallelism that can be exploited by multimedia extensions such as the Intel MMX[2]. They can exploit various kinds of fine grain parallelism in multimedia applications. Using more processing resources, they can achieve higher performance than the multimedia extensions. To understand how these two coprocessor architecture compare, in this paper we evaluate the cost and performance of these architectures. The architecture of FPGA coprocessors are still in flux, so we evaluate the performance of the FPGA coprocessor with a varying number of CLBs and vary the cycle time of the FPGA coprocessor from 1x to 10x that of the main processor. For the performance evaluation, we use detailed simulators and two realistic application programs, DES encryption and MPEG-2 decoding. We also estimate the chip sizes of processors with REMARC and the FPGA coprocessor and compare their performance when the same die size is used for both architectures.

The rest of this paper is organized as follows. In Section 2, we describe the reconfigurable coprocessor architectures, both REMARC (array based) and FPGA based. In Section 3, we show the results of our performance evaluation. In Section 4, we estimate chip sizes of processors with REMARC and the FPGA coprocessor. Finally, we conclude in Section 5.

2 Reconfigurable Coprocessor Architecture

2.1 Architecture Overview



Main Processor
Instruction Cache
Data Cache
Global Control Unit
Coprocessor Data Registers
Reconfigurable Logic Array
Reconfigurable Coprocessor
Bus Interface Unit

Figure 1: Block Diagram of a Microprocessor with Reconfigurable Coprocessor

Figure 1 shows a block diagram of a microprocessor which includes a reconfigurable coprocessor. The reconfigurable coprocessor consists of a global control unit, coprocessor data registers, and a reconfigurable logic array. Recently, we proposed the REMARC architecture which includes an 8x8 16-bit processor (nano processor) array as its reconfigurable logic array[13]. The other reconfigurable coprocessor that we consider in this paper, the FPGA coprocessor, uses FPGAs for the reconfigurable logic array. The global control unit controls the execution of the reconfigurable logic array and the transfer of data between the main processor and the reconfigurable logic array through the coprocessor data registers.

We use the MIPS-II ISA [14] as the base architecture of the main processor. The MIPS ISA is extended for the REMARC and the FPGA coprocessor using the instructions listed in Table 1. The main processor issues these instructions to the reconfigurable coprocessor which executes them in a manner similar to a floating point coprocessor. Unlike a floating point coprocessor, the functions of reconfigurable coprocessor instructions are configurable (or programmable) so that they can be specialized for specific applications.

The configuration instructions, *rcon*, *rgcon*, or *rncon*, download the configuration data from memory and store them in the reconfigurable coprocessor. The start address of the configuration data is specified by the value of the source register (*src*). The *rex* instruction starts execution of a reconfigurable coprocessor instruction. The sum of

rcon	*src (rncon or rgcon)*
rex	*cop2_reg, offset(base)*
lduc2	*cop2_reg, offset(base)*
sduc2	*cop2_reg, offset(base)*
mtc2	*cop2_reg, src*
mfc2	*cop2_reg, dst*
ctc2	*cop2_reg, src*
cfc2	*cop2_reg, dst*

Table 1: New Instructions Used in Reconfigurable Coprocessors

the *offset* field and the *base* register specifies one of the operations to execute. The *lduc2* and *sduc2* instructions are load and store coprocessor instructions which transfer double word (64-bit) data between memory and the reconfigurable coprocessor data registers. The *mfc2* and *mtc2* instructions transfer word (32-bit) data between the general-purpose registers (integer registers) in the main processor and the reconfigurable coprocessor data registers. The *cfc2* and *ctc2* instructions transfer data between the integer registers and the reconfigurable coprocessor control registers.

The reconfigurable coprocessors do not have a direct interface to the data cache or memory. The main processor has to set the input data to the coprocessor data registers using *lduc2* and *mtc2* instructions before execution of *rex* instructions. Then, the reconfigurable coprocessor reads the input data, executes the operations, and stores the results into the coprocessor data registers. Finally, the main processor reads the results using *sduc2* and *mfc2* instructions.

2.2 Pipeline Organization

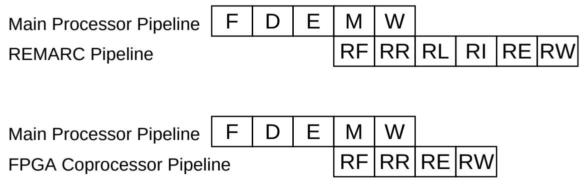


Figure 2: Pipeline Organization of Reconfigurable Coprocessor

The pipeline for REMARC, the FPGA coprocessor, and the main processor are shown in Figure 2. The main processor pipeline is similar to the MIPS R3000 and the MIPS R5000 and consists of five stages: Instruction Fetch (F), Instruction Decode (D), Execution (E), Memory Access (M), Register Write-back (W). The reconfigurable coprocessor pipelines are independent of the main processor pipeline; therefore, the main processor can execute concurrently with the reconfigurable coprocessors.

The REMARC pipeline starts from the M stage of the main processor and has the following six stages:

RF : An instruction of the global control unit is fetched.
RR : The REMARC data registers are read.
RL : The data are aligned or "unpacked".
RI : The instructions of the nano processors are fetched.

RE : The nano processors execute the instructions.

RW : The executed results are packed and stored into the REMARC data registers.

The FPGA coprocessor pipeline has four stages as follows:

RF : The sequencer of the global control unit starts its execution.

RR : The coprocessor data registers are read.

RE : The reconfigurable logic array starts execution.

RW : The execution results are stored into the coprocessor data registers.

The RL and RI stages are unnecessary in the FPGA coprocessor because load alignment or unpack operations are realized directly in the FPGA array and FPGAs do not have instructions to fetch and execute.

2.3 REMARC Architecture

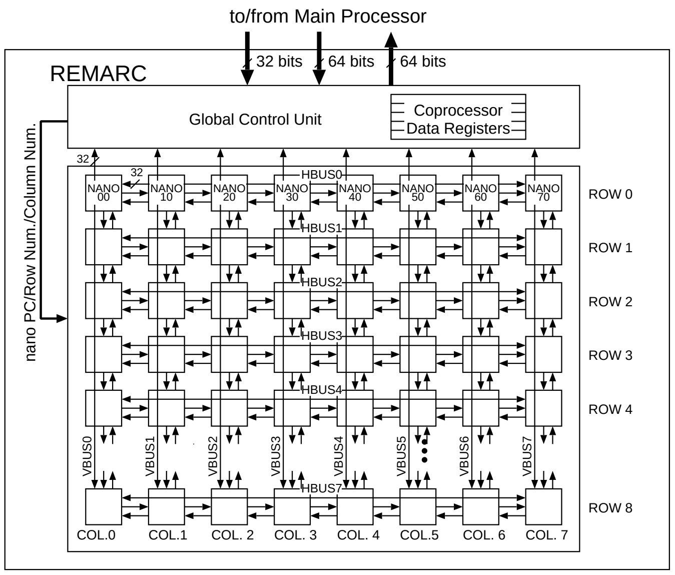


Figure 3: Block Diagram of REMARC

In this section, we briefly describe the REMARC architecture; more information can be found in [13]. Figure 3 shows a block diagram of REMARC. REMARC's reconfigurable logic is composed of an 8x8 array of the 16-bit processors, called nano processors. The execution of each nano processor is controlled by the instructions stored in the local instruction RAM (nano instruction RAM). However, each nano processor does not directly control the instructions it executes. Every cycle the nano processor receives a PC value, "nano PC", from the global control unit. All nano processors use the same nano PC and execute the instructions indexed by the nano PC in their nano instruction RAM.

Figure 4 shows the architecture of the nano processor. The nano processor contains a 32-entry nano instruction RAM, a 16-bit ALU, a 16-entry data RAM, an instruction

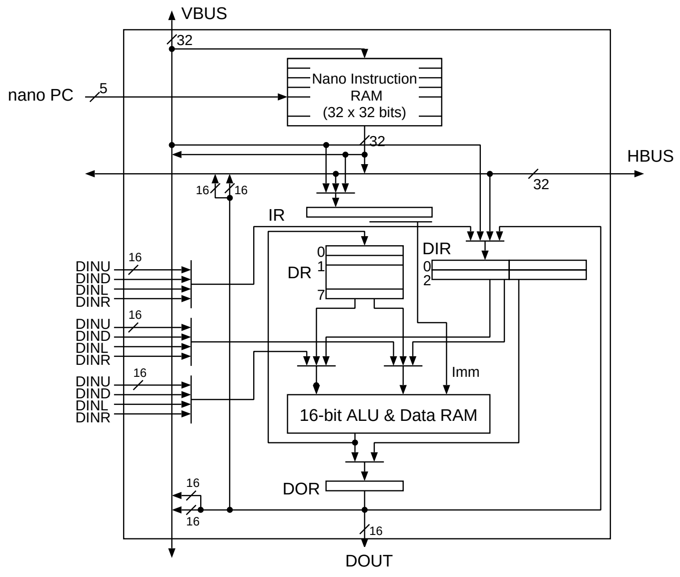


Figure 4: Nano Processor Architecture

register (IR), eight 16-bit data registers (DR), four 16-bit data input registers (DIR), and a 16-bit data output register (DOR).

Each nano processor can use the DR registers, the DIR registers, and immediate data as the source data of ALU operations. Moreover, it can directly use the DOR registers of the four adjacent nano processors (DINU, DIND, DINL, and DINR) as the source.

The nano processors are also connected by the 32-bit Horizontal Buses (HBUSs) and the 32-bit Vertical Buses (VBUSs). Each bus operates as two 16-bit data buses. The 16-bit data in the DOR register can be sent to the upper or lower 16 bits of the VBUS or the HBUS. The HBUSs and the VBUSs allow data to be broadcast to the other nano processors in the same row or column. These buses can reduce the communication overhead between processors separated by long distances.

The DIR registers accept inputs from the HBUS, the VBUS, the DOR, or the four adjacent nano processors. Because the width of the HBUS and the VBUS is 32 bits, data on the HBUS or the VBUS are stored into a DIR register pair, DIR0 and DIR1, or DIR2 and DIR3. Using the DIR registers, data can be transfered between nano processors during ALU operations.

It takes a half cycle to transfer data using the VBUSs or HBUSs. It should not be a critical path of the design. Other operations, except for data inputs from nearest neighbors, are done within the nano processor. Because the width of a nano processor's datapath is only 16 bits, which is a quarter of those of the general purpose microprocessors, this careful design does not make REMARC a critical path of the chip.

2.4 FPGA Coprocessor Architecture

The reconfigurable logic array of the FPGA coprocessor is composed of configurable logic blocks (CLBs). Each CLB

	REMARC	FPGA Coprocessor
Processor Size	Large (16-bit datapath)	Small (two 4-1 LUTs)
Num. of Procs.	Small (64 processors)	Large (500 – 5000 CLBs)
Execution control	Instruction	Hardwired
Communication	Controlled by instruction	Configured by switch matrix
Interconnection	4 neighbors, VBUS, and HBUS	Short wires and long wires
Cycle Time	Tcpu	1x, 2x, 5x, 10xTcpu

Table 2: Coprocessor Architecture Comparison

is equal to a CLB of the Xilinx 4000 series. The CLB includes two 4-1 lookup tables (LUTs) and two flip-flops.

We do not fix the number of the CLBs in the FPGA coprocessor. Instead, we evaluate the performance of the FPGA coprocessor using a varying number of CLBs. The cycle time of the FPGA coprocessor is also parameterized. Cycle times of current FPGA systems are longer than those of the microprocessors by factors of five to ten. For instance, FPGA systems usually operate at 30 to 100 MHz while state-of-the-art microprocessors operate at more than 400 MHz. However, the recently proposed reconfigurable coprocessor Garp [8] aims to operate at the same operating frequency as its main processor. Therefore, we assumed the cycle time of the FPGA coprocessor could be 5x or 10x that of the main processor for current FPGA architectures and 1x or 2x for future FPGA architectures.

2.5 Coprocessor Architecture Comparison

Table 2 summarizes the comparison of the two reconfigurable coprocessor architectures. REMARC has larger processing elements, the nano processors, than the FPGA coprocessor. However, the number of nano processors is less than that of CLBs in the FPGA coprocessor. In REMARC, both the execution and data transfer are controlled by instructions, while these are controlled by hardwired logic in the FPGA coprocessor. REMARC has limited hardware interconnections. Each nano processor has direct inputs from the four nearest neighbors and it is connected by two 32-bit data buses. The FPGA coprocessor has more flexible hardware interconnections which are configured in a bit-wise fashion.

We assume that REMARC will operate at the same frequency as the main processor. The cycle times of the FPGA coprocessor (*Tfpga*) are varied. We evaluate the FPGA coprocessor performance, assuming Tfpga values that are 1x, 2x, 5x, and 10x the CPU cycle time (*Tcpu*).

3 Performance Evaluation

3.1 Simulation Methodology

We developed the reconfigurable coprocessor simulator using the SimOS simulation environment [15] . SimOS models the CPUs, memory systems, and I/O devices in sufficient detail to boot and run a commercial operating system. As a base CPU simulation model, we used the "*MIPSY*" which models a simple single issue RISC processor similar to the MIPS R3000.

REMARC functions are added to the MIPSY model. The latency of a reconfigurable execution (*rex*) instruction is the sum of the number of the executed global instructions and the pipeline latency (5 cycles). If a following instruction attempts to read the result of a *rex* instruction, the pipeline will stall for the cycles of the *rex* instruction's latency (Data Dependency). Furthermore, a following *rex* instruction will stall if a previous *rex* instruction is still executing (Resource Conflict).

We evaluate the execution time of the FPGA coprocessor by changing the latencies of the *rex* instructions. First, we estimate the number of execution cycles of the *rex* instructions based on the FPGA delay model. In this model, two sequences can be executed within one FPGA cycle:

- One stage of interconnection by long or short wire and one stage of function not using the carry chain
- One stage of interconnection by short wire and any one stage of function

This model is almost the same as the Garp's delay model [8] and the XC4000's medium frequency design which will operate at about 70 MHz [16]. Then, we normalize the number of execution cycles based on the CPU cycle time, assuming the FPGA cycle time is 1x, 2x, 5x, or 10x the CPU cycle time. Finally, we use this estimated cycle count of the *rex* instruction in the simulator.

We also developed simulators which can execute multimedia instructions similar to the Intel MMX instruction set extensions [2].

To make the comparison fair, the same application source codes are used for the evaluation except for the use of the multimedia instructions or the augmented coprocessor instructions. The memory system parameters used commonly through the performance evaluations are found in Table 3. We used the "gcc" compiler with the "-O2" optimization option. This option executes most global compiler optimizations except for loop unrolling and function inlining.

I-cache	32 K bytes, 2-way set.
D-cache	32 K bytes, 2-way set.
L2 cache	256 K bytes, 2-way set.
L1 miss penalty	5 cycles
L2 miss penalty	50 cycles

Table 3: Memory System Parameters

3.2 DES Encryption

The Data Encryption Standard (DES) is one of the most important encryption algorithms and has been a worldwide standard for over 20 years. It is widely used to provide secure communication over the Internet. DES is also a good application for reconfigurable processors because it has a lot of fine-grained parallelism in the form of bit-level irregular data movements which make software implementation on conventional microprocessors difficult and inefficient.

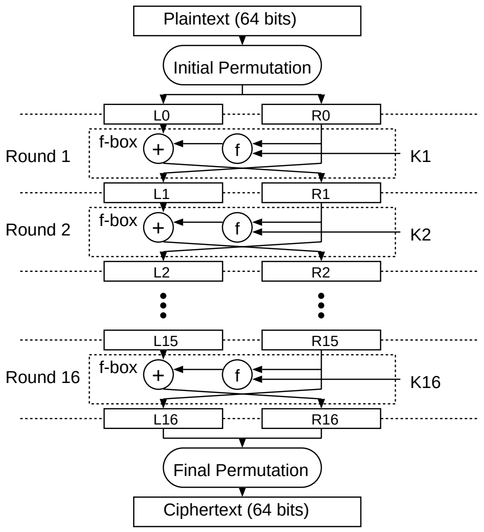


Figure 5: DES Encryption Algorithm

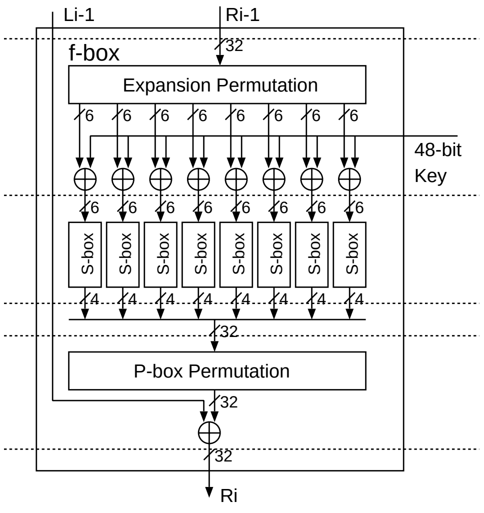


Figure 6: DES f-box Algorithm

Figure 5 shows an outline of the DES algorithm. DES takes as input 64-bit plaintext. After the initial permutation, there are 16 rounds of "f-box" operations, as shown in Figure 6, including expansion permutation, XOR with the key, S-box table lookup, P-box permutation, and XOR with the result of the previous round. The final permutation is performed on the 64-bit result of the sixteenth round.

We used the DES encryption program [17] based on the Electronic Codebook (ECB) mode. Although the ECB mode is less secure than the Cipher Block Chaining (CBC) mode, it is more commonly used and its operation can be pipelined.

3.2.1 DES Implementation on REMARC

We decided to divide the algorithm between the main processor and REMARC. The initial permutation and the final permutation are executed by the main processor and the 16 rounds of f-box operations are executed by REMARC. Each row of nano processors executes two f-box operations. For instance, eight nano processors in the row 0 execute the first and second rounds, the row 1 execute the third and fourth rounds, and so on.

Operation	CPU Cycles
Data Load	2
Exp. Permutation	6 (3 x 2 iter.)
Key XOR	2 (1 x 2 iter.)
S-box	12 (6 x 2 iter.)
P-box Permutation	22 (11 x 2 iter.)
Left XOR	4 (2 x 2 iter.)
Data Transfer	3
Total	51

Table 4: Execution Cycle Breakdown of Two f-box Operations on REMARC

Table 4 is the execution cycle breakdown of the two f-box operations implemented by the eight nano processors in each row. The 16 f-box operations are pipelined into 8 stages by the eight rows of the nano processor array. REMARC can generate a result of the 16 f-box operations every 51 cycles. As Table 4 shows, because of the limited interconnection of REMARC, more than half (28 cycles) of the execution time are used for the expansion permutation and the P-box permutation.

3.2.2 DES Implementation on the FPGA Coprocessor

First, we estimate the latency and the throughput of one f-box operation. The expansion permutation and the XOR operation with the keys can be executed in one cycle because it can be implemented by long wire and simple logic. The S-box table lookup requires two cycles to execute because it consists of LUTs and MUXs. The P-box permutation and the XOR operation can be executed in one cycle. The total latency of the f-box operation is four cycles, and it can be fully pipelined. Therefore, the maximum throughput of the f-box operation is one FPGA cycle.

We assume three distinct cases, implementing one f-box, 16 f-boxes, and all of the DES encryption algorithm including 16 f-boxes, the initial permutation, and the final permutation.

In the case of one f-box implementation, because each f-box operation takes as input the result of the previous f-box operation, the f-box operations cannot be pipelined.

Tfpga	FPGA Coprocessor (1 f-box)	FPGA Coprocessor (16 f-boxes)
Tcpu	64	1
Tcpu x 2	128	2
Tcpu x 5	320	5
Tcpu x 10	640	10

Table 5: Execution Throughput of 16 f-box Operations on Different FPGA Coprocessors (CPU Cycles)

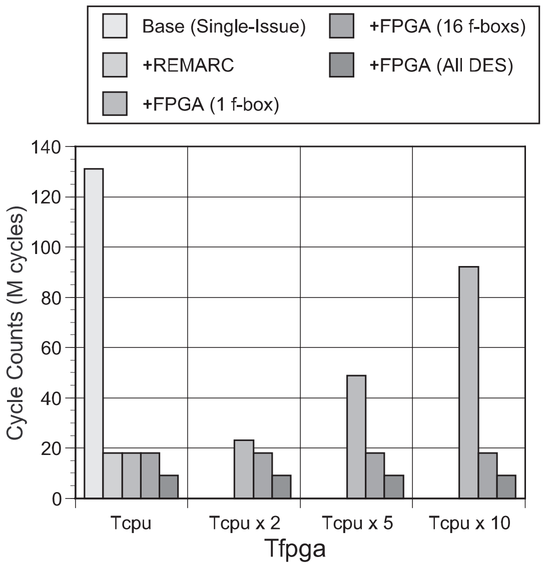


Figure 7: DES Encryption (1M Bytes) Results – Single Issue Architecture

It takes 64 FPGA cycles (4 cycles x 16 rounds) to execute the 16 rounds for the f-box operations. When the 16 f-boxes are implemented in the FPGA array, the operations can be pipelined. It takes only one FPGA cycle to generate a new result for the 16 f-box operations after the pipeline is filled up. Table 5 normalizes the throughput of the 16 f-box operations in CPU cycles for both the one f-box implementation and the 16 f-box implementation.

When the complete DES encryption algorithm is implemented in the FPGA coprocessor, the initial and final permutations can be fully pipelined and a new 64-bit ciphertext character is generated every FPGA cycle.

To estimate the number of CLBs without wiring overhead, we use the following method. As shown in Figure 6, the f-box operation requires a 48-bit XOR, eight 6-bit input 4-bit output LUTs, a 32-bit 4-1 Multiplexer, and a 32-bit XOR. A two-bit XOR or a 4-1 Multiplexer fits to one CLB. A 6-4 LUT needs 16 4-1 LUTs or 8 CLBs. In total, 136 CLBs are required for one f-box operation. The 16 f-box implementation needs 2176 (136 x 16) CLBs. We expect that in an actual implementation more CLBs would be dedicated to the extra wiring which realizes the initial permutation and the final permutation.

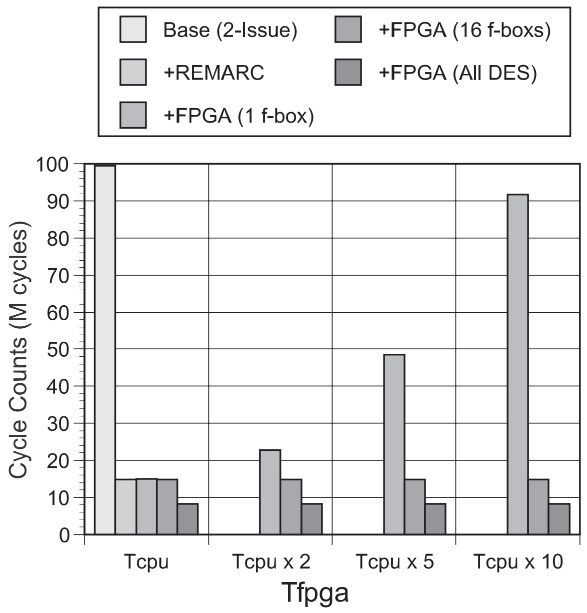


Figure 8: DES Encryption (1M Bytes) Results – 2-way Superscalar Architecture

3.2.3 Performance Evaluation Results

Figure 7 shows the result of the DES encryption of 1M bytes data. The base architecture of the main processor is a single issue processor. To optimize performance we implemented a software pipelining technique which can overlap operations in the main processor and the reconfigurable coprocessors. REMARC achieves a 7.3 times performance improvement. The FPGA coprocessor achieves the same performance improvement if its execution cycles for the 16 f-box operations are less than 64 CPU cycles. This indicates that the main processor which executes the initial permutation and the final permutation limits the performance improvement in these cases.

If the FPGA coprocessor implements all of the DES operations, the execution cycles are reduced to 9.0 M CPU cycles. Reducing the load on the main processor, this implementation delivers a 14.5 times performance improvement over the base processor. Even though the cycle time of the FPGA coprocessor is 10x Tcpu, it takes only 10 CPU cycles to generate one 64-bit encrypted data. This execution time on the FPGA coprocessor is short enough that the main processor still limits the performance.

Using a higher performance main processor is another way to reduce the total execution time. As shown in Figure 8, if a 2-way superscalar processor is used as the main processor, the execution times are reduced from 17.3 M CPU cycles to 14.8 M CPU cycles for the +REMARC, +FPGA(1 f-box) at Tcpu cycle time, and +FPGA(16 f-box) implementations.

The other point that is demonstrated in Figure 7 and Figure 8 is that the one f-box implementation at slow cycle times, especially Tcpu x 5 and Tcpu x 10, delivers only limited performance improvements. Even though the execution time of one f-box operation is only four FPGA cycles, the slow cycle time of the FPGA coprocessor results in small performance gains. Furthermore, the performance is limited by the reconfigurable coprocessor; there is no performance improvement even if the 2-way superscalar main processor is used as shown in Figure 8.

3.3 MPEG-2 Decoding

MPEG-2 is one of the most popular video compression standards. The algorithm for MPEG-2 decoding consists of five stages, Variable Length Decoding (VLD), Inverse Quantization (IQ), Inverse DCT (IDCT), Motion Compensation (MC), and the addition of the IDCT and the MC results (Add_Block). We used the *mpeg2decode*[19] program distributed by MPEG software simulation group as a base MPEG-2 decoding program. The *mpeg2decode* program was modified using the reconfigurable coprocessor instructions. The bitstream "*mobile*" was chosen as a benchmark.

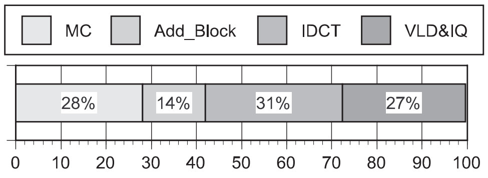


Figure 9: Execution Time Breakdown of MPEG-2 Decoding

To determine the execution time breakdown we profiled the MPEG-2 decoding program using the profiling tool *pixie* [18]. Figure 9 shows the percentage of execution time spent in each stage while decoding 30 frames. The figure indicates that the execution time is distributed among MC, Add_Block, IDCT, and VLD/IQ. This suggests that reconfigurable coprocessors need to support multiple functions to accelerate the entire MPEG-2 decoding. As a first step, however, we investigate the acceleration of IDCT, which represents 31% of the execution time for MPEG-2 decoding.

3.3.1 Acceleration of IDCT

We use the fast IDCT algorithm which is the reverse of the DCT algorithm presented in [20]. The 1D-IDCT algorithm consists of 12 multiplications and 32 additions/subtractions. The 8x8 2-D IDCT is composed using a row IDCT followed by a column IDCT. Each of the row and column IDCTs consists of eight 1-D IDCTs.

For REMARC, the 1D-IDCT algorithm is changed slightly. The algorithm consists of 16 multiplications and 30 additions/subtractions to achieve more regular data movements. Each of the eight nano processors in the same row or column calculates a 1-D IDCT. Therefore, two multiplications are mapped into each nano processor to maintain the load balance even among the nano processors. First, each of the eight nano processors in the same row calculate the 1-D row IDCT. Then, each of the eight nano processors in the same column calculate the 1-D column IDCT. Because the 2-D IDCT is done in place, there is no need to execute a matrix transposition. In this fashion an 8x8 nano processor array naturally implements the 2-D IDCT. The 2-D IDCT can be executed in 54 cycles by REMARC. REMARC can realize a fast 2-D IDCT implementation which exploits the large amount of parallelism from the algorithm.

We estimate the latency and the throughput of the 1-D IDCT for the FPGA coprocessor. The IDCT algorithm has five stages of additions/subtractions and one stage of multiplication. The bit width of these operations is 16 bits. We assumed each stage of the additions/subtractions requires two FPGA cycles. The multiplication requires four FPGA cycles to execute using a constant (k) coefficient multiplier (KCM). Therefore, the latency of the 1-D IDCT is 14 FPGA cycles (2 FPGA cycles x 5 + 4 FPGA cycles). The throughput is restricted to four FPGA cycles by the KCM.

We implement the IDCT in the FPGA array using two models. One model only implements one 1D-IDCT and the matrix transposition circuits. This 1D-IDCT circuit is used 16 times to realize a 2-D IDCT. The other model implements eight 1D-IDCTs and the matrix transposition circuits.

In the first IDCT implementation, eight 1D-IDCTs can be pipelined. It takes 42 FPGA cycles (14 FPGA cycles + 4 FPGA cycles x 7) to execute eight 1D-IDCTs, or a row/column IDCT.

The matrix transposition is required between the row IDCT and column IDCT, and it breaks the IDCT execution pipeline, The total execution cycle counts for one 2D-IDCT are 84 FPGA cycles for the one 1D-IDCT implementation and 28 FPGA cycles for the eight 1D-IDCT implementation. Table 6 normalizes the execution cycles in CPU cycles.

We estimate the number of CLBs to implement 1D-IDCT and the matrix transposition. At least 808 CLBs are necessary for 1D-IDCT and 512 CLBs for the matrix transposition. The implementation of one 1D-IDCT and the matrix transposition requires 1320 CLBs, and the implementation of eight 1D-IDCTs and the matrix transposition requires 6976 CLBs. Again these estimates include no wiring overhead.

Tfpga	FPGA Coprocessor (1x 1D-IDCT HW.)	FPGA Coprocessor (8x 1D-IDCT HW.)
Tcpu	84	28
Tcpu x 2	168	56
Tcpu x 5	420	140
Tcpu x 10	840	280

Table 6: 2D-IDCT Execution Cycles (CPU Cycles)

Figure 10 shows the execution time of the MPEG-2 decoding for 15 frames of the "mobile" bitstream. Execution times of the processors with the reconfigurable coprocessor are compared with the single issue base processor and the processor with the multimedia extensions (+MMX).

Although the multimedia extensions can reduce the cycle counts of the IDCT part by a factor of 2.7 from the base processor, REMARC and the FPGA coprocessor which operate at the same frequency as the main processor achieve

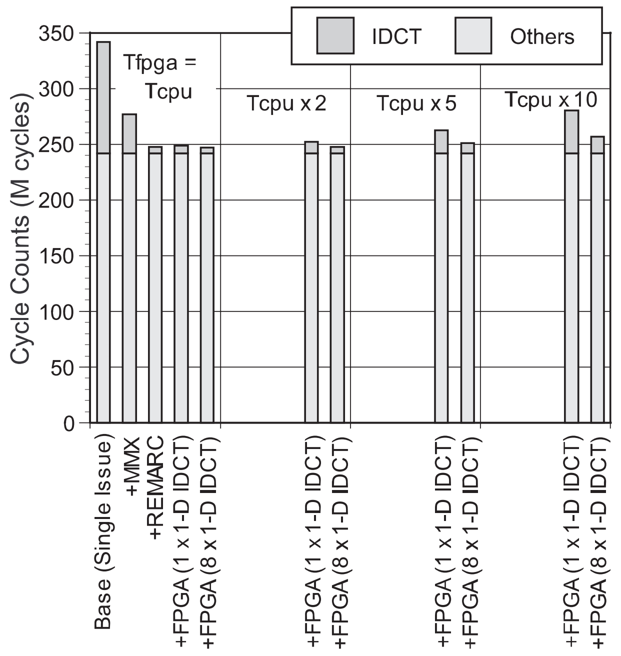


Figure 10: MPEG-2 Decoding (15 Frames) Results with Accelerating IDCT

from 14.4x to 20.4x performance improvements. They are from 5.3x to 7.6x faster than the multimedia extensions.

Another observation from Figure 10 is that the small and slow FPGA coprocessor does not achieve good performance improvement, which is similar to the results we saw with DES encryption. The FPGA coprocessor implementing a single 1-D IDCT with a cycle time of Tcpu x 10 is no longer faster than the multimedia extensions.

3.3.2 Acceleration of MC and Add_Block

The execution time of IDCT can be dramatically reduced by the reconfigurable coprocessor. However, the execution time of IDCT is only 31% of the entire MPEG-2 decoding. The performance improvement of the entire program is only 22.8% for the multimedia instructions and 38.0% for REMARC. This suggests that a reconfigurable coprocessor needs to support multiple functions to accelerate the entire MPEG-2 decoding.

In addition to IDCT, we can accelerate MC and Add_Block using the multimedia instructions and REMARC. This covers more than 70% of the original execution time. The REMARC instructions for MC and Add_Block are SIMD type instructions and very similar to multimedia instructions. They can operate on 8 or 16 data elements at the same time using 8 or 16 nano processors.

The performance evaluation result is shown in Figure 11. Using REMARC as a coprocessor the performance improves by a factor of 2.3 from the base processor and 1.33 from the architecture with the multimedia extensions.

We were not able to evaluate the FPGA coprocessor implementation of MC and Add_Block in detail. However, the FPGA coprocessor can also implement the same SIMD type instructions as those used in REMARC for these algorithms. The performance of the FPGA coprocessor will be almost the same as REMARC, if its cycle time is Tcpu or Tcpu x 2. If the cycle time of the FPGA coprocessor is much slower than that of the main processor, its performance will decrease greatly because the SIMD type instructions only exploit limited parallelism.

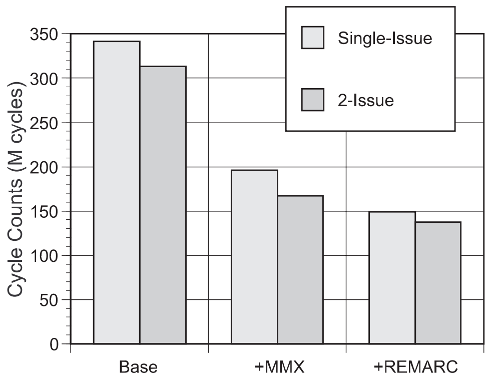


Figure 11: MPEG-2 Decoding (15 Frames) Results with Accelerating IDCT, MC, and Add_Block

To implement the specialized instructions for IDCT, MC, and Add_Block in REMARC, 36% of the nano instruction RAMs and 19% of the instruction RAM of the global control unit are required.

In the case of the FPGA coprocessor, we have to consider implementations of multiple functions in hardware. One implementation approach is to divide the hardware resources among each function. This implementation reduces the number of CLBs and limits the parallelism exploited in each function. This would be a more serious problem for the FPGA coprocessor than for multi-chip FPGA systems, because the on-chip FPGA coprocessor supports a smaller number of CLBs than multi-chip FPGA systems.

Another implementation approach would be to use run-time reconfiguration to support multiple functions. Reconfiguration should be done after each operation, such as IDCT, MC, and Add_Block, for all data in one frame. Because the data size of one frame is more than 1M bytes and much larger than the size of the configuration data, run-time reconfiguration is feasible for this application.

4 Hardware Estimation

We estimate the hardware size of the nano processor from the layout of the Stanford "*Torch*" project [21]. Figure 12 shows the floor plan of the nano processor.

Two nano processors could fit in 3480λ x 5000λ. Therefore, the size of one nano processor is 8.7 M λ^2. In a 0.25μm process the size of 64 nano processors would be 1.74mm x 5.0mm or 8.7mm^2. The size of REMARC including the 4K-byte global instruction RAM and the coprocessor data registers would be roughly 1.74mm x 6mm or 10.44mm^2. We assumed the size of the CLB is about 1.25 M λ^2[22]. Figure 13 shows four examples of microprocessors with the reconfigurable coprocessors using a 0.25μm process.

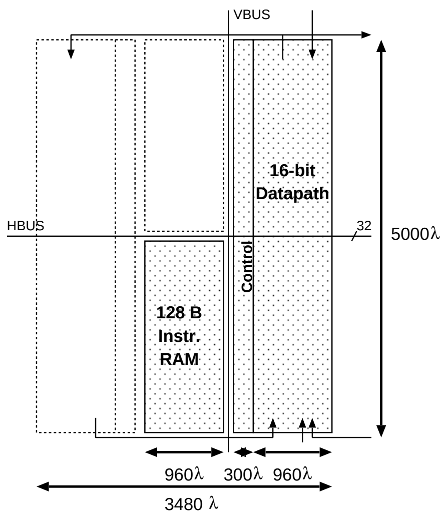


Figure 12: Nano Processor Floor Plan

Figure 13 (a) and (b) integrate REMARC or the FPGA coprocessor with a single issue processor. The area of the single issue processor is based on the MIPS R5000. The R5000 can issue a limited combination of two instructions in the same cycle. The latest embedded microprocessors use a full 2-way in-order issue superscalar architecture [23]. This indicates that reconfigurable coprocessors can be integrated with embedded microprocessors with a small amount of extra hardware. As shown in Figure 13 (a), about 17% of the die is dedicated to REMARC. The die size of this chip is still small enough to provide low-cost processors for embedded applications.

The same area as used by REMARC can hold about 540 CLBs which is too small to implement both a 1D-IDCT and the matrix transposition at the same time or 16 f-box operations of the DES encryption. As the performance evaluation results (Figure 7, 8, and 10) show, the small FPGA array will only provide limited performance improvements if the FPGA coprocessor operates at a much lower frequency than the main processor. On the other hand, REMARC can improve the performance dramatically using the same area.

Figure 13 (c) integrates REMARC with the 4-way out-of-order superscalar processor. The die size of MIPS R10000 [24] is used for this estimation. Only 6.5% of the die is used for REMARC. Figure 13 (d) uses the same die size for a single issue processor and a relatively large FPGA coprocessor. In this case 63% of the die is dedicated to the FPGA coprocessor and includes more than 5000 CLBs.

The comparison of these two chips is interesting because the large FPGA coprocessor, for instance, can implement the entire DES encryption algorithm including the initial permutation and the final permutation. These permutations are inefficient for a general purpose processor and REMARC to implement. However, the wide issue superscalar processor also can improve the performance compared to the single issue processor. We do not have a simulator which models the wide issue superscalar processor together with REMARC. We assume that the f-box operations executed on REMARC are overlapped completely

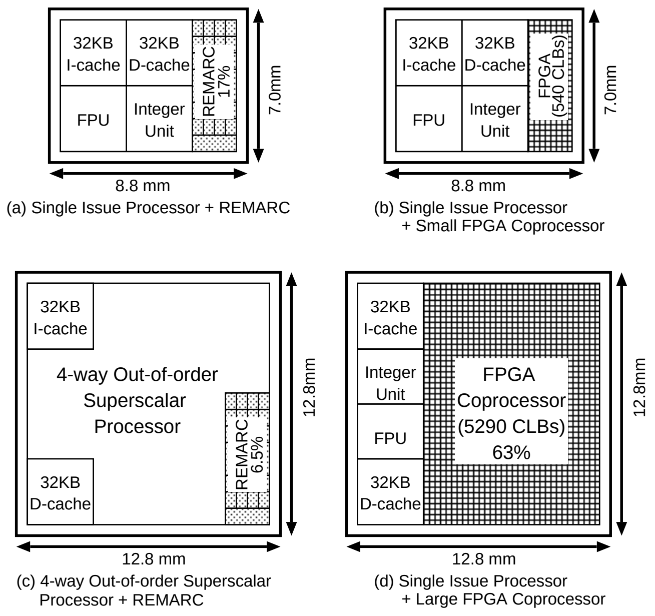

32KB I-cache
32KB D-cache
REMARC 17%
FPU
Integer Unit
7.0mm
8.8 mm
(a) Single Issue Processor + REMARC
32KB I-cache
32KB D-cache
FPGA (540 CLBs)
FPU
Integer Unit
7.0mm
8.8 mm
(b) Single Issue Processor + Small FPGA Coprocessor
32KB I-cache
4-way Out-of-order Superscalar Processor
REMARC 6.5%
32KB D-cache
12.8mm
12.8 mm
(c) 4-way Out-of-order Superscalar Processor + REMARC
32KB I-cache
Integer Unit
FPU
32KB D-cache
FPGA Coprocessor (5290 CLBs) 63%
12.8mm
12.8 mm
(d) Single Issue Processor + Large FPGA Coprocessor

Figure 13: Chip Size Estimation

with the execution in the superscalar main processor and the execution in REMARC is fast enough to be eliminated from the evaluation. We then estimate the performance by executing the DES encryption program without the f-box operations using the 4-way out-of-order superscalar processor simulation model. The execution cycles are 8.7 M cycles and almost the same as that of the FPGA coprocessor implementing the whole DES encryption algorithm. Both architectures in Figure 13 (c) and (d) would deliver comparable performance.

In the case of the MPEG-2 decoding, even the large FPGA coprocessor does not have enough CLBs to implement eight 1D-IDCTs and the matrix transposition which require about 7000 CLBs. REMARC can execute a 2D-IDCT faster than the large FPGA coprocessor with one tenth of the hardware area. Furthermore, the 4-way superscalar main processor integrated with REMARC can also accelerate the part of MPEG-2 decoding executed by the main processor, which is about 27% of the execution time. On the other hand, the large FPGA coprocessor could implement the VLD hardware which cannot be accelerated by the multimedia instructions or REMARC. At this time, we are not sure whether the large FPGA coprocessor would have better performance for MPEG-2 decoding than the 4-way superscalar processor with the multimedia instructions or REMARC. However, programmability and run-time reconfiguration overhead would be large disadvantage of the FPGA coprocessor. A more detailed comparison of these architectures is an interesting topic for future work.

5 Conclusion

In this paper, we compared two reconfigurable coprocessor architectures, REMARC and the FPGA coprocessor. Through the performance evaluations of two realistic multimedia applications, we verified that both of the architectures could exploit more fine grain parallelism from the applications than multimedia extended ISAs such as MMX and both could improve the performance dramatically.

As the simulation results show, REMARC achieves speedups of a factor of 7.3 for DES encryption and 2.3 for MPEG-2 decoding. Because the area of REMARC is relatively small, it is a good candidate for a reconfigurable coprocessor to augment the performance of general purpose microprocessors.

The small FPGA coprocessor can exploit little parallelism and limits the performance improvement, especially if its operating frequency is low. The large FPGA coprocessor would be a much more interesting architecture because it has the potential to accelerate some algorithms which are inefficient on other architectures. For instance, the FPGA coprocessor is more suitable for implementing the VLD portion of MPEG-2 decoding. However, difficulty of programming and run-time reconfiguration overhead would still be a disadvantage of the FPGA coprocessor.

Acknowledgment

We wish to thank Basem A. Nayfeh for his help with SimOS and Marco Platzner, Rachid Helaihel, Victor Lam, and Manohar Prabhu for their insightful comments. We also appreciate the help of Eiji Iwata for developing the MPEG-2 decoding implementation.

References

[1] Marc Tremblay, J. Michael O'Connor, Venkatesh Narayanan, and Liang He, *"VIS Speeds New Media Processing"*, IEEE Micro, pp. 10–20, Aug. 1996.

[2] Alex Peleg and Uri Weiser, *"MMX Technology Extension to the Intel Architecture"*, IEEE Micro, pp. 42–50, Aug. 1996.

[3] Ruby B. Lee, *"Subword Parallelism with MAX-2"*, IEEE Micro, pp. 51–59, Aug. 1996.

[4] Kunle A. Olukotun, Rachid Helaihel, Jeremy Levitt, and Ricardo Ramirez, *"A Software-Hardware Cosynthesis Approach to Digital System Simulation"*, IEEE Micro, Vol. 14, pp. 48–58, Nov. 1994.

[5] Peter M. Athanas and Harvery F. Silverman, *"Processor Reconfiguration through Instruction-set Metamorphosis"*, IEEE Computer, vol. 26, no. 3, pp. 11–18, Mar. 1994.

[6] Rahul Razdan, Karl Brace, and Michael D. Smith, *"PRISC Software Acceleration Techniques"*, IEEE International Conference on Computer Design, 1994.

[7] Ralph D. Witting and Paul Chow, *"OneChip: An FPGA Processor with Reconfigurable Logic"*, IEEE Symposium on FPGAs for Custom Computing Machines, 1996.

[8] John. R. Hauser and John Wawrzynek, *"Garp: A MIPS Processor with a Reconfigurable Coprocessor"*, IEEE Symposium on FPGAs for Custom Computing Machines, 1997.

[9] Scott Hauck, Thomas W. Fry, Matthew M. Hosler, and Jeffrey P. Kao, *"The Chimaera Reconfigurable Functional Unit"*, IEEE Symposium on FPGAs for Custom Computing Machines, 1997.

[10] Ulrich Schmidt and Sönke Mehrgardt, *"Wavefront Array Processor for Video Applications"*, International Conference on Computer Design, 1990.

[11] Alfred K. Yeung and Jan M. Rabaey, *"A 2.4GOPS Data-Driven Reconfigurable Multiprocessor IC for DSP"*, IEEE International Solid-State Circuits Conference Digest of Technical Papers, pp.108–109, 1995.

[12] Ethan Mirsky and André DeHon, *"MATRIX: A Reconfigurable Computing Architecture with Configurable Instruction Distribution and Deployable Resources"*, IEEE Symposium on FPGAs for Custom Computing Machines, 1996.

[13] Takashi Miyamori and Kunle Olukotun, *"REMARC: Reconfigurable Multimedia Array Coprocessor"*, FPGA'98 (as poster paper), 1998.

[14] Gerry Kane, *"MIPS RISC Architecture"*, Prentice Hall, Englewood Cliffs, 1988.

[15] M. Rosenblum, S. Herrod, E. Withchel, and A. Gupta, *"The SimOS Approach"*, IEEE Parallel and Distributed Technology, Vol. 4, No. 3, 1995.

[16] Xilinx, *"XC4000XL-1 FPGAs Exceed 100MHz"*, XCELL Articles, Q3, 1997.

[17] Bruce Schneier, *"Applied Cryptography: Protocols, Algorithms, and Source Code in C"*, 2nd edition, John Wiley and Sons, 1996.

[18] Michael D. Smith, *"Tracing with pixie"*, Technical Report No. CSL-TR-91-497, Computer Systems Laboratory, Stanford University, 1991.

[19] MPEG Software Simulation Group, *"mpeg2encode/mpeg2decode version 1.2"*, http://www.mpeg.org/MSSG/, July, 1996.

[20] R. Wood, A. Cassidy, and J. Gray, *"VLSI Architectures for Field Programmable Gate Arrays: A Case Study"*, IEEE Symposium on FPGAs for Custom Computing Machines, 1996.

[21] Michael D. Smith, *"Support for Speculative Execution in High-Performance processors"*, Ph.D Thesis, Stanford University, Nov. 1992.

[22] André DeHon, *"Reconfigurable Architectures for General-Purpose Computing"*, Technical Report 1586, MIT Artificial Intelligence Laboratory, Chapter 4, Oct. 1996.

[23] F. Arakawa, O. Nishii et al., *"SH4 RISC Microprocessor for Multimedia"*, HOT Chips IX Symposium Record, Aug. 1997.

[24] Kenneth C. Yeager, *"The MIPS R10000 Superscalar Microprocessor"*, IEEE Micro, pp. 28–40, April 1996.