The Reconfigurable Instruction Cell Array

Sami Khawam, Ioannis Nousias, Mark Milward, Ying Yi, Mark Muir, and Tughrul Arslan

***Abstract*—This paper presents a novel instruction cell-based reconfigurable computing architecture for low-power applications, thereafter referred to as the reconfigurable instruction cell array (RICA). For the development of the RICA, a top-down software driven approach was taken and revealed as one of the key design decisions for a flexible, easy to program, low-power architecture. These features make RICA an architecture that inherently solves the main design requirements of modern low-power devices. Results show that it delivers considerably less power consumption when compared to leading VLIW and low-power digital signal processors, but still maintaining their throughput performance.**

***Index Terms*—Dynamic reconfiguration, parallel processing, reconfigurable computing.**

I. Introduction

THE NEED FOR new hardware architectures for future semiconductor devices was recognized several years ago as current architectures will not be able to cope with rising future requirements in data processing and throughput. Typical examples are mobile devices for next generation networks where a high amount of audio and video data will need to be processed. This adds extra burden on the processing elements in order to maintain a high throughput and stay efficient in terms of power-consumption and silicon costs. The hardware also needs to prove to be adaptable to upcoming standards. The four, equally important, requirements of such computing architectures are cost-reduction, short design time, low-power consumption, and high performance. Research into suitable architectures is an ongoing process, as a comprehensive solution capable of meeting all these requirements has yet to be found.

To date there are several established ways in which algorithms can be implemented and run on silicon. Fig. 1 summarizes the merits and drawbacks of selecting a particular computation architecture in relation to each other. It is worthy to note that not a single architecture meets all the desired criteria, thus leaving room to consider other tradeoffs.

Application-specific integrated circuits (ASICs) are well known to provide low power and high throughput compared to other architectures, however, their high design costs and limited post-fabrication flexibility means that they are only really feasible for large production count or for ultra low-power applications. CPUs are popular due to their ability to implement designs after fabrication and their familarity to designers.

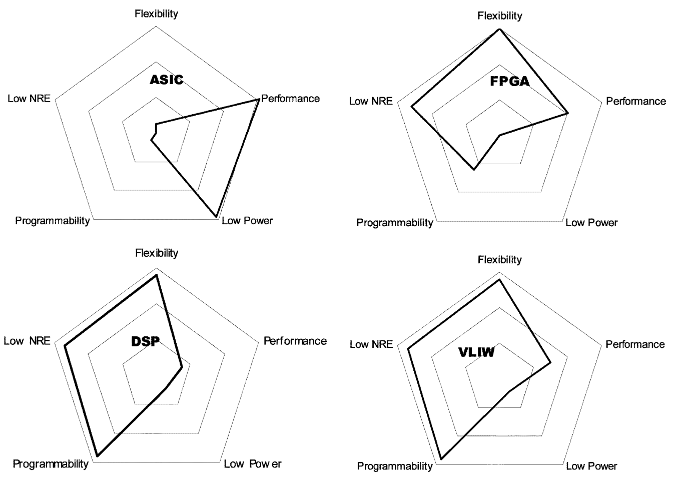


Fig. 1. Computation architecture tradeoffs.

Over the years a number of techniques, such as several layers of cache, memory management, and branch prediction have been added to the CPU in order to keep the arithmetic logic unit (ALU) supplied with adequate instructions and data. This means that a diminishing proportion of the available silicon is actually used for the active computation. VLIW DSP architectures offer advantages in terms of extra parallel processing. However, they are easily restricted by the limited amount of instruction level parallelism (ILP) found in typical programs. Field-programmable gate arrays (FPGAs) represent one possible implementation of a range of reconfigurable computing devices. Their success lies in their ability to map algorithms onto their logic and interconnects after fabrication. This allows fabrication costs to be shared across many designs. While FPGAs provide the ability to map any logic functions on their fine-grained lattice, this has an impact on energy consumption since the majority of the available transistors are used to provide flexibility. Moreover designers are often required to have specialized skills in order to convert algorithms into a suitable register transfer level (RTL) code for synthesis. To achieve a suitable compromise of all design characteristics, designers develop custom system-on-chip (SoC) with a mixture of these architectures. To solve the problem of finding the ideal architecture, the research community has proposed many solutions based on the promise of high-performance offered by reconfigurable computing in terms of flexibility and a high amount of parallel processing. This includes architectures like Matrix, Garp, Elixent, PACT XAPP, SiliconHive, Montium, Pleiades, and Morphosys [2]–[9]. Even though these solutions achieve remarkable results in high computational-power and flexibility, they either do not provide enough power savings or are too difficult to program.

Manuscript received May 10, 2006; revised May 30, 2007.

S. Khawam, M. J. Milward, and M. Muir are with Spiral Gateway, Edinburgh EH9 3JL U.K.

I. Nousias and T. Arslan are with the School of Engineering and Electronics, University of Edinburgh, Edinburgh EH9 3JL U.K., and also with Spiral Gateway, Edinburgh EH9 3JL U.K.

Y. Yi is with the School of Engineering and Electronics, University of Edinburgh, Edinburgh EH9 3JL U.K.

Digital Object Identifier 10.1109/TVLSI.2007.912133

In this paper, we present and evaluate our novel reconfigurable instruction cell architecture (RICA) that tries to specifically address all the desired requirements for future portable devices. By designing the silicon fabric in a similar way to reconfigurable arrays but with a closer equivalence to software, we achieve the same high performance as coarse-grain FPGA architectures and maintain the same flexibility, low cost, and programmability as digital signal processors (DSPs). The main distinguishing and novel features of our reconfigurable architecture are as follows.

- An array of customizable instruction cells (ICs) that can be dynamically reconfigured on every clock cycle enabling the mapping of data-paths of both dependent and independent instructions.
- The reconfigurable core has ICs to deal with execution flow control (conditional branches). Therefore, it does not require coupling to a general purpose processor, since it provides these features.
- The cell array configuration can be tailored towards different application domains, thus providing additional power optimizations and performance.
- A reconfiguration rate controller is provided to minimize the impact of varying critical paths, which is an intrinsic characteristic of supporting dependent instructions
- The architecture is programmable with a high level language.

Consequently, a new dynamically reconfigurable system is presented in this paper with its associated programming environment.

Section II of this paper covers related and previous work on reconfigurable computing architecture. Section III describes the RICA design along with descriptions of various key architectural design features. Section IV discusses some of the techniques used in the RICA to reduce power dissipation. Section V covers the tool-flow for programming RICA. Section VI covers the evaluation of a sample RICA design. Finally, the conclusion is drawn in Section VII.

II. Related and Previous Work

Traditionally, algorithm functions are either statically realized in hardware (ASIC) or temporarily run on general-purpose processors (GPP). These two cases form the boundaries of a large space of design exploration for possible ways of computing on silicon. The center ground of this space is filled by reconfigurable computing devices, with FPGAs being a well-known commercially successful example.

The concept of reconfigurable computing has been around since the 1960s, when Estrin's landmark paper proposed the concept of a computer consisting of a standard processor and an array of "reconfigurable" hardware [1]. The main processor would control the behavior of the reconfigurable hardware. The reconfigurable hardware would then be physically tailored to a specific task, such as image processing or pattern matching, with the aim of running as quickly as a dedicated piece of hardware. Once the task was complete, the hardware could be manually adjusted to do some other tasks. This resulted in a hybrid computer structure aimed at combining the flexibility of software with the speed of hardware. Unfortunately, this idea was way ahead of its time in terms of electronic technology.

In the last decade there has been a recent renaissance in this area of research with many proposed reconfigurable architectures developed both in industry and academia, such as Matrix, Garp, Elixent, PACT XAPP, SiliconHive, Montium, Pleiades, and Morphosys [2]–[9] to name but a few. These designs have only really become feasible by the relentless progress of silicon technology, allowing for complex designs to be implemented onto a single chip. Nevertheless, the development of reconfigurable hardware architecture is only one side of the problem, often overlooked is the importance of appropriate software tools that are essential for the programmability of the system and gaining the maximum performance. It is vital for reconfigurable systems to be programmed from a common high-level language, as this enables complex algorithms to be quickly and effectively taken to the underlying architecture. To date, there is no clear classification taxonomy for these developed reconfigurable systems. However, they can be grouped together depending on various architectural characteristics, such as the level of coupling with a host processor, the logic granularity selected, the type of interconnect and topology and the rate of configuration and its management. Design decisions taken on one architectural aspect greatly influences design decisions taken on others. For example, choosing a fine-grained logic granularity means that a larger amount of silicon is allocated to the interconnect for connecting the logic together. This in turn impacts the rate it can reconfigure the device in real time due to the larger bitstreams of instructions needed.

Several proposed reconfigurable systems combine a RISC processor with a coarse-grain reconfigurable array; some such systems are Morphosys, Garp, Elixent, and Pact XAPP. The Garp reconfigurable architecture developed at the University of California in 1997 involves a host processor that manages the main thread of control while certain loops or subroutines use a reconfigurable array. The array is composed of rows of blocks, which have a resemblance to configurable logic blocks (CLBs) of a Xilinx FPGA. The blocks operate on 2-bit data. Vertical and horizontal block-to-block wires provide data movement within the array, the interconnect only connects to its neighboring logic blocks. Separate memory buses are provided to move information in and out of the array. Each computation unit in the Garp array can perform a logical or arithmetic operation on the input 2-bit operands. Use of the array is controlled by the main processor. Host and reconfigurable array share the same memory hierarchy. Four memory buses are used to transfer data to and from the array. Other reconfigurable devices which use a reconfigurable array and processor coupling is Elixent, a start-up company based in Bristol, U.K., that was spun out from HP research labs. The reconfigurable array is based on the D-fabrix consisting of a homogeneous grid of 4-bit ALU units. To interface the core, an advanced microcontroller bus architecture (AMBA) bus is used for programming the array and for transferring data to and from the host RISC. Each array has two high speed input/output (I/O) ports of 32 bits; these are divisible into multiples of 4 bits. The Morphosys reconfigurable computing project marries an on-board RISC processor with an array of reconfigurable cells. The Morphosys architecture includes a reconfigurable processing unit, a GPP, and a high bandwidth memory interface. Each reconfigurable cell incorporates a 28-bit ALU, 16×12-bit multiplier, a shift unit, a 16-bit input and 8-bit multiplexer, and 16-bit register file. In Pact's XAPP array, the data-flow sections of algorithms consisting mostly of the pure arithmetic processing are mapped to the reconfigurable array. A microcontroller is utilized to load these code sections dynamically to the XAPP-array, which then processes these tasks. Data exchange between the XAPP, acting as coprocessor, and the microcontroller is done via direct memory access (DMA) or shared memories. The ADRES architecture [11] closely couples the processor and reconfigurable fabric by sharing the memory and register-file; it tries to simplify the programming model through the use of a high-level C language. Nevertheless, the effect of this approach on area and power consumption was not measured.

The main decision why these systems have a separate processor and reconfigurable array is based on the observation that some code is "control flow" intensive, where there are a lot of conditional branches to change program flow, while others parts of code contain highly parallel data paths, suitable for the reconfigurable arrays. However, these systems suffer from difficulties in programming the arrays due to the use of specialized low-level languages. They also suffer from limitations due to the loose coupling between the processors and the arrays, such as how to allocate the program and large amounts of data-transfers between the array and processor.

Standalone coarse-grain reconfigurable arrays, such as those proposed earlier by Khawam *et al.* [10], provide good performance in area and power when compared to FPGAs, however, like FPGAs they require special hardware skills HDL to program them. RaPiD [12] is another one of these arrays, where it utilizes a 1-D structure to take advantage of the fact that all of its functional components are word-width computational devices. One of the advantages of a 1-D structure is a reduction in complexity, especially in the communications network. Another advantage is their ability to map systolic arrays very efficiently, leveraging all of the research into the compilation of algorithms onto systolic arrays. Finally, while a 2-D RaPiD is possible, most 2-D algorithms can be mapped onto a 1-D array through the use of memory elements. To create different versions of RaPiD that target different application domains, the following changes need to be made to the array: modify existing or add new functional units, change the width of the buses or the number of buses present, and modify the routing fabric. The RaPiD architecture can be programmed using the high-level RaPiD-C language but this language is not compatible with ANSI-C and requires manual scheduling of parallel operations.

A fairly recent development in the reconfigurable arena is Stretch's [13] offering of a configurable processor with an internal reconfigurable fabric. Stretch's solution is capable of directly compiling an application from C code. The software developer identifies sections of code suitable for the reconfigurable fabric using their profiling tool. This part of the C/C++ source code can be compiled into configuration data for the reconfigurable fabric while the rest is run on the processor. This approach tries to solve the problem of communication overhead.

On the other side of the reconfigurable spectrum is the multiprocessors approach, where a sea of small processors are connected through an interconnect network. Examples of this are picoChip [14], Ambric [15], SiliconHive [6], etc., and, to a lesser extent, the large CELL microprocessor with its eight fully-functional Synergistic processing elements (co-processors). PicoChip comprises of an array of heterogeneous independent processors with a programmable communication interconnect between the processors. Each processor is a 16-bit-wide RISC, with the inter-processor communication protocol based on a time-division multiplexing (TDM) scheme; data transfers between processor ports occur during time slots and are scheduled at compile time, and controlled using the bus switches. The bus switch programming and the scheduling of data transfers is all fixed at compile time. Ambric concentrates on the interconnect structure and associated protocol. Ambric objects communicate through a simple parallel structure of hardware channels. Each channel is word-wide, unidirectional, point-to-point from one object to another, and acts like a first-input–first-output (FIFO) buffer. Channels carry both data and control tokens, in simple or structured messages. Channel hardware synchronizes its objects at each end dynamically as needed at run time, not scheduled at compile time. Sending or receiving a word on a channel is as simple as reading or writing a processor register. Since Ambric channels synchronize transparently and locally, the application achieves high performance with none of that complex global synchronization. It is entirely feasible for the RICA to be the processors in the Ambric design.

III. RICA ARCHITECTURE

This section starts with a demonstrative example to allow an easier understanding of the basic concept behind the architecture and then explains the structural elements of RICA.

A. *Parallel Processing Example*

The sample C code shown in Table I requires 19 cycles to execute on a typical sequential processor. However, if the same code is compiled for a VLIW DSP, such as the TI C6203, then it would execute in 15 cycles, since the VLIW architecture would try to concurrently execute up to eight independent instructions (six ALUs and two multipliers are available) [16]. If four simultaneous multiplications and four memory accesses were permitted, then the number of cycles would reduce to eight. This is still a long time considering the simplicity of the code and compared to what is achievable in hardwired solutions like FPGAs. The presence of dependent instructions, i.e., operations which are dependent on previous operations, prevents the compiler from achieving further reductions in the number of clock cycles. We can observe that if the hardware architecture supports the mapping of both dependent and independent datapaths, then we could execute a big block of instructions in a single clock cycle without limitation.

We could easily execute the previous C code in only two cycles if the architecture provided 14 operational elements that can execute 4× ADD, 4× RAM, 4× MUL, and 2× REG simultaneously, as shown in Fig. 2. This overcomes the limitation faced by VLIW processors and enables a higher degree of parallel processing. As shown in *Cycle 1* in Fig. 2, the longest

TABLE I
EXAMPLE C-CODE AND ITS ASSEMBLED SEQUENTIAL AND VLIW CODE COMPILED WITH LEVEL-2 OPTIMIZATIONS

C Code	Sequential ASM
`b0 = in_mem[add+0];`	`LD  [r3+0] →r11`
`b1 = in_mem[add+1];`	`LD  [r3+8] →r9`
`b2 = in_mem[add+2];`	`MUL r11, r5→r11`
`b3 = in_mem[add+3];`	`LD  [r3+12]→r13`
`e = b0 * f0 - b2 * f2;`	`LD  [r3+4] →r3`
`f = b1 * f1 - b3 * f3;`	`MUL r3, r6 →r6`
`out_mem[add+0]= e + f;`	`MUL r9, r7 →r5`
`out_mem[add+1]= e - f;`	`MUL r13, r8→r3`
`out_mem[add+2]= f + 2*e;`	`SUB r11, r5→r5`
`out_mem[add+3]= f - e;`	`ADD r5, r5 →r7`
TMS320C6x VLIW ASM	`SUB r6, r3 →r3`
	`SUB r5, r3 →r8`
`LDH  *+A4(2)→A7`	`ADD r7, r3 →r7`
`LDH  *+A4(6)→A3`	`ADD r5, r3 →r6`
`LDH  *+A4(4)→A0`	`LD  r8 →[r4+12]`
`LDH  *A4→A5`	`SUB r3, r5 →r3`
`MPY  A7,B6→B5`	`LD  r6 →[r4+0]`
`MPY  A3,B8→B6 || MPY  A0,A8→A0`	`LD  r3 →[r4+4]`
`MPY  A5,A6→A3`	`LD  r7 →[r4+8]`
`SUB  B5,B6→B5`	
`SUB  A3,A0→A0  || EXT B5,16,16→B5`	
`RET  B3        || EXT A0,16,16→A0`	
`MV   B5→A3     || SUB B5,A0→B6`	
`ADDAH A3,A0→A4|| STH B6→*+B4(6)`	
`ADD  B5,A0→B5  || STH A4→*+B4(4)`	
`STH  B5→*B4    || SUB A0,A3→A0`	
`STH  A0→*+B4(2)`	
15 Cycles (8 cycles if 4 MPY and 4 LD/ST are allowed)	**19 Cycles**

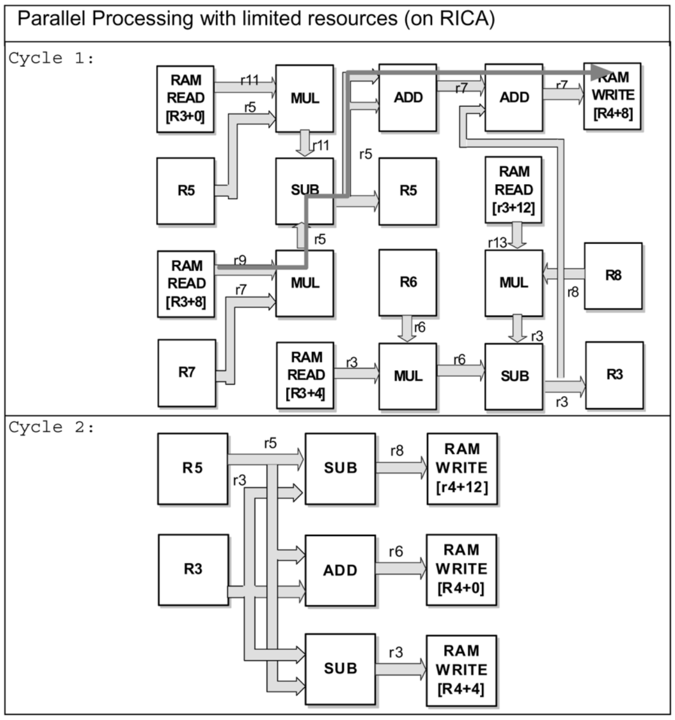
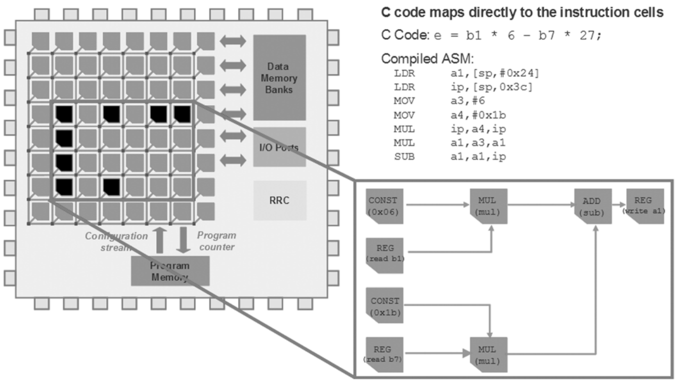


Fig. 2. Execution of the 19 instructions in two cycles if the following parallel resources are available: 4× MUL, 4× ADD, 4× RAM, and 6× REG. The RICA provides a heterogeneous array of instruction cells to map such steps.

delay-path is equivalent to two RAM accesses, one multiplication, and some simple arithmetic operations. This is not much longer than critical paths in typical DSPs when compared to the greater amount of instructions executed in parallel. Therefore, an architecture that supports such an arrangement might be able to achieve similar throughputs as VLIWs but at a lower clock frequency, depending on the type of computation.



C Code: `e = b1 * 6 - b7 * 27;`

Compiled ASM:

```
  LDR     a1,[sp,#0x24]
  LDR     ip,[sp,0x3c]
  MOV     a3,#6
  MOV     a4,#0x1b
  MUL     ip,a4,ip
  MUL     a1,a3,a1
  SUB     a1,a1,ip
```

Fig. 3. Harvard-like structure of the RICA architecture. The reconfigurable core is composed of ICs and a network of programmable interconnects.

B. RICA Reconfigurable Core

With the lessons learned from other reconfigurable fabrics, the RICA architecture has been purposely developed to handle both control and dataflow aspects of the algorithms, along with being able to be programmed from industrial standard programming languages such as ANSI-C, on a high performance dynamically reconfigurable fabric. The concept behind the RICA architecture is to provide a dynamically reconfigurable fabric that allows building circuits such as the ones described in Fig. 2. By providing hardware modules that can execute assembly-like instructions similar to those in Fig. 2, a straightforward and CPU-like design-flow can be easily obtained. In the RICA architecture, these hardware modules are the named ICs. Having an array of interconnectable ICs allows building circuits from an *assembly* representation of programs. The software is then executed in multiple steps, where each step is a datapath of both dependent and independent *assembly* instructions.

The ICs are interconnected through a network of programmable switches to allow the creation of datapaths. In a similar way to a CPU architecture, the configuration of the ICs and interconnects are changeable on every cycle to execute different blocks of instructions. As shown in Fig. 3, the processing data-path of RICA is a reconfigurable array of ICs, where the program memory contains the configuration bits (i.e., instructions) that control both the ICs and the switches of the interconnects. Special ICs in the core provide the interface to the data and program memories.

Although the RICA architecture is similar to a CPU, the use of an IC-based reconfigurable core as a data path gives important advantages over DSP and VLIWs, such as better support for parallel processing. The reconfigurable core can execute a block containing both independent and dependent assembly instructions in the same clock cycle, which prevents the dependent instructions from limiting the amount of ILP in the program. Other improvements over DSP architectures include reduced memory access by eliminating the centralized register file and the use of distributed memory elements to allow parallel register access.

TABLE II
ICs AND THEIR OPERATIONS

Instruction Cell	Supported Operations
ADD	Addition, Subtraction
MUL	Multiplication (Signed and Unsigned)
DIV	Divisions (Signed and Unsigned)
REG	Registers
I/O REG	Register with access to external I/O ports
MEM	Read/Write from Data Memory
SHIFT	Shifting operation
LOGIC	Logic operation (XOR, AND, OR, etc.)
COMP	Data comparison
MUX	Allows simple branches to be taken
RRC	Controls the rate of reconfiguration
JUMP	Branches (and sequencer functionality)

The characteristics of the RICA core are that it is able to be fully customizable at design time and can be set according to the application's requirements. This includes options such as the bitwidth of the system and the flexibility of the array, which is set by the choice of ICs and interconnects deployed. These parameters also affect the extent of parallelism that can be achieved and device characteristics such as area, maximum throughput, and power consumption. Once a chip containing a RICA core has been fabricated, the system can be easily reprogrammed to execute any code in a similar way to a GPP.

C. ICs

In contrast to other reconfigurable architectures like [3], [4] the IC-array in the RICA is heterogeneous and each cell is limited to a small number of operations as listed in Table II. The cells are not restricted to the primitive operations shown in Table II, although the use of primitive cells compared to multipurpose ALUs can potentially give a higher silicon utilization, since an ALU does not use all of its silicon estate at the same time. Limiting the cells to such simple operations also reduces the number of I/O pins required for each cell (these are typically two inputs and one output as in simple assembly instructions). The use of heterogeneous cells also permits tailoring the array to the application domain by adding extra ICs for frequent operations. Each IC can only have one instruction mapped to it at a time. The instruction cells currently developed support the same instruction sets found in general processors like the OpenRISC [17] and ARM7 [21] but are not limited to these. Hence, with this arrangement, the RICA can be made assembly-level compatible with any GPP/DSP system.

As shown in Table II, registers are defined as standard ICs. With this arrangement, the register memory elements can be distributed in the array in such a way to operate independently, which is essential to allow a high degree of parallel processing. Having distributed registers opens the opportunity to allocate a stack to each register, and hence providing a distributed stack scheme.

As seen in the motivational example, to program the RICA array the assembly code of a program is sliced into blocks of instructions that are executed in a single *step*. Typically, these instructions, which were originally generated for a sequential GPP, would include access to registers for the temporary storage of intermediate results. In the case of the RICA architecture, these read/write operations are simply transformed into wires, which gives a greater efficiency in register use. By using this arrangement of registers the RICA offers a programmable degree of pipelining of the operations and hence it easily permits breaking up long combinatorial computations into several clock cycles if need be.

Special ICs include the JUMP cell which is responsible for managing the program counter and the interface to the program memory in a similar way to the instruction controller found in CPUs. The interface with the data memory is provided by the MEM cells; a number of these cells are available to allow simultaneous read and write from multiple memory locations during the same clock cycle. This is achieved by using multiple ports on independent data memory banks and by clocking it at a higher speed than the reconfigurable core. Since the core uses a relatively low clock frequency, this makes the data memory operate on a clock similar to that used with a conventional DSP or VLIW. Furthermore, some special REG ICs are mapped as I/O ports to allow interfacing with the external environment.

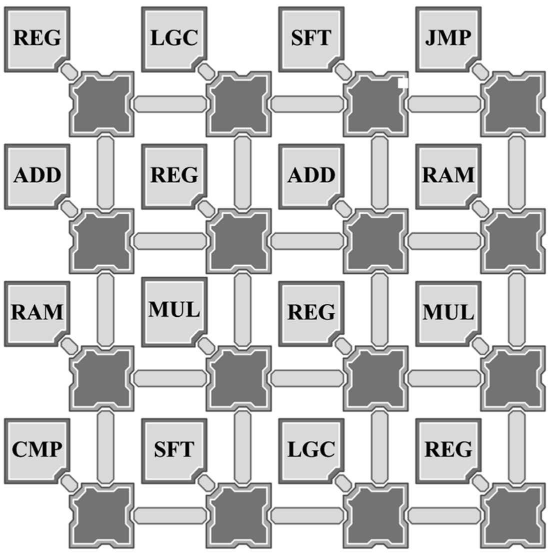
D. Reconfiguration Rate Controller

In usual CPUs, the highest clock frequency at which the processor can be clocked is determined by the longest possible delays in the programmable data-path. For example, if the CPU has a multiplier (which takes a much longer time to execute than operations like addition), then the highest clock frequency has to provide enough time for it to operate. The problem is that if such a clock is used then we might end up with instruction cycles where only an adder is used but there would be unnecessary waiting time than needed, which limits the overall maximum achievable throughput. In conventional CPUs, this problem has to be solved by making the CPU clock at a higher frequency than the one required by the multiplier, and at the same time making the multiplier pipelined, hence requiring multiple cycles to execute. In the RICA architecture a similar problem is encountered since the high flexibility provided allows the creation of data-paths with many levels of calculations, and hence longer possible delay requirements. If the RICA was to be clocked at the highest frequency dictated by the longest data-path, then there would be a restriction on the maximum achievable throughput.

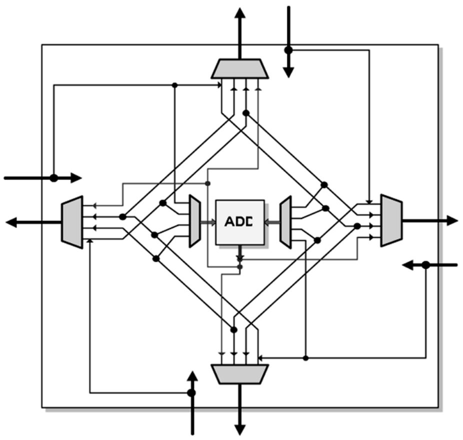
To solve this, we increased the clock frequency of the program counter and registers, and we introduced a new IC termed reconfiguration rate controller (RRC) that generates an *Enable* signal for the program counter and registers (the rest of ICs are not clocked). The amount of clock cycles the RRC waits for before generating the *Enable* signal is programmable as part of the array's configuration. By combining this with the clock-gating technique on the register and program counter, we practically achieve variable clock cycles which are programmable as part of the software. Consequently, longer critical paths wait for more clock cycles before the data is written to the registers.

E. Interconnects

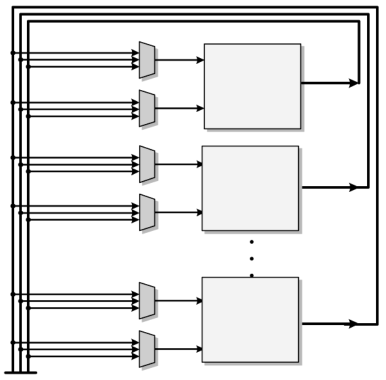
The programmable switches perform directional connections between the output and input ports of the ICs. Different solutions are available for the circuit design and for the topology of the switches, such as multiplexer-based crossbar or the island-style mesh found in typical FPGAs [18] as shown in Fig. 4(a) different interconnect structures have been tested and compared.



(a)



(b)



(c)

Fig. 4. (a) Mesh-based topology. (b) Examples of a switchbox interconnect. (c) Examples of a multiplexer interconnect.

However, this comparison is beyond the scope of this paper. It is worth highlighting that multiplexers are nonblocking, allowing routing any cell's output to any other cell's input, however, there is a penalty in the increased silicon area needed. The island style interconnect structure can have situations where a cell cannot be connected to another cell but it has the benefit of reduced silicon area. Fig. 4(b) shows one possible configurable switch around an instruction cell for island style mesh and Fig. 4(c) of a possible multiplexer interconnect arrangement.

F. Dynamic Reconfiguration and Control Flow

One of the potential hindrances for dynamic reconfiguration is the size of the configuration data to be fetched from program memory and its associated fetch bitwidth requirements. Normally, a large bitwidth instruction fetch mechanism is required to supply multiple instructions per cycle from the memory to the several processing units of the architecture thus determining the configuration time. In the RICA case, this is tackled by several approaches; first, each IC in the array only performs a limited set of operations, thus needing a smaller amount of opcode bits compared to supporting the complete functionality of an ALU.

Additionally, code compression plays a crucial role in reducing the amount of information needed to represent the code by removing redundant information in the configuration and only storing active ICs opcodes. However, code compression in multiple-issue architectures faces extra challenges than single-issue due to the need of decompressing a very large instruction word quickly enough so not to compromise the speed of execution. A code compression technique is used to compress the ICs configuration instructions for the target reconfigurable system, whereas the routing interconnects' configuration instructions are handled independently due to their different redundancy characteristics. The following code compression papers [27], [28] present a more detailed overview of applying code compression techniques for RICA.

The configuration latency of a sequence of contexts is easily hidden using prefetching. The next configuration context can start decoding while the current one is executed. In most such cases, the execution time is larger than the configuration latency. Contexts that can be prefetched are those that either have no branch or the branch is unconditional. A more special case is that of a context that "loops" to itself, in which case the following context can be prefetch and ready to execute once the loop has finished. Of course more sophisticated branch prediction can be applied here as well, but is less crucial when dealing with mostly streaming processing.

In RICA, program control flow, where an instruction when executed can cause a change in the subsequent control flow, can be performed in a couple of ways. First, is through the use of a JUMP cell, which behaves in a similar fashion to a processor. The Jump control flow instruction works by altering the program counter and this in turn causes a new step context to be fetched from memory.

The other approach of handling change of program flow is through the use of multiplexer (MUX) IC. The MUX operation aims to increase the parallelism present in a block of code by reducing the number of conditional jumps. Ordinarily, branches reduce the number of computations that must be performed by skipping sections of code that are not applicable when certain conditions exist. In RICA, often it is more efficient, in terms of speed and energy, to evaluate both branches and then select the appropriate result using a multiplexer operation due to the increased availability of computational resources. In using this technique, attention is paid to the relative costs of evaluating one branch over the other. If one side requires substantially more computation then it is often preferable to leave the jump in place and focus on the body of each branch However, in simple situations, for example where an "if" statement is simply assigning a value, it is still more efficient to map imbalanced branches as multiplexers since the benefits of increased parallelism outweigh the cost of always evaluating both paths.

IV. Power Saving Techniques

This section discusses some of the techniques used in the RICA to reduce power dissipation.

Elimination of Register Files: RICA limits power primarily by using single registers for interim storage rather than multiported register files, and by avoiding large multiplexers and multiple-load buses. This is only part of the story. It also eliminates many register operations depending upon the amount of combinatorial ICs that can be chained into a single operation. Between every pair of ICs, a conventional architecture would perform two read and one write operations to a multiported register file (excluding direct memory access, etc.). We eliminate this register operation completely and use the interconnect fabric to move data from one IC output directly to the input of the next IC in a chain. Only where no further ICs can be chained, for example due to resource limitation, do we register a result, and there we use a single register.

Instruction Fetch From Program Memory: RICA is exceptionally good at executing large steps that loop back to themselves. A good example of such loops is the 8k fast Fourier transform (FFT) used in DVB-T signal processing. A Radix-2 implementation of this computation requires 106 496 iterations of a block that contains 27 instructions (in reality it is two nested loops). On a 32-bit RISC CPU that means fetching $106\,496 \times 27 \times 32 = 92\,012\,544 = 92$ Mbit of data from the instruction memory or instruction cache. RICA, on the other hand, can store the complete configuration of a step statically. The FFT inner-loops requires 1016 bits to be represented on the RICA and fit into a single step. During the execution of these loops, only 26 fetches from the instruction memory occur. This comes down to $26 \times 1016 = 26\,416 = 26.5$ Kbit of data read from the instruction memory. This is 3400× less than the 92 Mbit. Hence, the energy consumed by the program memory of the RICA will be significantly less than that of a RISC. Of course, to achieve such improvement these loops need to be appropriately mapped to the array.

Computation Resources: In a RISC or DSP processor, an ALU will typically perform at least 16 different operations. If you wish to execute an ADD operation, there are many more transistors that are switching in the ALU to select the ADD operation than are required by just the ADD operation alone. In contrast, our add/subtract IC is an optimized circuit that only performs an add/subtract operation, thus reducing this overhead. A small number of extra transistors are required in an interconnect junction to route the result to the next chained cell. Therefore, we contest that we will generally save many more transistor switching operations through the use of optimized single function ICs.

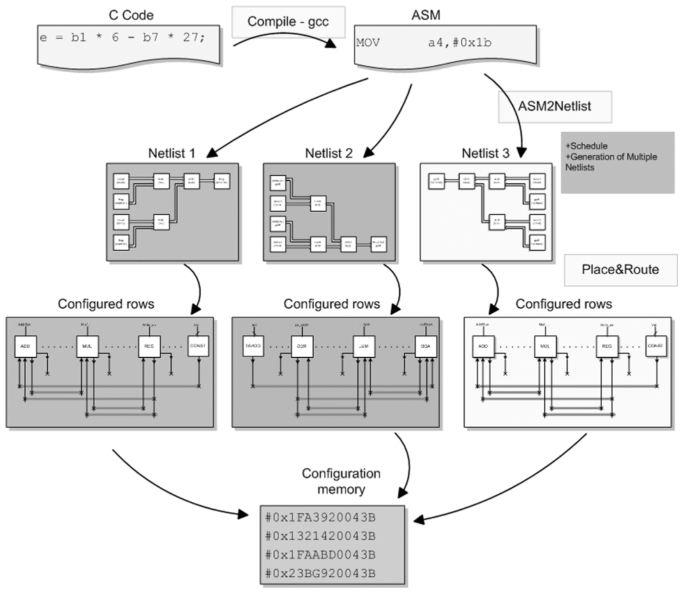
V. Toolflow Overview

There are two main components of software support available for the RICA; the first is the hardware generation of the arrays and the other is a complete flow for programming the arrays from high level languages.

An automatic tool flow has been developed for the hardware generation of the RICA arrays. The tool takes a definition of the available ICs in the array along with other parameters such as their count, positions, bitwidth, and the type of programmable interconnects. The output is a synthesizable RTL definition of the array that can be used in standard system-on-chip (SoC) tool flow for verification, synthesis, layout, and analysis such as power consumption and timing. Alternatively, the specified hardware resource can be modelled using a simulator written in high-level C/C++ code. If the required performances determined by RTL simulation or through the RICA software simulator are not met then the high level code can be modified or the mixture of cell resources changed. Adjusting the hardware resources allows the architecture to be tailored to the specific application domain where it is to be used, thus saving power and reducing unnecessary resources. When array parameters are fixed the generated files are used for fabrication. Consequently, if the algorithm continues to change during or after the fabrication process then the code is simply recompiled for those fixed resources. Often similar operations are required for the updated code when compared to the previous code and a large percentage of the algorithm remains relatively unchanged. This means the generated code is still fairly optimized for the given architecture.

The programming of the RICA architecture is performed with a collection of different tools. As explained earlier, the use of instruction cells greatly simplifies the overall effort needed to map high-level programs to the RICA architecture. Having the arrays programmable in a similar way to standard CPUs allows us to reuse existing developments and methodologies available for processors, such as optimizing compilers and macro assemblers. As can be seen in Fig. 5, the programming flow is split into four main stages.

Step 1) High-Level Compiler: This is the compilation step that takes the high-level code and transforms it into an intermediate assembly language format. This step is performed by a standard open source GNU C Compiler (*gcc*) [19], which compiles C/C++ code (among other front-ends) and transforms it into assembly format describing which ICs need to be used. As *gcc* has grown up around CPU architectures, the output of *gcc* is written with the supposition that instructions are executed in sequence, i.e., one instruction per cycle; the compiler has no knowledge about the parallelism available on the RICA. However, due to the accessibility of the source code for *gcc*, the compiler has been slightly adjusted to take into account some details of the target RICA, like the maximum number of available registers and the available ICs (which define the allowed operations). The compiler automatically deals with issues like register allocation, stack handling and prologue/epilogue definitions. Moreover, it performs all the optimizations that will be useful later-on, like loop-unrolling and loop-peeling in conjunction with loop-fusion.



```
e = b1 * 6 - b7 * 27;
```

```
MOV      a4,#0x1b
```

```
#0x1FA3920043B
#0x1321420043B
#0x1FAABD0043B
#0x23BG920043B
```

Fig. 5. Automatic software flow for programming the RICA, starting from a high-level C program.

(a)

```
1   Load r1, X
2   Load r2, Y
3   Mult r3, r2, r1
4   Load r4, A
5   Mult r2, r4, r1
6   Add r5, r2, r4
7   Mult r1, r2, r5
8   Load r3, B
9   Add r7, r1, r3

6 Registers, 9
cycles
```

(b)

```
Step 1:
   1  Load  wire1, X
   2  Load  wire2, Y
   3  Mult   r1, wire1, wire2
   4  Load  wire3, A
   5  Mult   wire4_r2, wire1, wire3
   6  Add    r3, wire4, wire3
Step 2:
   7  Mult   wire5, r2, r3
   8  Load  wire6, B
   9  Add    r4, wire5, wire6

      4 Registers, 2 cycles
```

Fig. 6. (a) Initial assembly code (b) Scheduled assembly code.

Step 2) RICA Scheduling: In this step all the optimizations related to the RICA architecture are performed. The RICA scheduler process takes the assembly output of *gcc* and tries to create a sequence of netlists to represent the program. Each netlist contains a block of instructions that will be executed in a single clock cycle on RICA, as described in the parallel processing Section III-A. The partitioning into netlists is performed after scheduling the instructions and analyzing the dependencies between them, where dependent instructions are connected in sequence in the netlist while independent ones run in parallel. The scheduling algorithm [23] takes into account IC resources, interconnect resources, and timing constraints in the array; it tries to have the highest program throughput by ensuring that the maximum number of ICs is occupied, and at the same time the longest path delay is reduced to a minimum. Finally, it also performs crucial optimizations like removing the temporary registers generated by *gcc* and replacing them with simple wires. Fig. 6 shows an example of this process. At a later date, the RICA scheduler functionality will be integrated with the gcc compiler to allow enhanced code optimizations.

Step 3) Allocate and Route: As there can be numerous available IC resources to which the assembly instruction can be allocated, a tool is provided to minimize the distance in which cells are connected together. In this process step, if the island style interconnect is selected then a standard place and route tool like VPR [20] can be used to map the netlists into the array. When a multiplexer-based interconnect is selected, the routing is trivial due to the full connectivity.

Step 4) Configuration-Memory: From the mapped netlists, we can simply generate the required content of the configuration memory, in this case, the program RAM.

TABLE III
ICS IN THE SAMPLE ARRAY

Cell	Count	Cell	Count
ADD	4	LOGIC	2
MUL	4	COMP	1
REG	32	JUMP	1
SHIFT	2	MEM	8
DIV	1		

VI. Evaluation of a Sample RICA

A. Sample Array

The sample RICA array chosen for comparison contains the cells listed in Table III. These cells are interconnected using multiplexer-based switches. The mixture of IC resources was manually selected to be adequate for general applications; other combinations can provide better performance depending on the application. These 32-bit cells provide the same basic functionality as an OpenRISC CPU. With the selected type of interconnects and ICs, the reconfigurable core requires a 518-bit wide instruction word. The array was implemented using a UMC 0.13-μ m technology.

The sample RICA was compared to the following DSP architectures: the simple OpenRISC CPU [17] implemented on UMC 0.13-μm technology, the ARM7-TDMI-S [21] again on 0.13-μ m technology, the TI C55X [24] two-way datapath low-power DSP, and the powerful TI64X 8-way VLIW [16]. The benchmarks are mainly based on TI's benchmarks for the TI C64X. All the benchmarks are direct unoptimized C representations of the algorithms—all optimizations are left for the C compilers (Level-3/O3). The compiler used for the RICA did not include any advanced techniques like predications or the use of rotating register as the compiler provided by TI does. All benchmarks include memory transfers, stack control, and function's prologue and epilogue and hence they show a representative evaluation of the architecture's execution performance.

For the RICA and OpenRISC, the power and area were found using post-layout simulations on PrimePower from Synopsys. The ARM7 datasheet [21] provides power and area values of the ARM core in 0.13-μm technology, while [22] and [25] allows us to estimate the power consumption of just the datapaths in the TI C64x and TI C55x. All these power estimations were measured at 1.2-V operating voltage and only focus on the energy consumed in the data path without the memory. The area of the data path in the TI C64x was estimated using proportionality from the published die-photo [26] knowing that the whole chip has 64 M transistors (no cache memory was included). No area information was available for the C55x. Table V also includes variations in program size, as they differ for each architecture and compiler technology used. The size of the data RAM is the

TABLE IV
COMPARISON OF DATAPATH AREA IN MILLIMETERS SQUARED ON 0.13-μm OF VARIOUS CPUs EXCLUDING VARIATIONS IN PROGRAM MEMORY

	RICA	OR32	ARM7	C55x	C64x
Datapath Area	1.90	0.25	0.32	N/A	2.01

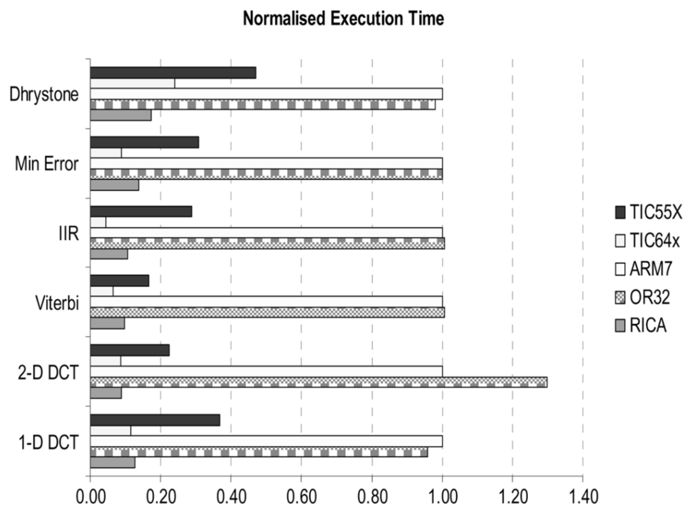


Fig. 7. Normalized execution time graph of the benchmarks on RICA and other architectures.

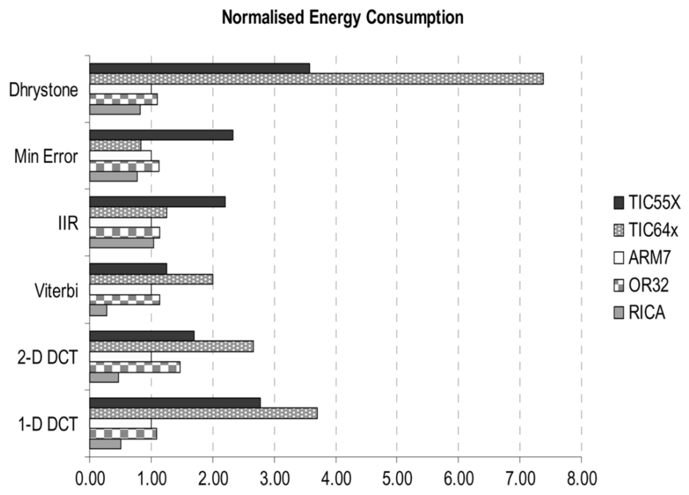


Fig. 8. Normalized energy consumption graph of the benchmarks on RICA and other architectures.

same for all processors, and hence it is not included in the comparison. The Dhrystone benchmark, which today has become an outdated measurement, is included here for reference. As shown in Table V, the fact that the Dhrystone takes more cycles to run on the highly pipelined TI DSPs than on the ARM7 shows how inappropriate a benchmark it is for modern processors.

B. Results

The results are listed in Tables IV and V and Figs. 7 and 8.

From the tables, we can see that for all the benchmarks we achieve better performance on the RICA than on the conventional OR32 and ARM7 CPUs; we get around 1–3.6× less energy consumption while achieving around 5–8× higher maximum throughput. Due to the increase in program memory size and the increase in the datapath area, the power and throughput improvements come at the cost of an area increase of around 7×. A big part of the power reductions achieved over the four DSP systems are savings gained by eliminating the register-files and having distributed registers.

When compared to the low-power C55X DSP, RICA achieves a promising reduction in energy consumption between 2 to 6× while achieving a throughput of up to 3× higher. RICA achieves similar timing performances to the VLIW for applications containing significant datapath operations such as the DCT, while faster operation is seen for Dhrystone. For benchmarks that have a lot of independent blocks and control parts (i.e., small loops and comparisons) like minimum error, RICA is around 50% slower than the 600-MHz VLIW—this is expected as the TI compiler can optimize such code by using techniques like predication in a better way than *gcc*. For the Viterbi and IIR, RICA was around 20%–30% slower with the bottleneck being the memory access. However, for the case of the Viterbi, the *gcc* compiler was able to correctly identify the use of multiplexers ICs which improved speeds and reduced branching. It should also be noted that the RICA is built from synthesizable standard-cell libraries while the circuits in the VLIW have been manually laid out to achieve the 600-MHz operating frequency. In terms of energy, around 6× less power is consumed for DCT, Viterbi, and Dhrystone, since the RICA goes less into states where ALUs are idle but consuming power than the VLIW. The power reductions for the minimum error and IIR benchmarks were lower at around 17%. In terms of area, the datapaths of the RICA and VLIW are similar.

VII. CONCLUSION

In this paper, we presented a new coarse-grain reconfigurable computing architecture that offers comparable computation performance to leading DSP processors with a significant reduction in power consumption. The development of RICA was progressed bearing in mind the problem of taking a high-level description of an algorithm and mapping it directly into the reconfigurable fabric, as well as having an integrated control flow handling to enable it to function independently to a standard processor, unlike some reconfigurable fabrics. This approach led to the adoption of an instruction-cell-based architecture, making RICA compatible with instruction representations of programs, and thus allowing seamless integration to existing well established software design tool-flows. This is RICA's distinctive strength over other reconfigurable architectures that lack a straightforward programming interface.

The architecture demonstrates good results regarding the four important requirements for future systems: low-NRE costs, low-power consumption, high-flexibility, and straightforward design-flow. Unlike conventional reprogrammable FPGAs, the RICA architecture is easily reprogrammed through high-level languages like conventional processors, which is done using existing compilers technologies such as the standard GNU Compiler. The RICA outperforms current low-power DSP architectures such as the TI C55x by providing up to 3× higher throughputs but with 2–6× less power consumption at a cost of increased program size. It should be noted that the full extent

TABLE V
COMPARING RICA WITH OTHER PROCESSOR, LOW-POWER DSP, AND VLIWS USING BENCHMARKS

	RICA				OpenRISC CPU (on UMC 0.13µm) - 112MHz				ARM7-DTMI-S (Syn. on 0.13µm) - 110 MHz			
	RRC Cycles	Min Execution Time (us)	Raw Code (bytes)	Energy per Op (nJ)	Cycles	Min Execution Time (us)	Code size (bytes)	Energy per Op (nJ)	Cycles	Min Execution Time (us)	Code size (bytes)	Energy per Op (nJ)
1-D DCT	43	0.12	993	**4.7**	102	0.91	402	**10.2**	104	0.95	406	**9.36**
2-D DCT	1351	3.01	1785	**159.3**	4972	44.39	516	**497**	3760	34.18	508	**338**
Viterbi	1838	7.78	1286	**218.3**	9032	80.64	308	**903**	8803	80.03	316	**792**
IIR	120	0.17	755	**16.33**	180	1.61	510	**18**	176	1.60	464	**15.8**
Min Error	5164	11.10	1070	**620.1**	9073	81.01	442	**907**	8908	80.98	412	**802**
Dhrystone	798	1.12	1289	**52.57**	711	6.35	870	**71.1**	712	6.47	912	**64.1**

	TI C64x 8-ways VLIW - 600MHz				TI C55x 2-way low-power DSP - 300 MHz			
	Cycles	Min Execution Time (us)	Code size (bytes)	Energy per Op (nJ)	Cycles	Min Execution Time (us)	Code size (bytes)	Energy per Op (nJ)
1-D DCT	68	0.11	316	**34.68**	104	0.35	451	**26**
2-D DCT	1763	2.94	588	**899.1**	2300	7.67	655	**575**
Viterbi	3120	5.20	664	**1591**	3980	13.27	262	**995**
IIR	39	0.07	160	**19.89**	139	0.46	436	**34.8**
Min Error	1320	7.20	952	**673.2**	7479	24.93	380	**1870**
Dhrystone	928	1.55	424	**473.3**	916	3.05	1021	**229**

of power savings depends on the amount of control operations in the program.

When compared to current VLIW processors, RICA considerably reduces the number of required clock cycles in applications containing numerous dependent instructions since it allows the execution of both dependent and independent instructions in the same cycle, which overcomes the problem of statistical ILP-limit faced by VLIW. In terms of performance, RICA achieves similar timing to the VLIW for datapath applications, while being up to 50% slower in control intensive applications. This is due to the fact that the VLIW circuitry has been handcrafted to achieve 600-MHz operating frequency. Nevertheless, RICA can achieve up to 6× less power than the VLIW.

The measured performance of the initial array are encouraging, however, more changes can be done on the compiler level, such as making the scheduling occur inside *gcc*, to greatly boost the performance.

REFERENCES

[1] G. Estrin, "Organization of computer systems—The fixed plus variable structure computer," in *Proc. Western Joint Comput. Conf.*, 1960, pp. 33–40.

[2] E. Mirsky and A. DeHon, "Matrix: A reconfigurable computing architecture with configurable instruction distribution and deployable resources," in *Proc. IEEE Symp. FPGAs Custom Comput. Mach.*, 1996, pp. 157–166.

[3] J. R. Hauser, "Augmenting a microprocessor with reconfigurable hardware," M.S. thesis, Comput. Sci. Dept., Univ. California, Berkeley, 2000.

[4] Elixent Ltd., Bristol, U.K., "D-Fabrix processing array, reconfigurable signal processor," 2005 [Online]. Available: www.elixent.com

[5] XPP, PACT, Munich, Germany, "OFDM decoder for wireless LAN—Whitepaper," May 2002 [Online]. Available: ww.pactcorp.com

[6] Philips, Avispa, Eindhoven, The Netherlands, "Reconfigurable computing," 2005 [Online]. Available: www.siliconhive.com

[7] P. M. Heysters, G. J. M. Smit, and E. Molenkamp, "Montium—Balancing between energy-efficiency, flexibility and performance," *Eng. Reconfig. Syst. Algorithms*, pp. 235–241, 2003.

[8] M. Wan, H. Zhang, V. George, M. Benes, A. Abnous, V. Prabhu, and J. Rabaey, "Design methodology of a low energy reconfigurable single-chip dsp system," *J. VLSI Signal Process.*, vol. 28, pp. 53–63, 2000.

[9] H. Singh, M. Lee, G. Lu, F. Kurdahi, N. Bagherzadeh, and E. Filho, "MorphoSys: An integrated reconfigurable system for data-parallel and computation-intensive applications," *IEEE Trans. Comput.*, vol. 49, no. 5, pp. 465–481, May 2000.

[10] S. Khawam, S. Baloch, A. Pai, I. Ahmed, N. Aydin, T. Arslan, and F. Westall, "Efficient implementations of mobile video computations on domain-specific reconfigurable arrays," in *Des. Autom. Test Eur. (DATE)*, 2004, pp. 1230–1235.

[11] B. Mei, S. Vernalde, D. Verkest, H. De Man, and R. Lauwereins, "ADRES: An architecture with tightly coupled VLIW processor and coarse-grained reconfigurable matrix," in *Proc. 3rd Int. Conf. Field-Program. Logic Appl.*, 2003, pp. 61–70.

[12] C. Ebeling, C. Fisher, G. Xing, M. Shen, and H. Lui, "Implementing an OFDM receiver on the rapid reconfigurable architecture," *IEEE Trans. Comput.*, vol. 53, no. 11, pp. 1436–1448, Nov. 2004.

[13] Stretch, Sunnyvale, CA, "Configurable processor," 2007 [Online]. Available: www.stretchinc.com

[14] R. Baines and D. Pulley, "A total cost approach to evaluating different reconfigurable architectures for baseband processing in wireless receivers," *IEEE Commun. Mag.*, vol. 41, no. 1, pp. 105–113, Jan. 2003.

[15] Ambric, OR, "Programmable multicore," 2007 [Online]. Available: www.ambric.com

[16] S. Agarwala *et al.*, "A 600-MHz VLIW DSP," *IEEE J. Solid-State Circuits*, vol. 37, no. 11, pp. 1532–1544, Nov. 2002.

[17] Opencores, Dobrova, Slovenia, "OpenRISC," (2005). [Online]. Available: http://www.opencores.org/projects.cgi/web/or1k

[18] J. Rose and S. Brown, "Flexibility of interconnection structures for field-programmable gate arrays," *IEEE J. Solid-State Circuits*, vol. 26, no. 3, pp. 277–282, Mar. 1990.

[19] GNU, Boston, MA, "GNU C compiler," 2005 [Online]. Available: http://gcc.gnu.org/

[20] V. Betz and J. Rose, "VPR: A new packing, placement and routing tool for FPGA research," in *Proc. 7th Int. Workshop Field-Program. Logic*, 1997, pp. 213–222.

[21] ARM Ltd., Cambridge, U.K., "ARM7 thumb family datasheet," ARM DOI 0035-3/02.02, 2002.

[22] G. Martinez, "TI TMS320VC5501/02 power consumption summary," Appl. Rep. SPRAA48, 2004.

[23] Y. Yi, I. Nousias, M. Milward, S. Khawam, T. Arslan, and I. Lindsay, "System-level scheduling on instruction cell based reconfigurable systems," in *Proc. Des. Autom. Test Eur. Conf. (DATE)*, 2006, pp. 381–386.

[24] Texas Instruments Incorporated, Dallas, TX, "TMS320C5000 CPU and instruction set reference guide," 2000.

[25] G. Martinez, "TMS320VC5501/02 power consumption summary," Appl. Rep. TI SPRAA48, 2004.

[26] D. Wentzlaff, "Architectural implications of bit-level computation in communication applications," M.Sc. thesis, Dept. Elect. Eng. Comput. Sci., Massachusetts Inst. Technol., Boston, 2002.

[27] N. Aslam, M. Milward, I. Nousias, T. Arslan, and A. Erdogan, "Code compression and decompression for instruction cell based reconfigurable systems," presented at the IEEE Int. Parallel Distrib. Process. Symp., Reconfigurable Arch. Workshop, Long Beach, CA, 2007.

[28] N. Aslam, M. Milward, I. Nousias, T. Arslan, and A. Erdogan, "Code compressor and decompressor for ultra large instruction width coarse-grain reconfigurable systems," in *Proc. IEEE Symp. Field-Program. Custom Comput. Mach.*, 2007, pp. 297–298.

Sami Khawam received the B.Eng. degree in electrical and electronics engineering and the Ph.D. degree in system level integration from the University of Edinburgh, Edinburgh, U.K., in 2001 and 2006, respectively.

Presently, he is a Senior Hardware Engineer with SpiralGateway, Edinburgh, U.K. His research interests include low-power VLSI hardware, reconfigurable hardware architectures, and embedded reconfigurable hardware.

Ioannis Nousias received the B.Sc. degree in electrical and electronic engineering from the Cretan University, Chania, Greece, in 2001, and the M.Sc. degree in system level integration from the University of Edinburgh, Edinburgh, U.K., in 2003, where he is currently pursuing the Ph.D. degree in reconfigurable computing.

In June 2007, he became a Senior Engineer with Spiral Gateway LTD, Edinburgh, U.K., where he continues his work in reconfigurable computing technologies. In 2001, he was a Senior Hardware and Software Engineer with Protogenea Inc., Athens, Greece, where he developed a GPS- and GSM-based tracking and monitoring system for industrial and small business applications, such as terrestrial map alignment and remote monitoring of containers with sensitive shipments. His research interests include reconfigurable computing, bioinspired technologies, artificial intelligence, interconnection technologies, network on chip, and micro-processor architectures."

Mark John Milward received the B.Eng. degree in electronic and electrical engineering and the Ph.D. degree in the area of parallel lossless compression from Loughborough University, Leicestershire, U.K., in 2000 and 2004, respectively.

Currently, he is a Senior Engineer with Spiral Gateway Ltd., Edinburgh, U.K., where he is continuing his work on reconfigurable systems. He was a Research Associate with the System Level Integration Group, University of Edinburgh, Edinburgh, U.K. His research interests include reconfigurable architectures, parallel hardware architectures, and lossless compression.

Ying Yi received the B.Sc. degree in computer and application from Harbin Engineering University, Harbin, China, and the Ph.D. degree from the Queen's University, Belfast, U.K., in 1996 and 2003, respectively.

Currently, she is a Research Fellow with the School of Engineering and Electronics, University of Edinburgh, Edinburgh, U.K. She was a Software Engineer with the WuHan Institute of Mathematical Engineering, China, where she researched, developed, and maintained intranet and management information systems. From 1997 to 2000, she was with the China Ship Research and Development Academy (CSRDA), Beijing, China, where she was involved in research, development, and management of computer correlative problems. Her research interests include low-power reconfigurable SoC systems, compiler optimization techniques for reconfigurable architecture, architectural level synthesis optimization, and multiprocessor SoC.

Mark Muir received the M.Eng. degree (first class) in electrical and mechanical engineering and the Ph.D. degree from the University of Edinburgh, Edinburgh, U.K., in 2004, where he developed software tools for the RICA project.

He is currently writing up, and is an employee of Spiral Gateway, Edinburgh, U.K., which licenses this technology. His research includes a broad range of interests, with particular emphasis on software development and algorithms.

Tughrul Arslan holds the Chair of Integrated Electronic Systems with the School of Engineering and Electronics, University of Edinburgh, Edinburgh, U.K., and is also a cofounder and the Chief Technical Officer of SpiralGateway Ltd., Edinburgh, U.K. He is a member of the Integrated Micro and Nano Systems (IMNS) Institute and leads the System Level Integration Group (SLIg) in the University. his research interests include low-power design, DSP hardware design, system-on-chip (SOC) architectures, evolvable hardware, multiobjective optimization, and the use of genetic algorithms in hardware design issues.

Prof. Arslan is an Associate Editor for the IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS—I: REGULAR PAPERS, a member of the IEEE CAS Committee on VLSI Systems and Applications, and sits on the editorial board of IEE Proceedings on Computers and Digital Techniques and the technical committees of a number of international conferences. This year he is the general Chair of the NASA/ESA Conference on Adaptive Hardware and Systems, and Co-Chair of ECSIS Bio-inspired, Learning, and Intelligent Systems for Security Symposium (BLISS). He is a principal investigator on a number of projects funded by EPSRC, DTI, and Scottish Enterprise together with a number of industrial and academic partners.