REDEFIS : A System with a Redefinable Instruction Set Processor

Goulart Ferreira, Victor M.
Fukuoka Laboratory for Emerging & Enabling Technology of SoC (FLEETS)

Gauthier, Lovic
Fukuoka Laboratory for Emerging & Enabling Technology of SoC (FLEETS)

Kando, Takayuki
Fukuoka Laboratory for Emerging & Enabling Technology of SoC (FLEETS)

Matsuo, Takuma
Tokyo Electron Limited

他

出版情報：Proceedings of the annual symposium on Integrated circuits and systems design. 19, pp.14-19, 2006-08
バージョン：
権利関係：

REDEFIS — A System with a Redefinable Instruction Set Processor

Victor M. GOULART FERREIRA † , Lovic GAUTHIER † , Takayuki KANDO † ,
Takuma MATSUO ‡ , Toshihiko HASHINAGA§, and Kazuaki MURAKAMI§
† Fukuoka Laboratory for Emerging & Enabling Technology of SoC (FLEETS)
‡ Institute of Systems & Information Technologies/KYUSHU (ISIT)
§ Department of Informatics, Kyushu University
Fukuoka, Japan
E-mail: redefis@fleets.jp

ABSTRACT
The growing complexity and production cost of processor-based systems have imposed big constraints in SoC design of new systems. GPPs and ASICs are unable to fit the tight performance or power constraints, or too complex to design in short TAT/TTM. REDEFIS is a HW/SW design platform for high level, efficient implementation of ASIPs/engines for SoC systems. It is composed of a reconfigurable instruction-set processor, capable to redefine its ISA according to the user application written in high level C language, and a set of design tools (an ISA Generator and a retargetable compiler). These processors can be used as flexible engines in embedded MPSoC systems, where its ISA is fully customized and design is done at high level C (no HDL writing is necessary). In this paper we present the Redefis design platform and an implementation of our dynamically reconfigurable ISA processor (codename Vulcan). Our results demonstrate the effectiveness of the system for encryption and bitwise applications.

Categories and Subject Descriptors
C.1.3 [**Processor Architectures**]: Other Architecture Styles - adaptable architectures. C.3 [**Special-purpose and Application-based Systems**]: Real-time and Embedded Systems.

General Terms
Design, Performance.

Keywords
Dynamically reconfigurable processor, ISA customization, high performance, low power, SoC.

1. INTRODUCTION
Convergence within consumer electronics like mobile phones with digital cameras, media players and security embedded features require great processing power with energy efficiency which general-purpose processors (GPP) can hardly deliver. The ever increasing demand for high performance, feature-rich products have been driving the industry and research communities to develop design platforms which permit to develop more efficient custom logic solutions in a short time frame. The tighter TAT and TTM constraints for a whole ASIC design solution are not affordable either due to its complexity and cost. Instead of designing a new ASIC from scratch, ASIPs and reconfigurable computing (RC) systems are becoming very attractive solutions with a plethora of systems based on (re-)configurable processors / engines with tools support from compilers up to the OS. Some commercial systems include Altera's NIOS II [3], IPFlex's DAPDNA2 [4], PACT XPP dynamically reconfigurable processor array [5], Tensilica's Xtensa processors [6], Stretch processors [7], Toshiba's MeP processor [8], NEC's Dynamically Reconfigurable Processor (DRP) [9], Elixent's (RAP - Reconfigurable Algorithm Processing) [10], Morpho Technologies' MS2 [11], and others.

In this research we focus on systems with the flexibility and application suitability of extensible or (re-)configurable ASIPs which we call **Redefinable Instruction Set (REDEFIS)** platform [1]. The REDEFIS is a HW/SW design platform for high level, high performance, and fast implementation of ASIPs. According to the intrinsic characteristics of an application the development tools automatically identify and completely redefine a new ISA (Instruction Set Architecture) to better fit the target application. Compared to other approaches based of reconfigurable fabrics/processors, the Redefis project targets processors where the ISA is fully redefinable and not extended. Redefis also proposes a design flow based on standard C programs (no HDL writing is necessary).

The Redefis platform is composed of a reconfigurable instruction-set processor and a set of software design tools. These processors can be used as flexible engines in embedded MPSoC systems.

The remainder of this paper is organized as follows: Section 2 presents and overview of the Redefis system, followed by a more detailed description of the reconfigurable processor (Vulcan) in Section 3. The experiment framework, evaluation and experimental results of the architecture are presented in Section 4. Final discussions and conclusions are given in Sections 5 and 6.

2. REDEFIS SYSTEM PLATFORM
The Redefis design tool chain is used both to compile an input C program (the "application" written in high level C) and to produce an ISA specifically optimized for this given input application. The object code for this application will use instructions of the ISA and the processor will reconfigure itself in order to execute each specific CI (Custom Instruction). The general overview of the system can be seen in Figure 1(a).

Figure 1(b) shows the design flow of the Redefis platform. The input is a program in high level C, and the output is both the binary object code of the program and the newly defined ISA as configuration bit-streams generated from the mapping of the CI's into Vulcan's Reconfigurable Data Path – RDP (explained later). The design tool chain is composed of:

- **ISAGen** [2]: gets an input C program and outputs the HW description of the ISA, "ISA unmapped" in Figure 1(b), and a C program ("ISA-C program") referencing this new ISA.

SBCCI'06, August 28–September 1, 2006, Minas Gerais, Brazil.

- **Place & Route tool**: places and routes each CI of the ISA onto the RDP of the Vulcan processor.
- **Retargetable Compiler**: compiles the ISA-C program to Vulcan assembly.
- **Assembler/Linker**: generates the object code (based on the instruction format of Figure 3) and appends to it the ISA configuration data.

An important part in the tool chain of the system is the **ISAGen** (**I**nstruction **S**et **A**rchitecture **Gen**erator) [2]. ISAGen will decompose the input C program, analyze it and produce a number of CI's based on its DFG which will compose the computation CI's of the new ISA. It will also preprocess the C program to reflect the generated set of instructions before going into the retargetable compiler. This compiler will generate the rest of the ISA composed of CI's for flow control specifically optimized for this application. The retargetable compiler compiles the resulting C program into the target (re-)configurable processor's object code, which can be ASIPs (Application-Specific Instruction Set Processors); Multi-Processor SoC or **Vulcan**, a dynamically reconfigurable instruction-set processor.

The description of the internals of ISAGen and the retargetable compiler is out of scope of this paper. Here we focus on the architecture and implementation of Vulcan, a processor capable to redefine and execute different ISA's specific to each application. Vulcan details are going to be presented in the next section.

3. REDEFINABLE ISA PROCESSOR

Vulcan is one implementation of a dynamically reconfigurable processor in the Redefis Project, where the ISA is fully redefinable (at compile time) and the execution of its instructions is made by reconfiguring the processor data-path for every custom instruction according to program execution flow.

3.1 Outline of Vulcan Processor

Vulcan executes the application by configuring its data-path over and over according to program execution flow. Throughout the program execution the RDP changes its "function" in order to execute a given instruction. Execution "*step*" refers to the processing of reconfiguration of the RDP and the computation of the configured instruction altogether, which involves the amount of time it takes to perform these tasks; each step's number of cycles can vary according to the complexity of the CI.

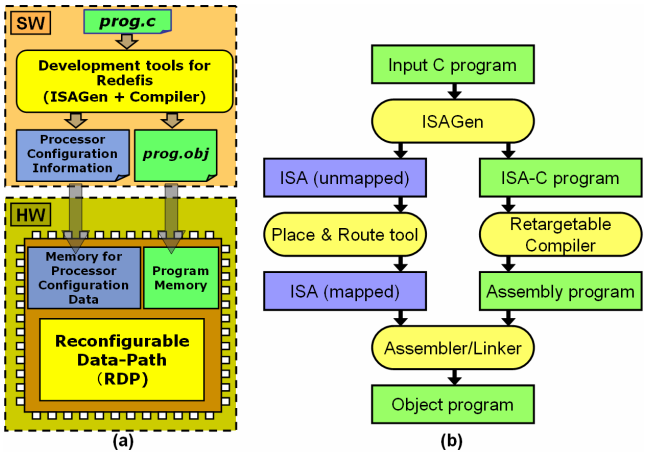

Figure 1: (a) Overview of Redefis HW/SW Platform; (b) Redefis design tool chain

Configuration data or bit-stream is equivalent to FPGA systems; in Redefis it is used to "reprogram" the reconfigurable module (Reconfigurable Data Path – RDP). We call "custom instructions" those Vulcan maintains and is capable to execute (there is a total of 128 CI's attainable due to its *CF_num* field (7-bits) in its instruction format – see Figure 3). The full set of configurations loaded into the configuration memory is the current ISA of the processor (generated at compile time). Moreover the user can set up the working cycle time for each instruction, i.e., different CI's can work at different cycle times, which is made by dividing the main clock cycle (*Work_Rate* field).

An overview of Vulcan architecture is presented in Figure 2. Vulcan's computation power and flexibility is due to its RDP module which is a reconfigurable network of processing elements (PE's). Additional calculation units are present for more demanding arithmetic computations like adders and multipliers (two 16-bit each). The controller is responsible for fetching instructions, reconfiguring the RDP and handling flow changes (e.g. branching). In the processor every "custom instruction" (identified from the application's source code) is associated with one configuration of the RDP (*CF_Num*). A typical execution cycle of Vulcan is performed in 3 steps:

1) The controller fetches a CI and reconfigures the RDP.
2) The RDP reads the data and passes it to be computed in the RDP (for every custom instruction).
3) The RDP writes back the result of the calculation.

3.2 Dynamically Reconfigurable Processor

The Vulcan processor is our first implementation of a family of reconfigurable processor architectures under research. In the following we explain in more details the architecture and its parts. Vulcan's architecture (Figure 2) is composed of a reconfigurable fabric, or reconfigurable data-path structure (RDP), a number of register files and memories (configuration memory, program memory, data memory), and a very simple in-order control unit where instructions are executed sequentially. Explanation of each unit follows:

3.2.1 Controller

It manages the execution of the entire system such as loading of instructions, configuration data and load/store of data in memory. Seven index registers of 22bits are installed internally, and it is also possible to access the memory by using the value in the index register. Moreover, the values in these index registers can connect it with the stack.

3.2.2 Memory System

The main memory of the system is composed of 32MB of SDRAM off-chip; each word is 64-bit long and is used to hold both program and data. On-chip, there are two caches (P$ - program cache; D$ - data cache) whose sizes are 8 words x 32 sets.

3.2.3 Instruction Format and Types

The instruction format of Vulcan is presented in Figure 3. Differently from the "normal" processor, every instruction in Vulcan specifies a number which references the location where the configuration data for each CI can be found and then executed. It is possible to specify source and destination operands inside each instruction. Moreover there are *computation instructions* and *control instructions* indicated by *Exe_Non* field where RDP reconfiguration takes place or not accordingly. When a branch occurs the condition can be specified (*Flow_Param* fields) as well as the relative address of jump instructions (*Rel_Adr* field).

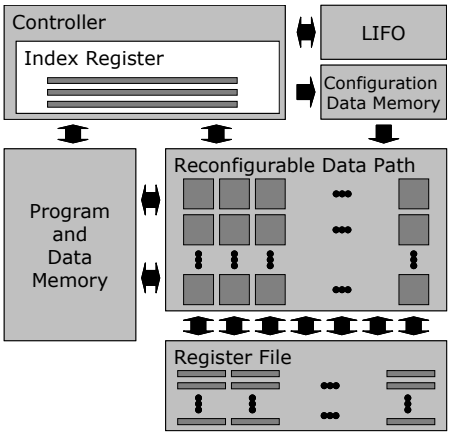

Figure 2: Overview of Vulcan Architecture

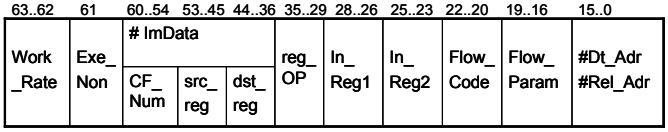

Figure 3: Instruction Format of Vulcan

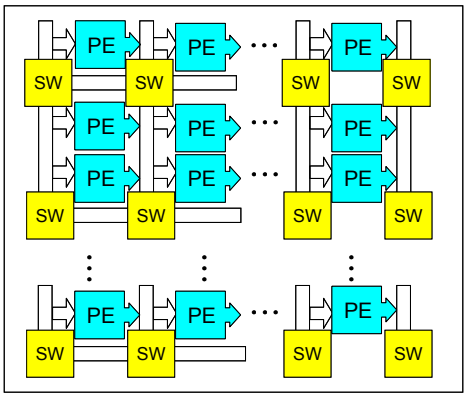

Figure 4: Reconfigurable Data Path (RDP) structure.

3.2.4 Configuration Memory

It is where the configuration data is stored. It can store the configuration data of up to 128 custom instructions. The bit-stream size for each configuration or CI is about 3.8KB maximum i.e., the configuration of the whole RDP, all PE's and SW's.

3.2.5 Register Files

There are two types of Register Files, one called IFRAM and other called SHRAM. The IFRAM is composed of differently addressable 512 x 256-bit registers. However each bit of these registers can be directly attached to different PE's in the RDP, which might be very powerful for bit shuffling (it can be done by just selecting which PE's are going to be used), though most of times it is hard to handle it because of the P&R tool complexity. Another RF is the SHRAM with 2K x 64-bit registers. The SHRAM registers can also be addressed by index registers.

3.2.6 Reconfigurable Data Path (RDP)

Figure 4 shows the organization of the PE's (Programmable Elements) and SW (programmable switches) in the RDP. PE's are arranged in 16 rows and 8 columns. Each PE is a 6-to-2 LUT (Look-Up Table) which can implement an arbitrary logic function of its inputs. Every PE is connected by wide buses in vertical and horizontal directions; Vertical Lines (VL) and Horizontal Lines (HL) respectively. Each switch specifies the connection between VL and HL, like if a given signal is to be transferred from HL to VL or vice-versa. VL is composed of 8 buses of 64-bits while the HL, 7 buses of 64-bits, where each bit can be connect to any other bit of a different orthogonal bus. The RDP can directly load/store a 64-bit data from/to memory at once.

3.3 Execution Flow

In this subsection the execution flow of Vulcan is explained. First, the controller reads the program from main memory to the cache; if it is a *computation instruction*, the appropriate configuration data is loaded from the address of the configuration data memory specified in the instruction (the *CF_Num* field). After the RDP is reconfigured the input data is processed. The execution result of each PE is output to VL, writing the respective bits in the destination register, and it is also possible to hand over data to another instruction as an intermediate result. By using these registers efficiently big custom instructions can be executed by dividing it into two or more instructions. Moreover, if it is a *control instruction* the program counter gets the value dictated by the operation indicated in the instruction word.

This way, shuffling, masking and other expensive bitwise operations in GPP can be implemented very efficiently in Vulcan. Due to the dynamic execution flow inherent to Vulcan, the RDP has only 128 PE's in order to minimize the impact on clock time.

4. EXPERIMENTS

In the remaining sections we are going to evaluate the performance and execution flow of Vulcan processors compared to GPP's and to consider how fast and cost effective the Redefis platform can be when designing ASIPs. Another objective is to gather information about the bottlenecks in the architecture for further improvements as we have chosen a "limit situation" of reconfiguration at every "step" of execution.

For the Vulcan processor we have implemented an ISS (Instruction Set Simulator), an emulator of the Vulcan processor on an FPGA board (Figure 5) and a chip implementation as well (3M gates, 80MHz@0.11µm). We are using both simulated (PISA simulator of Simplescalar tool set [12], Table 1) and the Intel Pentium 4 in order to discuss performance and efficiency numbers.

4.1 Evaluation

In order to evaluate the Redefis system we have implemented the image compression algorithm JPEG, the cryptographic algorithms DES, AES, and IDCT (Inverse Discrete Cosine Transform). We have adopted these because of their broad use in security and media applications.

In our experiments we have emulated the Vulcan using the hardware emulation board (Figure 5). In this board there are two Xilinx FPGAs (XC2V8000), where the functionality of the Vulcan processor is implemented. In order to implement each application we described each PE (Processing Element) and their interconnections and used a place and routing tool for Vulcan in conjunction with its assembly which describes their custom instructions. A 32MB flash card is used as main memory. We now describe the implementation of each application.

4.2 Application Set

4.2.1 JPEG Image Compression

JPEG [15] is a broadly used algorithm for image compression used in digital photography, medical imaging, internet, remote sensing and surveillance, among other applications. We got a freely available source code and mapped it into Vulcan.

Table 1: Simplescalar parameters

Instruction issue method	OOO (Out-of-order)
ISA set	PISA
Branch predictor	
Type BTB size	2 levels (bimod, 2K entries) 512 entries, 4-way
Inst.Issue/decode width	4 inst/clock cycle (c.c.)
IFQ size	4 entries
RUU size	16 entries
LSQ size	8 entries
Cache Memory (Size, Latency)	
L1 (Data cache) L1 (Instruction cache) L2 (Shared cache)	32KB (4-way, 128 entries), 1 cc 32KB (1-way, 512 entries), 1 cc 64KB (4-w., 1024 entries), 6 cc
Memory bandwidth	8 Bytes
Num. memory ports	2
Integer oper. units (# of ALUs, delay, issue latency)	
ALU Multiplier Divider	4, 1 c.c., 1 c.c. 1, 3 c.c., 1 c.c. 1, 20 c.c., 19 c.c.

4.2.2 DES Encryption Algorithm

The Data Encryption Standard (DES) [13] is a cipher (a method for encrypting information) selected as an official Federal Information Processing Standard (FIPS) for the United States, in 1976. The original DES algorithm (Figure 6) was processed and then reduced to 6 big "functions" or custom instructions (Figure 7), i.e., the DES encryption algorithm could be implemented in Vulcan with an ISA of just 6 instructions!

4.2.3 AES Encryption

The Advanced Encryption Standard (AES) [14], also known as Rijndael, is a block cipher was adopted by National Institute of Standards and Technology (NIST) as US FIPS PUB 197 in November 2001, after its predecessor, DES, became obsolete.

AES algorithm can be seen in the execution flow below (Figure **8**). Both the key and block sizes can be of 128, 192, and 256 bits. It is a conventional encryption algorithm which uses the same key for both encryption and decryption. In our experiments we used both key and block sizes of 128 bits. In the AES encryption, the plaintext is delimited by units of 8 bits (4x4 rows) for input, and then the round function will process this unit to obtain the ciphertext. When the key is 128 bits the number of rounds is 10. In each round the following four computations are done:

- *Add-Round-Key*: Exclusive-OR of input and round key
- *Mix-Column*, *Shift-Row*: Homogeneous transformation like bit shift, etc
- *Byte-Sub*: a non-linear transformation called S-Box.

Mix-Column is not done in the tenth iteration of round. Moreover, the round key is generated by using the bit shift and the S-Box conversion on every round.

In order to implement it in Vulcan we got the source code (C program), analyzed it and generated the CI's and the assembly code of the whole program by hand. We tested the resulting assembly program in Vulcan's ISS. Now, we used the 32-bit PISA OOO – out-of-order simulator from the Simplescalar tool set [12] to compare the results. In Table 1 one can see the description of the parameters used. We used the same program as to the Vulcan implementation; in Simplescalar the AES was compiled by gcc-2.6.3 with optimization option -O2.

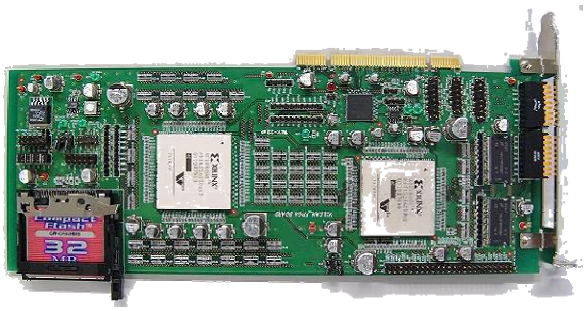

Figure 5: Vulcan emulation board

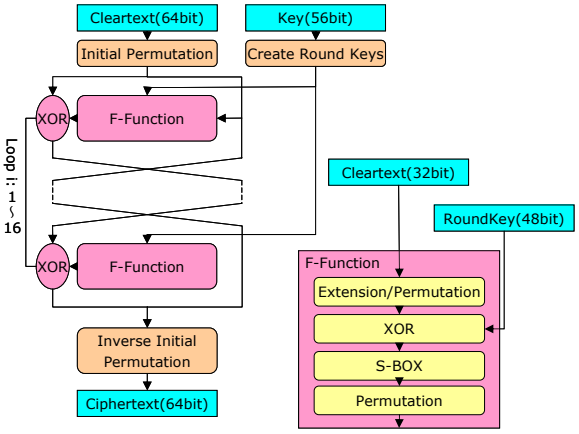

Figure 6: Original DES encryption algorithm

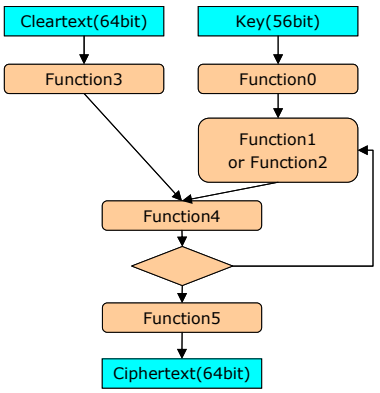

Figure 7: DES implementation on Vulcan

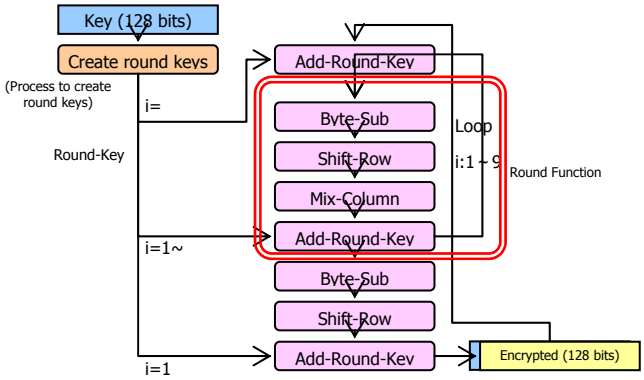

Figure 8: Execution flow of AES

4.3 Results

4.3.1 JPEG

In Table 2 we show the working frequency of the emulated Vulcan (implemented on the FPGA board) and the Pentium 4 processor (compiled by GCC [16] with –O2 optimization). For about 1/3800 of the working frequency, the performance was 260 times worse.

4.3.2 DES

The DES program was also compiled with *gcc –o2*. Although the working frequency of Vulcan is about **1/400** the frequency of the Pentium 4, we got a throughput **3.8 times** bigger. DES involves basically bit manipulations which are handled very efficiently by the RDP.

4.3.3 AES

The implementation of AES encryption algorithm on Vulcan required 309 instructions to encrypt the plaintext of one block (128 bits); the number of instructions in which the scheduling of the key was not included, was 79 instructions. There were a total of 38 different custom instructions. Table 3 shows the number of PE's used to process each custom instruction for the AES encryption algorithm. The first round was implemented using only 8 instructions. Especially, in the plaintext and round-key, the Exclusive-OR done at Add-Round-Key used all PE's for one custom instruction. Regarding the S-Box processing, as we need to process 128 bits and 124 PE's are necessary to process 4 bytes, we needed 4 different CI's. In the Shift-row processing (a shift in the row direction) we could implement it with one CI, however in the column direction (Mix-column) we needed two CI's.

In Table 4 one can see the number of executed instructions and the number of clock cycles needed to encrypt one block of plaintext in AES for Vulcan and Simplescalar's PISA. Vulcan needed **1/36** of the number of executed instructions. Assuming both work at the same frequency, AES encryption on Vulcan is about 5.3 times faster than the superscalar MIPS.

Table 2: Results for emulated DES and JPEG (800x640pixels)

Platform	Frequency		Throughput (Exec Time)	
	DES	*JPEG*	*DES*	*JPEG*
Vulcan	6.25MHz	625KHz	570KB/s	125.37s
Pentium 4	2.4GHz	2.4GHz	150KB/s	0.48s

Table 3: Number of PE's used in each CI for AES.

CI Executed	# of PE's used
Add-Round-Key	128
S-Box1	124
S-Box2	124
S-Box3	124
S-Box4	124
Shift-Row	64
Mix-Column1	104
Mix-Column2	104

Table 4: Performance numbers of AES (Encryption)

Processor	# Inst. Executed	Exec.Time (# of cycles)
Vulcan	309	1497
PISA(Superscalar)	11258	7902

5. SUMMARY AND DISCUSSIONS

In this section we discuss the result numbers presented in the previous section. Vulcan performed quite well for bitwise apps (DES and AES) but not as such for memory or compute intensive apps like JPEG. Fine-grain reconfiguration of the RDP (at bit-level) and the intrinsic correlation of PE's and IFRAM register's bits allows operations like bit-shuffling, masking, shifting and so one to be done just by specifying which PE's are going to compute the input data. These same operations are the most costly for GPPs. Although the use of RDP and IFRAM can be quite efficient when programmed "by hand", using ISAGen and the automatic P&R tool to generate a new ISA turned out to limit the actual size of CI's because of its strict communication constraints and the relatively scarce routing elements in the RDP.

Media applications like JPEG are very computing intensive, so the small 16-bit Adder and Multipliers in Vulcan are not sufficient to meet their timing constraints; also JPEG demands high memory bandwidth that was not our first concern when designing Vulcan at first (Vulcan actually needs to use the RDP to access the RF and memory). These bottlenecks are going to be addressed on the next version of the Redefis dynamically reconfigurable processor (under development).

Figure 9 and Figure 10 present the distribution of the most executed instructions (classified by their types) for Vulcan and Simplescalar, respectively.

Implementing AES in Simplescalar showed that over 40% of executed instructions are Load/Store ones. This is because the program used implemented the transformation S-Box and the procedure to generate the key by referencing a table in memory; also, the program stores the intermediate temporary results in memory. In Vulcan the intermediate results are kept in the internal registers, so as to the S-Box transformation, where the LUTs in the RDP were sufficient to store all data, we just needed 16 Loads and Stores to cipher one block of plaintext.

In Simplescalar the next most frequent instruction type was addition. This is due to the loops and indexing of lines and columns. In Vulcan as the Round function have been completely unfolded there is no increment in loops. The Exclusive-OR logic function in Add-Round-Key was used very much. In Vulcan it was possible to implement 128 bits Exclusive-OR in just one CI. Simplescalar processes the shift operation at byte level, differently from Vulcan which for one round, only 1 CI was needed for shift-row and 2 CI for Mix-column. In contraposition, *move* instructions in Vulcan counted over 40%. This is due to limitations on the number of inputs of the RDP (it is necessary to arrange the input in the same word when two or more inputs are necessary).

Vulcan surpassed Simplescalar when we consider the dynamic instruction count. However, for 1/36 of the number of instr. exec., the reduction in total number of cycles was just 1/5 compared to Simplescalar. This is because the simulated Simplescalar is a 4-way OOO processor, while Vulcan is an in-order one. Moreover, the design complexity, area, cache predictability problems and power characteristics associated with OOO processors usually restrict their usage as a solution in embedded or real-time systems.

In Figure 11 the distribution of execution time according to the internal processing of Vulcan, i.e., the phases or steps during the execution are presented. We observe that about 50% of the executed cycles were spent in IF (Instruction Fetch) phase. Cache missed a lot as we have completely unrolled the Round functions and consequently lost their temporal locality. Although the size of the cache was big enough, its hit rate was only 87.34%.

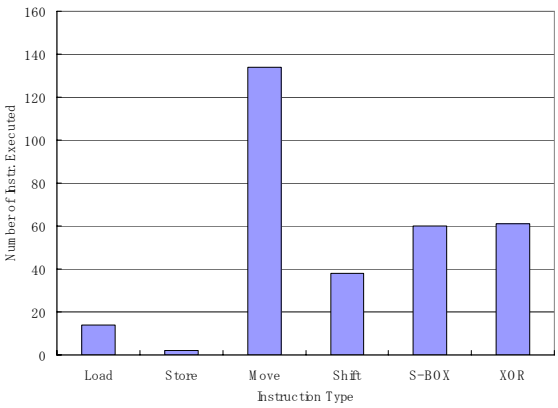

Figure 9: Distribution of Inst types executed in Vulcan for AES Encryption

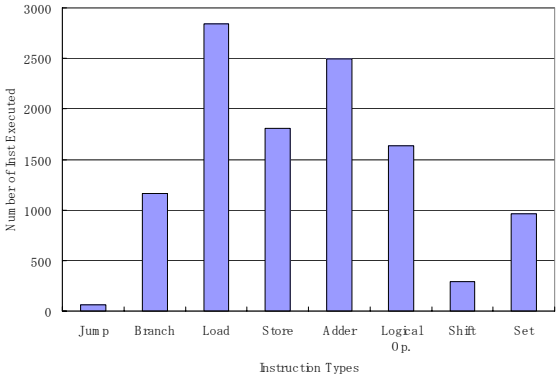

Figure 10: Distribution of Instr. types executed in Simplescalar for AES Encryption

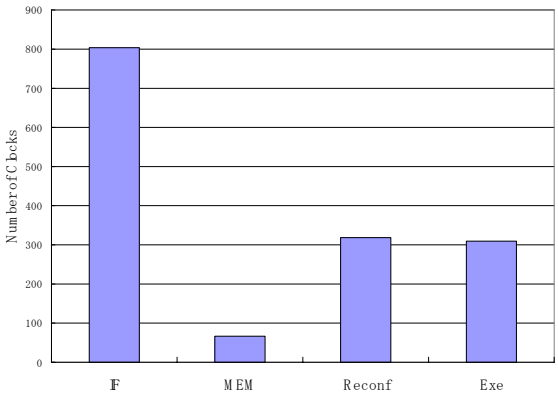

Figure 11: Distribution of Execution time (number of cycles) of AES Encryption on Vulcan

Lastly, as stated before, datapath reconfiguration occurs at every step. The reconfiguration overhead is much bigger than the time to execute that same instruction on the RDP. Some ways to solve this limitation is to pipeline the execution in the RDP or promote partial reconfiguration; or increasing the computation power of each PE (like [5] or [9]), however as the reconfigurable fabric becomes more coarse-grain we loose programmability and the benefits of low-level bit manipulation.

6. CONCLUSION AND FUTURE WORKS

In this paper, we have presented REDEFIS – a HW/SW design platform for high level, efficient implementation of ASIPs/engines for SoC systems. It is composed of a reconfigurable instruction set processor (Vulcan), capable to redefine its ISA according to the user application written in high level C language, and a set of design tools (an ISA Generator and a retargetable compiler).

We have mapped some encryption and media applications on Vulcan. Our preliminary results showed it performed very well for cryptography and other bitwise applications (compared to P4 the emulated system was 3.8X faster@1/400 the clock speed for DES). We continue to investigate other application niches and different topologies for the RDP, in order to meet the specifications to run JPEG, for instance.

We believe the Redefis solution provides a flexible and efficient framework (small footprint and power consumption) for easy design retrofit (triggered by new standards, on-the-field bug fixing), along with low design/production cost and short TAT suited to embedded systems and alike.

Other future works include the comparison of Redefis and an application-specific hardware designed from scratch (RTL), including design time, chip area, working speed and cost.

7. ACKNOWLEDGMENTS

This research was supported by Ministry of Education, Culture, Sports, Science and Technology (Japan), by its Knowledge Cluster Initiative (Fukuoka area).

8. REFERENCES

[1] T. Hashinaga et al. “Vulcan: the first implementation of Redefis processor and its design tool chain”, Technical Report of IEICE – Subject: Reconfigurable Systems, Dec. 2004 (In Japanese).

[2] M. Shuto et al. “Redefis: a SoC Design Platform”, Technical Report of IEICE – Subject: Reconfigurable Systems, Dec. 2004 (In Japanese).

[3] Altera Corporation. http://www.altera.com

[4] IPFlex Inc. http://www.ipflex.com

[5] PACT Corporation. http://www.pactcorp.com

[6] Tensilica Inc. Xtensa Configurable Processors. http://www.tensilica.com

[7] Stretch Inc. http://www.stretchinc.com

[8] MeP – Media Embedded Processor Architecture. http://www.mepcore.com/english/index_e.html

[9] Dynamically Reconfigurable Processor (DRP), NEC Electronics. http://www.necel.com/drp/en/index.html

[10] Elixent Ltd. http://www.elixent.com/

[11] Morpho Technologies Inc. http://www.morphotech.com

[12] D. Burger and T. M. Austin, “The Simplescalar tool set, version 2.0”, University of Wisconsin-Madison Computer Sciences Department Technical Report, 1997.

[13] NIST FIPS PUB 46-3, “Data Encryption Standard (DES)”,http://csrc.nits.org/publications/fips/fips46/fips46-3.pdf, Oct. 1999.

[14] NIST FIPS PUB 197, “Advanced Encryption Standard (AES)”, http://csrc.nist.gov/publications /fips/fips197/fips-197.pdf , Nov. 2001.

[15] Home page of the JPEG Committee.http://www.jpeg.org

[16] GNU Compiler Collection. http://gcc.gnu.org