

The ByoRISC configurable processor family

Nikolaos Kavvadias and Spiridon Nikolaidis
Section of Electronics and Computers, Department of Physics
Aristotle University of Thessaloniki
54124 Thessaloniki, Greece
Email: {nkavv,snikolaid}@physics.auth.gr

Abstract— Customizable/extensible processors can be tuned to exploit applications of interest in modern platform-based design. However, in most cases they are implemented as extensions of legacy architectures, a policy that poses significant limitations to achieving high performance improvements. In this paper, the paradigm of the contemporary ByoRISC processor family is presented. The ByoRISC specification provides an extensive set of architectural parameters that can be applied to a baseline processor, that can be extended by application-specific hardware extensions (ASHEs) in the form of either custom instruction units or locally-interfaced coprocessors. Such ASHEs can implement multi-input multi-output (MIMO) computations with local state that may have an arbitrary number and combination of load/store accesses to the data memory. The performance characteristics of the proposed processors have been evaluated for both FPGA and ASIC implementations, proving that ByoRISC is a viable solution in present-day SoC design. A case study of an image processing pipeline is also presented to highlight the process of utilizing a ByoRISC-based custom processor.

I. INTRODUCTION AND RELATED WORK

A recent approach to embedded SoC design involves the use of configurable and extensible processor cores [1]–[4], offering architecture customization possibilities. Configurability lies in tuning architectural parameters, while extensibility usually refers either to tightly-coupled modifications obtained by adding single-, multi-cycle or pipelined versions of ASHEs (custom instructions) or to loosely-coupled accelerators not directly integrated within the processor pipeline.

Customizable processors either adhere to the configurable/extensible processor paradigm [1], [4] or can be ASIPs (Application-Specific Instruction-set Processors) completely designed from scratch. An approach to configurability is feature subsetting by enabling/disabling features as well as parameter tuning (e.g. changing cache sizes) [5]; such methods can help to reduce the footprint of a processor core but do not take advantage of application characteristics that can lead to acceleration. Recent works [6] advocate in favor of both the custom instruction (CI) and coprocessor approaches, proving that both techniques can be considered simultaneously by formalizing the problem as a form of two-level partitioning.

During development, the designers ought consider the entire space of instruction sets and underlying microarchitectures. However, it is often that solutions are artificially limited to those empirically derived from past practices in order to seemingly reduce complexity without negatively affecting performance. A relevant example is the domination of the three-address instructions limitation which is closely associated to

a general-purpose register file with a small number of read and write ports, typically two and one, respectively. While a multi-port register file could provide significant performance boost, it is regarded as an unnecessary complexity that dramatically degrades the timing characteristics of the processor presumably without interesting design tradeoffs.

In this work, we present the novel ByoRISC (“Build your own RISC”) processor family that aims to serve as infrastructure for the development of application-tunable programmable processors. The rich architectural parameter space of a ByoRISC processor allows to specialize the processor over a narrow application set. Due to their architectural flexibility, conditions for significant acceleration can be exploited. The ByoRISC architecture has been successfully used as a testbed for design space exploration with the help of YARDstick [7], a design automation tool —retargetable to user-defined compiler intermediate representations —for application analysis and ASHE generation.

The rest of this paper is organized as follows. The ByoRISC architecture is presented in detail in Section 2. Section 3 discusses area and timing characterization of ASIC and FPGA implementations of ByoRISC processors. In Section 4, a ByoRISC-based system is used to accelerate an image processing application set. Finally, Section 5 summarizes the paper.

II. THE BYORISC ARCHITECTURE

A. Elements of the architecture template

A key issue for the success of a SoC design involving ASIPs is the ease of application development for the corresponding platform. For fully supporting HLLs, the ASIP has to provide a self-contained set of primitive operators. For example, the instruction set of the SABRE RISC processor [8] includes 28 integer instructions to fully support the HLL integer subset, while non-integer arithmetic can be emulated [9]. Other instances of significantly reduced instruction sets involve DLX/MIPS-I manifestations, and embedded soft processors such as MicroBlaze [3] and Nios-II [4].

The need for a fundamental RISC instruction set implies the development of an underlying architecture, common across processor variations in order to sustain code reuse, minimal application compatibility requirements and tool stability. Instructions are classified into the base, coprocessor and custom instruction subsets. The base instruction set is comprised of primitive instructions that ought be supported across all processor variants as well as derived instructions that can be

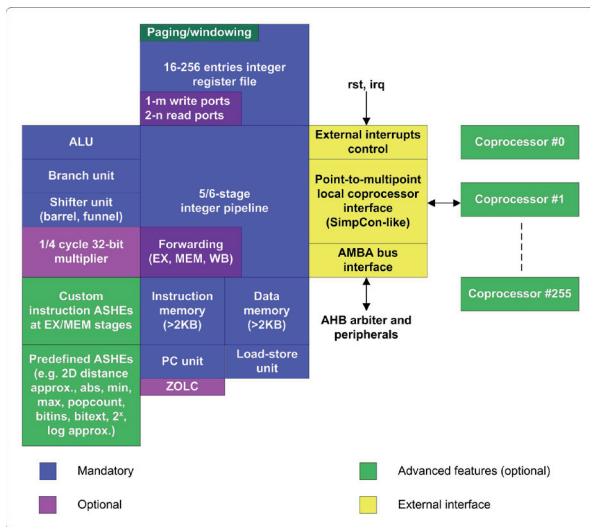


Fig. 1: Conceptual diagram of the ByoRISC architecture.

directly implemented in hardware, otherwise they should be emulated by embedded software.

B. Overview of the ByoRISC application-customizable processors

In order to fulfill the general requirements of section II-B for programmable application-specific computation, the ByoRISC processor architecture has been developed. The ByoRISC architecture encompasses the following characteristics that are common across all family member incarnations:

- 32-bit instruction and data word length; Harvard memory architecture.
- A base instruction set of 44 elements comprising of 22 primitive and 22 derived instructions.
- Up to 256 distinct primary opcodes, of which at least 192 are available to CI extensions.
- Configurable number of execution pipeline stages.
- Optional support for the ZOLC (Zero-Overhead Loop Controller) architecture [10] for the elimination of looping overheads within nested loop structures of arbitrary complexity.
- The size of the integer register file can be configured from a minimum of 16 to a maximum of 256 entries.
- Configurable number of read and write register file ports.
- Interface specifications for incorporating tightly-coupled (custom functional units) and local coprocessor application-specific hardware extensions (ASHEs).

A conceptual diagram of the ByoRISC architecture highlighting its constituent components is shown in Fig. 1.

C. Instruction formats

The ByoRISC instruction formats (Fig. 2) have been designed for maximum orthogonality in order to simplify the hardware of the instruction decoder. For this reason, the instruction fields for all formats start at an 8-bit (byte) boundary, with having only the type conversion instruction (cvt)

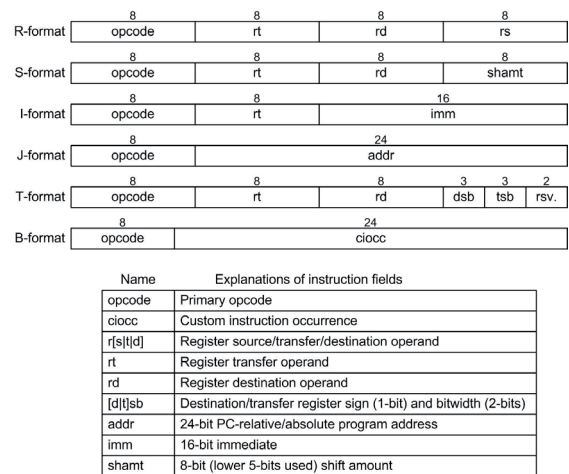


Fig. 2: ByoRISC instruction formats.

subdividing its secondary opcode to subfields for specifying sign and bitwidth of source/destination operands.

There are five distinct formats in the base instruction repertoire: R-format for instructions with two source and one destination register operand, S-format for shifts by an immediate constant, I-format for accessing 16-bit immediates, J-format for jump instructions and C-format for type conversion operations. Coprocessor instructions derived from the MIPS-I/32 specification follow the S-format while CIs are encoded in the B-format. The *ciocc* field denotes a specific occurrence of a CI usage in ASIP-targeted applications.

D. The ByoRISC instruction set

The ByoRISC instruction set shares characteristics to typical load-store machines such DLX, MIPS-I/32 [11], and SABRE [8]. The requirements of orthogonality in instruction encoding, and direct access to a large opcode space and programmer visible register set, limit the size of immediate operands for arithmetic instructions to 8 bits. Only the LLI, LHI and LOLI instructions allow the encoding of halfword-sized immediates (16-bits). Table I summarizes the instruction set.

The instruction set is subdivided into instruction groups for arithmetic (A), load/store (LS), multiply (M), division (D), logical (L), set/comparison (C), immediate constant load (I), type conversion (T), control-transfer (F), procedure call (P) and coprocessor access (CP). The custom instruction group is denoted as CI.

A minimal ByoRISC has to support the following 22 instructions directly in hardware: add, addu, sub, subu, and, or, xor, lw, sw, lli, lhi, loli, srav, srlv, sllv, slt, sltu, j, jr, bnez, beqz, halt.

E. Custom instruction support in ByoRISC processors

1) *Decoding of custom instructions*: For decoding CIs, the concept of Secondary Instruction Decoding (SID) has been introduced. SID operation takes place in a partial decoding stage preceding the actual ID stage for base instructions, where CIs are identified based on their opcode MSBs. In the

TABLE I: The ByoRISC predefined instruction set.

Mnemonic	Type	Description
<i>LLI, LHI, LOLI</i>	I	Halfword load (lower/upper/lower with OR)
<i>LW, LB, LBU, LH, LHU, SW, SB, SH</i>	LS	Load/store signed/unsigned words, bytes, and halfwords to/from a register
<i>ADD, ADDU, SUB, SUBU</i>	A	Arithmetic operation on two registers (rs, rt) and result write-back to rd
<i>CVT</i>	T	Convert type and transfer register
<i>MUL, MULU</i>	M	Multiply
<i>DIV, DIVU</i>	D	Divide
<i>AND, OR, XOR, NOR</i>	L	Logical operation on two registers (rs, rt) and write result to register rd
<i>SRAV, SRLV, SLLV</i>	S	Shift by a register operand
<i>SRA, SRL, SLL</i>	S	Shift by an immediate value
<i>SEQ, SNE, SLT, SLTU, SLE, SLEU</i>	C	Compare two registers (rs, rt) and set register rd on condition
<i>J, JR</i>	F	Direct/indirect unconditional jump
<i>BNEZ, BEQZ</i>	F	Conditional branch
<i>JAL</i>	P	Jump and link to address (procedure call)
<i>[C M][F T]CX, [L S]WCX</i>	CP	Coprocessor interface instructions
<i>SYSICAL, BREAK, HALT</i>	-	Software interrupt instructions

$\log_2(NR)$...	$\log_2(NR)$	n_o	$\log_2(NR)$...	$\log_2(NR)$	n_i
dst₀		dst_{no-1}	we_v	src₀		src_{ni-1}	re_v

Fig. 3: The SID lookup table entry format.

SID stage resides a LUT where the input and output register operand addresses for specific CI occurrences in user programs are kept. The LUT is addressed by the *ciocc* field of B-format instructions. An entry in the SID LUT is partitioned as shown in Fig. 3, where:

- NR is the number of registers in the integer register file
- n_i, n_o is the maximum allowable number of input and output operands of a CI
- $dst_0 \dots dst_{n_o-1}$ and $src_0 \dots src_{n_i-1}$ is the register address for output and input operands, respectively
- we_v and re_v is the write/read enable vector for output/input operands, correspondingly

For a requirement of n_i input and n_o output register file ports, SID LUT entries have a width of: $(n_i+n_o) \cdot (\log_2(NR)+1)$ which is simplified to: $2 \cdot n \cdot (\log_2(NR) + 1)$ given that $n_i = n_o = n$. For $n = 8$ and $NR = 256$, which is our typical case of a ByoRISC testbed architecture, each entry has a width of 144 bits. This implies allocating only 2 block RAMs in the Spartan-3 FPGA technology process (18k storage bits) for realizing a 256-entry 144-bit wide LUT, in a special, single read port, block RAM configuration mode.

2) *Accessing operands of MIMO custom instructions:* The operand interface between the ByoRISC register resources and the datapath has to provide sufficient bandwidth in order not to compromise the performance benefits of tightly-coupled ASHEs. In this context, recent approaches [12], [13] involve the utilization of a multi-port register file for zero-cycle overhead access to registers.

A possible register file topology regards a monolithic register file using a single register bank with appropriate

multiplexer (e.g. permutation) networks for direct read and write access to all registers. While feasible for ASIC processes, the deep multiplexer staging is problematic in FPGAs. Another approach uses n blocks of memory (assuming for the number of write/read ports: $m < n$) with one read and one write port, having the side-effect that simultaneous access to any register is not sustained. A third solution is the use of $m \times n$ block RAMs for maintaining the maximum number of multiple copies [13]. For this topology, the zero overhead access to register operands is ensured. However, the demand in block storage resources may prove overwhelming especially for small FPGA devices, while each memory bank is used for storing only NR/m registers. The ByoRISC multi-port register file follows the latter approach.

3) *Computational states of ByoRISC custom instructions:* In ByoRISC processors, tightly-coupled ASHEs can have direct access to the data memory. This capability allows for incorporating an arbitrary number and combination of load and store operations within CIs, a feature not available in any contemporary soft-core processor. The ASHE-data memory interface is implemented as part of the MEM pipeline stage (also seen as the last of the series of EX stages) and can be used in consecutive cycles by the same CI. The operational states of CIs are summarized in Table II.

TABLE II: Operational states for ByoRISC custom instructions.

State	Stage	Description
<i>COMPUTATION</i>	EX, MEM	Computations not accessing the data memory
<i>LOAD</i>	MEM	Load from the data memory
<i>STORE</i>	MEM	Store to the data memory
<i>SPECIAL_CS</i>	MEM	Execution of chained operations concluding to a single store

F. Local coprocessor support in ByoRISC processors

Local coprocessors in embedded processors can induce significant architectural advantages in the design of embedded SoCs such as incorporating coarse-grain enhancements that are visible to the programmer and reuse of existing optimized IPs. It is common for embedded processor families to specify a local interface for integrating one or more coprocessors to the system. For example, the MIPS32 specification defines 4 coprocessors, while the ARM [14], LEON3 [2], and COFFEE [15] processors define their own coprocessor interfaces. The ByoRISC architecture supports up to 256 coprocessors in principle, that are interfaced to the processor core via a point-to-multipoint interconnect, based on the SimpCon SoC interconnect [16]. SimpCon presents desirable attributes for overlapping execution among coprocessors and the main core: single-cycle master handover, acceptance of pipelined requests and postponing the readout of slave results.

G. The microarchitecture of ByoRISC processors

1) *Pipeline organization:* A microarchitecture for ByoRISC has been fully implemented with a configurable 5/6 stage pipeline, and an address space of a minimum of 11 to a

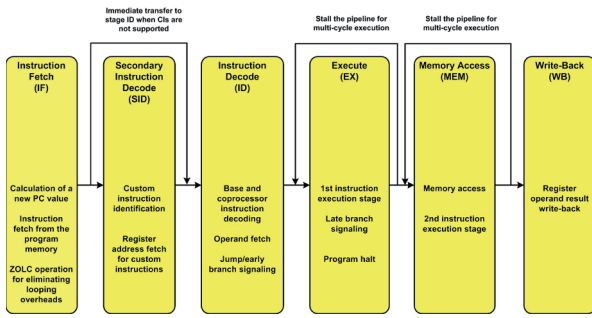


Fig. 4: Pipeline organization for ByoRISC.

maximum of 32 bits. The ByoRISC pipeline organization is shown in Fig. 4.

The pipeline stage organization of ByoRISC is based on the classic 5-stage pipeline design encountered in popular embedded RISCs. The main differences to these processors are the addition of the SID stage, and the fact that the execution stages (EX, MEM) can perform general multi-cycle operations. The storage resources (program and data memory, register file, SID LUT) are mapped to FPGA block RAMs.

At the IF stage, instruction fetch from the program memory can be optionally controlled by the ZOLC. Decoding of CI register operands takes place in the SID stage as described in section II-E.1. The decoding of base and coprocessor instructions and the operand fetch for all instructions take place at the ID stage. Stage EX is the first execution stage accessible to CIs and is also used for datapath computations of base instructions. As operands, either the values fetched during ID or forwarded ones can be used. The primitive base instructions are serviced by the ALU, the variable shifter and the branch unit. The basic addressing mode for these instructions is register direct, while additional modes can be introduced as user-defined extensions. An optional multiplier and a radix-2 divider are added when the corresponding derived instructions should be available in hardware. The EX stage also communicates to the local coprocessors for transferring the necessary data. At the MEM stage, load and store base instructions access the data memory. In addition, CIs may interface to the data memory when operating in LOAD, STORE or SPECIAL_CS computational states. The final pipeline stage, WB, is responsible for committing destination register operands to the centralized register file as those are calculated by base, coprocessor and CIs.

2) *The configuration space of ByoRISC processors:* A prominent characteristic of the ByoRISC architecture is the multi-parametric space, enriched by more than 20 parameters, that is used for user-defined configuration of the microarchitecture description prior logic synthesis. The parameter set is given in Table III.

3) *Integration of the ZOLC architecture:* ZOLC [10] which eliminates looping overheads in arbitrary loop structures with multiple-entry and multiple-exit nodes is integrated in the instruction fetch stage of embedded RISC processors. The

TABLE III: The configuration space of the ByoRISC microarchitecture.

Microarch. parameter	Description
<i>HAVE_CI</i>	Custom instruction support
<i>HAVE_COP</i>	Local coprocessor support
<i>HAVE_ZOLC</i>	ZOLC control architecture
<i>SMALL_IMM</i>	Small (8-bit) immediate operands
<i>FORWARDING</i>	SRB data forwarding
<i>BR_EARLY</i>	Choice of signaling a control flow transfer prior or after the EX/MEM pipeline register
<i>[I/D]MEMSIZE</i>	Program/data memory size
<i>OW, RAW</i>	Opcode/register address width
<i>NWP, NRP</i>	Number of register file write/read ports
<i>OPT_LS</i>	Load/store instructions for small data types (lb, lbu, lh, lhu, sb, sh)
<i>OPT_SHIFT</i>	Shift by immediate instructions (sra, srl, sra)
<i>OPT_CTI</i>	Optional control-transfer instructions (jal)
<i>OPT_CVT</i>	Type conversion (cvt)
<i>OPT_MUL</i>	Multiplication (mul, mulu)
<i>OPT_DIV</i>	Division (div, divu)
<i>OPT_SET</i>	Optional comparison instructions (seq, sne, sle, sleu)
<i>OPT_LOGIC</i>	Optional logical instructions (nor)
<i>MULT_TPL</i>	Multiplier topology (single-cycle or 4-cycle latency pipelined)
<i>SHIFTER_TPL</i>	Shifter topology (funnel, barrel, dedicated shifters)

notion of control transfer expressions (*CTEs*) among portions of the code situated at loop boundaries is used. The instruction lists that comprise each of these code segments are *Data-Processing Tasks (DPTs)* at the control-flow graph (*CFG*) level. The DPTs are distinguished between backward tasks that are involved in task switching decisions and update the loop index context of the CFG and forward tasks that may be involved in task switching but do not affect loop indices.

For ZOLC operation, the looping instruction pattern (loop index update, comparison to boundary values, and branch to the entry PC of the succeeding DPT) is eliminated. Instead, the necessary task switching takes place during the instruction fetch of the last useful instruction of the specific DPT. The purpose of ZOLC is to provide a proper candidate program counter (PC) target address to the PC decoding unit for each substituted looping operation.

4) *Scalable register bypassing scheme:* An abstract view of the ByoRISC register bypassing scheme, configurable via register-transfer level parameters, is shown in Fig. 5. The first execution stage receives up to *NRP* read register operands from a multi-ported register file and produces a result vector of up to *NWP* write register operands. The subsequent execution stages accept the result vector from their preceding stage, which is of $NWP \times DW$ bitwidth, where *DW* is the register word width. Further, it can be specified that they read up to *NRP* from the forwarded read operands, given that these have been stored in the pipeline registers of the previous stage. The final pipeline stage is responsible for committing the final result vector to the register file. Any of the *NPIPE* execution stages can be configured for multi-cycle execution, stalling the previous ones for the required number of cycles.

The SRB hardware mainly comprises of the following components:

- $NRP (NPIPE \times NWP + 1)$ -to-1 multiplexers in the

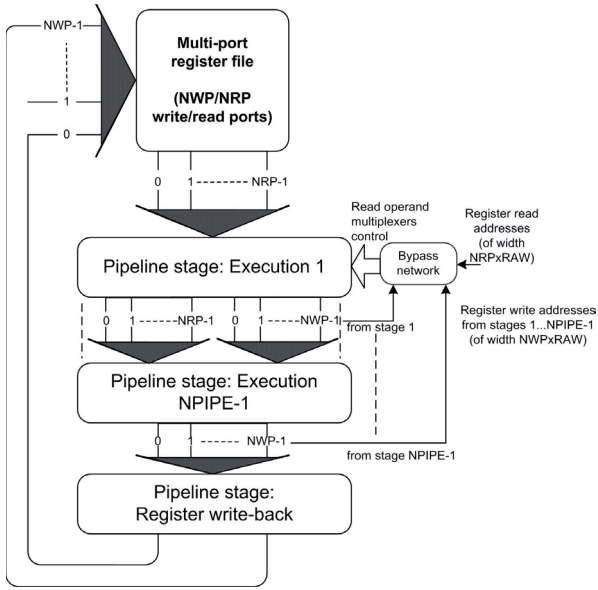


Fig. 5: Model of the ByoRISC scalable register bypassing (SRB) scheme.

first execution stage of the processor for selecting the proper forwarded datum per read port.

- $NRP \times NPIPE \times NWP$ comparators for evaluating the multiplexer control signals.

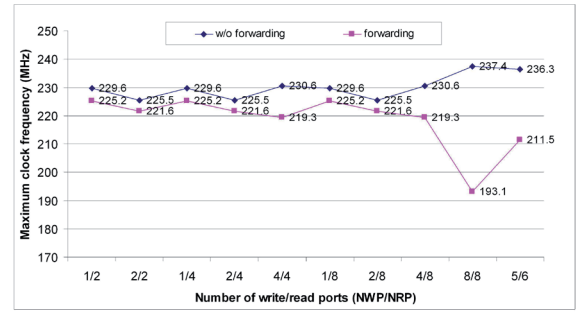
III. SPEED AND AREA CHARACTERIZATION OF A REPRESENTATIVE BYORISC PROCESSOR

In order to provide performance characterization results regarding the timing (critical path) and area of the VHDL description of a typical ByoRISC processor, we assume the following configuration set:

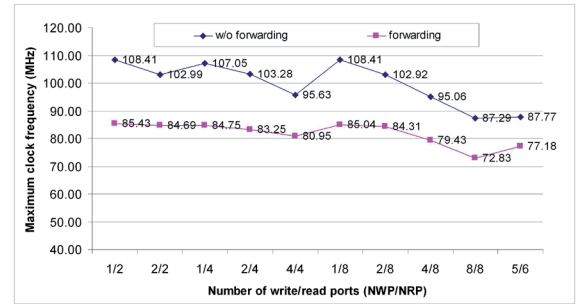
- HAVE_CI, HAVE_COP, OPT_LS, OPT_MUL, OPT_SHIFT, OPT_CTI, and OPT_LOGIC equal to one, BR_EARLY, OPT_CVT, OPT_DIV, OPT_SET, equal to zero.
- IMEMSIZE, DMEMSIZE = 8KB.
- OW, RAW = 8.
- SHIFTER_TPL = "FUNNEL", MULT_TPL = "MULT32X32-PIPELINED" selecting a topology with 4 internal pipeline stages.

For each case, the timing and area requirements are estimated with the help of the Mentor LeonardoSpectrum (ASIC) and Xilinx Webpack ISE 7.1.04i (FPGA) synthesis tools. For the Virtex-4 Xilinx FPGAs, a chip area scalar metric is extracted (ranging from 0 for zero area occupation to 100 for the entire chip resources), by means of a linear weighted cost function of the number of LUTs, 16-kbit block RAMs, and DSP48 embedded datapaths used. For the XC4VLX25 device (FF668 package, '-10' speed grade) that is used throughout the evaluations and is one of the smallest available Virtex-4 devices, we have chosen the corresponding weight coefficients: 0.5, 0.375, and 0.125.

Fig. 6 depicts the maximum clock frequency estimates for different number of supported read (NRP) and write (NWP)



(a) Standard cell VLSI STM 0.13 μ m process.



(b) Virtex-4 device: XC4VLX25

Fig. 6: Maximum clock frequency for the ByoRISC processor.

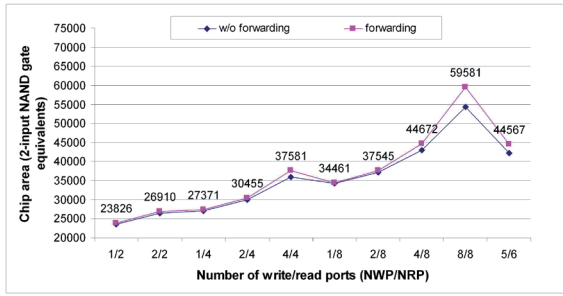
ports. The chip area requirements are shown for both processes in Fig. 7. The number of execution pipeline stages has been set to 2.

From these figures, it can be seen that the number of write/read ports escalates the chip area on the FPGA device to about an order of magnitude, from 6% to about 50% of the total resources. For the ASIC process, without accounting for the register file area, the corresponding value range is about three times compared to the baseline case figures (18k to 60k gates). In addition, the use of a full data forwarding network decreases the maximum clock frequency by 17.9%. On the contrary, for the ASIC process, this performance degradation measures to only 5%. The difference in maximum clock frequency among the cases of the minimum (1/2) and maximum (8/8) register file ports examined, with and without the use of full data forwarding, measure to 19% and 9.7%, respectively for the FPGA.

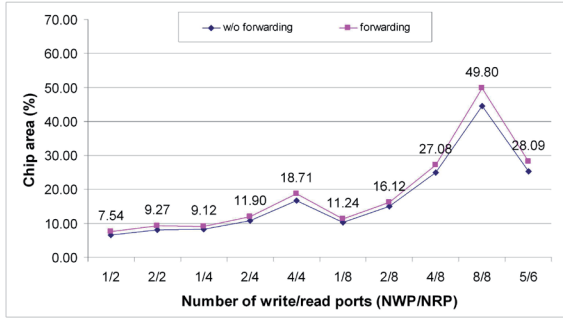
IV. CASE STUDY: AN IMAGE PROCESSING PIPELINE

In order to evaluate the performance of the ByoRISC architecture on realistic applications, an image processing pipeline (IPP) has been used. The IPP which is shown in Fig. 8 processing 256-level greyscale images, comprises of three application kernels: *fsdither* (Floyd-Steinberg dithering by error diffusion to a bilevel image), *htpack* (halftone image packer for 8-fold lossless compression of a bilevel image), and *xteaenc* (XTEA encryption [17]).

We performed application analysis and CI generation and selection with the YARDstick toolset [7]. First, the critical basic blocks of the applications have been identified



(a) Standard cell VLSI STM 0.13µm process.



(b) Virtex-4 device: XC4VLX25

Fig. 7: Chip area for the ByoRISC processor.

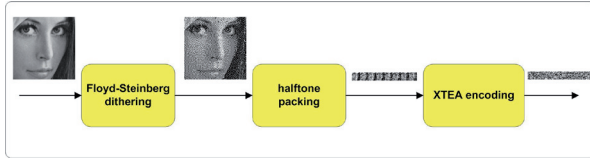


Fig. 8: An image processing flow comprising of dithering, data packing and XTEA encryption.

(Table IV). CI generation has been applied for identifying MIMO subgraphs, by evaluating a search graph representation for basic blocks. In the process of CI generation, control-transfer instructions have been excluded. A greedy selector for the ‘cycle gain’ priority metric has been used.

TABLE IV: ‘Hot’ basic blocks for the IPP applications.

Application	Instructions	Estimated dynamic cycles	% of application
fsdither.5	59	237568	85.19
fsdither.2	10	40960	14.69
htpack.2	80	40448	99.98
xteaenc.2	44	638976	95.63

A summary of the identified CIs is given in Table V. It can be seen that $N_i/N_o = 8/8$ write/read ports suffice for a 99.5% coverage of the maximum theoretical application acceleration assuming unlimited number of register file ports. However, when comparing the 8/8 and 8/4 cases it is seen that this difference is not negligible (about 17.6%). A speed improvement of $5.2\times$ is expected over the application set based on the estimations (averaging the results of column ‘Incrim. speedup’ over the three applications of IPP) made

by YARDstick.

TABLE V: CI characteristics. N_c refers to the number of constants in a CI. ‘MAU’ refers to the area occupation of a 32×32 -bit multiplier producing a 64-bit result.

CI	$N_i/N_o/N_c$	Cyc. gain	Incrim. speedup	SW cyc.	HW cyc.	Area (MAU)
fsdither0	5/2/9	180224	2.69	57	13	4.56
fsdither1	3/2/2	32768	3.87	9	1	0.18
htpack0	3/2/14	35328	7.25	78	9	1.10
xteaenc0	6/5/7	540672	4.55	38	5	0.83

V. CONCLUSIONS

In this paper, the configurable ByoRISC processor architecture for ASIP development has been presented. ByoRISC processor designs are well suited to design space exploration due to their scalability; such an example being the register file and data forwarding architecture. Further, ByoRISC processors allow the investigation of possibilities for ASHE integration. Hardware characterization of a reference ByoRISC model proves that this approach is feasible even on moderately sized FPGAs. A case study application set was explored and implemented on ByoRISC unveiling a potential acceleration of $5.2\times$ compared to the baseline processor.

REFERENCES

- [1] R. Gonzalez, “Xtensa: A configurable and extensible processor,” *IEEE Micro*, vol. 20, no. 2, pp. 60–70, Mar.-Apr. 2000.
- [2] Gaisler research. [Online]. Available: <http://www.gaisler.com>
- [3] Xilinx home page. [Online]. Available: <http://www.xilinx.com>
- [4] Altera Nios II home page. [Online]. Available: <http://www.altera.com/products/ip/processors/nios2/>
- [5] P. Yiannacouras, J. G. Steffan, and J. Rose, “Application-specific customization of soft processor microarchitecture,” in *Proc. 14th Int. Symp. on Field Programmable Gate Arrays*, Monterey, California, USA, Feb. 2006, pp. 201–210.
- [6] S. Sirowy, Y. Wu, S. Lonardi, and F. Vahid, “Two-level microprocessor accelerator partitioning,” in *Proc. Design, Automation and Test in Europe Conf.*, Nice, France, April 2007, pp. 313–318.
- [7] N. Kavvadias and S. Nikolaidis, “YARDstick: Automation tool for custom processor development,” in *presented at the University Booth of the Design, Automation and Test in Europe Conf.*, Nice, France, Apr. 2007.
- [8] SABRE RISC processor example. [Online]. Available: <http://www.celoxica.com>
- [9] SoftFloat. [Online]. Available: <http://www.jhauser.us/arithmetic/SoftFloat.html>
- [10] N. Kavvadias and S. Nikolaidis, “Elimination of overhead operations in complex loop structures for embedded microprocessors,” *IEEE Trans. on Computers*, vol. 57, no. 2, pp. 200–214, Feb. 2008.
- [11] MIPS technologies Inc. [Online]. Available: <http://www.mips.com>
- [12] A. K. Jones, R. Hoare, D. Kusic, G. Mehta, J. Fazekas, and J. Foster, “Reducing power while increasing performance with supercisc,” *ACM Trans. on Embedded Computing Systems*, vol. 5, no. 3, pp. 658–686, Aug. 2006.
- [13] M. A. R. Saghir and R. Naous, “A configurable multi-ported register file architecture for soft core processors,” in *Proc. 2007 Int. Workshop on Applied Reconfigurable Computing*, Mangaratiba, Rio de Janeiro, Brazil, Mar. 2007, pp. 14–25.
- [14] ARM Ltd. [Online]. Available: <http://www.arm.com>
- [15] The COFFEE RISC Core homepage. [Online]. Available: <http://www.cs.tut.fi/~coffee/>
- [16] SimpCon homepage. [Online]. Available: <http://www.opencores.org/projects.cgi/web/simpcon/overview>
- [17] R. M. Needham and D. J. Wheeler, “TEA extensions,” Computer Laboratory, University of Cambridge,” Technical Report, Oct. 1997.