Scalable register bypassing for FPGA-based processors ⋆

Nikolaos Kavvadias *, Spiridon Nikolaidis

Section of Electronics and Computers, Department of Physics
Aristotle University of Thessaloniki
54124 Thessaloniki, Greece

Abstract

In this paper, a scalable scheme, configurable via register-transfer level parameters, for full register bypassing in a modern embedded processor architecture, termed ByoRISC, is presented. The register bypassing specification is parameterized regarding the number of homogeneous register file read and write ports and the number of pipeline stages of the processor. The performance characteristics (cycle time, chip area) of the proposed technique have been evaluated for FPGA target implementations of the synthesizable ByoRISC model. It is proved that, a full bypassing network is a viable solution for the elimination of data hazards when servicing instructions with multiple read and write operands. While the maximum clock frequency is reduced by 17.9% in average, when using partial versus full forwarding, the positive effect of custom computation eliminates this effect by providing cycle speedups of 3.9× to 5.5× and corresponding execution time speedups for a ByoRISC testbed processor of 3.6×. Individual application speedups of up to 9.4× have also been obtained.

Key words: Microprocessors, Register bypassing, Field-programmable gate arrays, Embedded systems, Hardware description languages
PACS: 89.20.Ff

⋆ This work was supported by the General Secretariat of Research and Technology of Greece and the European Union.
* Corresponding author.
Email addresses: nkavv@physics.auth.gr (Nikolaos Kavvadias), snikolaid@physics.auth.gr (Spiridon Nikolaidis).

1 Introduction

A recent approach to embedded System-on-Chip design involves the use of configurable and extensible processors [1–5], usually in the form of synthesizable cores, offering architecture customization possibilities. Configurability lies in tuning architectural parameters, while extensibility of a processor usually refers either to tightly-coupled modifications obtained by adding single-, multi-cycle or pipelined versions of custom instructions or to loosely-coupled accelerators not directly integrated within the processor pipeline.

These architectural frameworks are regularly updated with enhancements targeting at the improvement of diverse and often conflicting requirements such as low power consumption, performance for the general-purpose or specific application domains, code size and overall system cost. During the development of such processors and especially regarding non-legacy architectures, the designers ought to consider the entire space of architectural solutions regarding the instruction set and underlying microarchitecture. However, it is often that designers limit themselves to solutions that are empirically derived from past practices in order to seemingly reduce complexity without negatively affecting performance. An interesting 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 unnecessary complexity, dramatically degrading the timing characteristics of the processor, especially if its implementation is combined with data forwarding mechanisms across several pipeline stages.

In this work, we focus on the design and evaluation of data forwarding (register bypassing) architectures, which is a technique for eliminating data hazards in pipelined processors. The function of the bypassing hardware is to resolve data hazards that arise when an instruction needs the results of previous instructions in the pipeline that have not been written to the register file by the time the current instruction reads its source operands from the register file. Generally, it is expected that extensive bypassing comes with a significant impact on cycle time, area and power consumption of the processor.

In this paper, a scalable and parameterized register bypassing scheme is presented that can be utilized in current embedded processors. The specification of the bypassing architecture can be configured for the desired number of register file read/write ports and pipeline stages of the processor in mind. The main contributions of this paper can be summarized as follows:

- development of a clear and concise register bypassing specification that is fully parameterized and can be easily applied to different processors. Espe-

cially, it can be of particular interest to developers of new/emerging processor architectures for providing more architectural options to their end users.
- the effect of the bypass circuitry on the timing and area of a representative processor are carefully examined. Most previous works only model either a partial processor or solely the bypassing mechanisms.
- specific issues regarding targeting the bypass specification to recent FPGA devices are highlighted. FPGAs have been neglected as an implementation medium even in recent works on the subject.

The rest of this paper is organized as follows. Related work is summarized in Section 2. The processor pipeline model is briefed in Section 3. Section 4 discusses the details of the scalable register bypassing (SRB) specification, and in Section 5 its performance is evaluated in terms of timing characteristics and area requirements as well as in context of an image processing application set running on an embedded RISC processor. Finally, Section 6 summarizes the paper.

2 Related work

In related work, a number of approaches have been proposed for the evaluation of register bypassing networks [6–10]. Most of them deal with exploring the design space of partial bypassing for an application set, representative of a particular domain, in order to drive the customization and reduction of a full bypass network. In the aforementioned works, neither a concise formalism nor a reusable model of bypassing, applicable to FPGAs, is presented that can provide sufficient assistance to the processor designer. Further, it is common that the bypass network is evaluated for timing and area characteristics apart from the processor, while a processor model taking account only the cycle behavior of using the bypassing mechanisms is used separately for obtaining execution cycles measurements.

In an early work in this field, Abnous et al. [6] analyzed partial bypassing between VLIW functional units in their 4-integer-unit VIPER processor. They argued that complete bypassing is too costly in VLIW processors even though significant performance benefits can be achieved. The pipeline model of VIPER is rather inflexible and cannot be used for exploration purposes: it is restricted to four stages, deduced from the classic 5-stage pipeline of early RISCs by removing the memory access stage. In order to achieve this, the processor model is limited to a single addressing mode (register indirect).

Further, in [7] the architecture of a detailed bypassing execution unit model is described and applied for a multiple instruction issue processor. Similar

to [6] the processor model features a four-stage pipeline, but in this case with configurable multiplicity of execution datapaths. A design space exploration approach for eliminating infrequently used routes in register bypass networks has been presented in [8] applied to the case of a 5-issue custom VLIW processor. In a similar architectural context, low-power optimizations that exploit the forwarding paths of a fixed register bypass network, for the purpose of minimizing power-costly accesses to/from the register file have been also examined [10].

In [9] an operation table formalism was developed for capturing the bypass mechanisms in pipelined embedded processors, along with an automation tool (PBExplore) for exploring the design space, constituted of the partial bypassing solutions, in terms of achievable performance. The authors assume that full register bypassing is not a viable solution, thus partial bypassing is preferred. However, the integration of the bypass networks within a synthesizable description of their testbed architecture (Intel XScale) is not considered at all, even though this would be necessary in order to evaluate the effect of the bypassing network on the processor cycle time and aggregate area. Further, in their work, processor implementations on FPGAs have not been considered at all.

A recent technique [11] on the design of register bypasses involves a compiler-driven approach based on the fact that certain register addresses are not actually read, given that the corresponding operands are forwarded to the appropriate ports by the bypass network. In this case, the processor instructions have to be statically rewritten to free the corresponding fields in order to derive the appropriate control signals. Their architectural model is closer to our approach, incorporating a multi-port register file and a configurable number of pipeline stages. However, the main aim of this technique is the energy consumption and area reduction of the redundant bypasses for a VLIW ASIC processor model, and not the thorough investigation of the practicality of full register bypassing on FPGA-based soft processors.

An extensive design space exploration of clustered VLIW architectures, typically employing 2, 4 or 8 partitioned register files can be found in [12]. The complexity of a full bypass network is reduced due to the smaller number of read and write ports of the partitioned register files, with the tradeoff of introducing copy operations among these. For the case of the unicluster architecture which is also investigated as a reference, it is stated that its performance is significantly lower to the clustered architecture. While this is true, the exploration targets a standard cell VLSI process, and performance on modern FPGA devices is not discussed.

A common denominator of some of these works [8, 9] is the consideration of compiler visibility of the partial bypasses. Here, although partial bypassed

networks are possible, we focus on presenting a scalable specification for full register bypassing hardware that is fully transparent to the programmer.

3 An abstracted view of the ByoRISC processor model

The architectural model targeted in this paper, shown in Fig. 1, is the ByoRISC processor [13]. A ByoRISC processor 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 ByoRISC template employs a pipeline stage structure consisting of:

- an instruction fetch stage, IF (not shown in Fig. 1) of a possibly wide instruction, incorporating one or more micro-operations to be executed in their corresponding execution slots
- a custom instruction operand address access stage and an instruction decode stage where NRP register operands are read
- $NPIPE$ execution stages with at least one of them accessing the data memory for full support of ByoRISC ASHEs
- a register write-back stage for writing NWP register operands

The basic assumption for the first execution stage is that it 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. There is no limitation on the policy followed in the architecture for the incorporated functional units: the processor can present a VLIW/EPIC architecture, servicing a number of independent micro-operations in the same control step, or it can evaluate MIMO (Multiple-Input Multiple-Output) instructions [14] that are represented by data-dependence directed acyclic graphs of basic block scope at the level of compiler intermediate language. The subsequent execution stages accept the result vector from their preceding stage, which is of width $NWP \times DW$, 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 corresponding pipeline registers. The final pipeline stage is responsible for committing the final result vector to the register file. Additional computations do not take place at this stage, so reading the read register operand vector and the corresponding register addresses (assumed through the figure) is not necessary. Any of the $NPIPE$ execution stages can be configured for multi-cycle execution, stalling the previous ones for the required number of cycles.

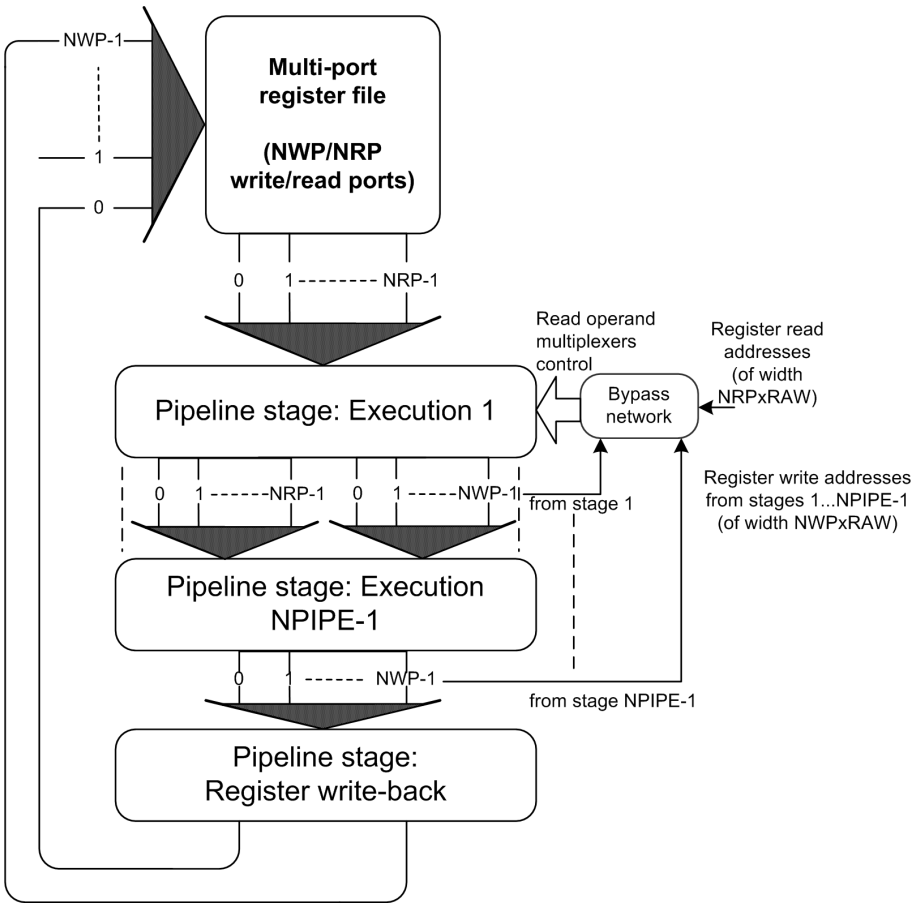


Fig. 1. Microarchitecture model for the ByoRISC contemporary soft-core processors, focusing on data forwarding of intermediate results.

The bypass network produces the multiplexer control signals that are used within the first execution stage for forwarding the appropriate data value; this would be either the one retrieved from the register file, or an intermediate result from one of the pipeline registers, to be written at a later stage to the same physical address. The first execution stage incorporates a set of multiplexers for selecting one of the forwarded values per register file read port.

This model encompasses the vast majority of the existing soft-core processors. It even includes provisions for future embedded soft-core processor architectures that will be able to issue, process and commit several operands in either the same cycle or the same control step.

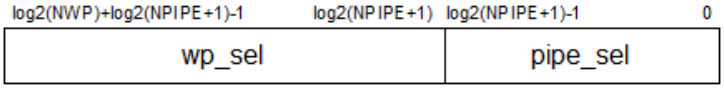


Fig. 2. Execution 1 stage multiplexer control signals (alu_opd_v_sel in pseudocode).

4 RT-level specification of the scalable register bypassing mechanisms

In order to assist the application of the Scalable Register Bypassing mechanisms we present here pseudocode semantics for their description. These semantics have inspired a synthesizable RTL description that has been successfully used for the ByoRISC in-house embedded processor, whose baseline version shares strong similarities with the DLX, MIPS-I [15] (regarding the instruction set) and MMIX [16] (orthogonality of the instruction encodings) processors. More specifically, the baseline ByoRISC utilizes a single issue 6-stage pipeline incorporating an integer register file with $NRP = 2$ and $NWP = 1$. A formal representation of SRB micro-architectural level operations can be exploited in the scope of designing contemporary embedded soft-core architectures or delivering architectural variations within a given processor family.

From a high-level point of view, the SRB architecture is fully parameterized regarding the number of read (NRP) and write (NWP) ports of the programmer-visible register set, and the number of execution pipeline stages ($NPIPE$). In the case of the in-house processor (ByoRISC), it is $NPIPE = 2$ and the two respective pipeline stages are EX (Datapath Execution) and MEM (Memory Access). The SRB hardware mainly comprises of the following components:

- NRP $(NPIPE \times NWP + 1)$-to-1 multiplexers in the 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. In case of supporting multi-cycle execution, the result of each comparator is AND-gated with a flag stating the completion of multi-cycle operation for the corresponding pipeline stage.

Each of the first execution stage multiplexers require a control signal of width $\lceil log_2(NWP) \rceil + \lceil log_2(NPIPE + 1) \rceil$. The multiplexer control signal format can be subdivided into two fields: field 'pipe_sel' which selects the appropriate pipeline execution stage for obtaining an intermediate result, with 0-th order referring to the register operand read stage and field 'wp_sel' for denoting a specific write port enumeration. The format is illustrated in Fig. 2.

The pseudocode of the data forwarding architecture, presented in Fig. 3, consists of two parts: the first part, named BypassNetwork is the actual forward-

ing unit which is a combinational module required for calculating the data forwarding multiplexer control signals, and the second part, DataForwarding-Multiplexers, implements the actual data forwarding multiplexers. In the pseudocode of Fig. 3 mutually-exclusive selections that can be performed in parallel are denoted by a *for . . . choice* statement, meaning that the iterations can be spatially mapped. It is implementation specific, if a balanced structure or a priority encoded one will be used. A *for . . . choice* can be used as a right hand side expression, typically for implementing, nestable conditional statements for a settable number of choices, controlling assignments to a single entity. They can be implemented by *case-when* or *if-elsif-else-end if* conditional statements in VHDL and *switch-case* blocks in ANSI C. Further, a *bitvector* intrinsic is used similar to the $conv\_std\_logic\_vector(x, y)$ library function in VHDL, where x is an integer, and y the bitwidth assigned for its binary representation. Vectors have a '_v' suffix, so that the *rdata*, *wdata*, *raddr* and *waddr* prefixes denote read and write, register operands and addresses, respectively. For these 2-dimensional vectors (arrays), the first dimension denotes the pipeline stage enumeration and the second addresses a bit-field of the entry for the specified pipeline stage. *alu_opd_v_sel* is the control signal (single-dimensional) vector for the data forwarding multiplexers. Trivially, a *concat* concatenates two vectors to form an aggregate one, in the essence of the VHDL '&' operator. A colon ':' is used for specifying a bit-field in the same way it is used in the Verilog-HDL and it corresponds to *downto* in VHDL.

Table 1 summarizes the notation used for parameters, signals and architectural resources of the SRB.

BypassNetwork()

begin
  for i in 0..NRP-1 do
    alu_opd_v_sel[$(log_2(NWP) + log_2(NPIPE + 1)) \cdot (i + 1) - 1$:$(log_2(NWP) + log_2(NPIPE + 1)) \cdot i$] ←
      for j in 0..NWP-1 choice
        for k in 1..NPIPE choice
          bitvector($j, log_2(NWP)$**) concat bitvector(**$k, log_2(NPIPE + 1)$**) when**
            (a register write is permitted) **and**
            (waddr_v[k,$RAW \cdot (j + 1) - 1 : RAW \cdot j$]
            equals raddr_v[0,$RAW \cdot (i + 1) - 1 : RAW \cdot i$]**);**
          endwhen
        endfor
      endfor
  endfor
end

DataForwardingMultiplexers()

begin

continued on next page

continued from previous page

```
  for i in 0..NRP-1 do
   rdata_v[1,DW · (i + 1) − 1:DW · i] ←
    for j in 0..NWP-1 choice
     for k in 0..NPIPE choice
      if k equals 0
       rdata_v[0, DW · (i + 1) − 1 : DW · i];
      else
       wdata_v[k,DW · (NWP − (j − 1)) − 1:DW · (NWP − j)] when
       (alu_opd_v_sel[(log2(NWP) + log2(NPIPE + 1)) · (i + 1) − 1:(log2(NWP) + log2(NPIPE + 1)) · i]
       equals bitvector(2^NPIPE · (NWP − j),log2(NWP) + log2(NPIPE + 1)));
       endwhen
      endif
     endfor
    endfor
  endfor
end
```

Fig. 3: Pseudocode for the scalable register bypassing (SRB) specification.

Table 1
Notation used in the pseudocode of Fig. 3.

Name	Description
RAW	Register address width
DW	Data word width
NWP	Number of register file write ports and pipeline stage result vector width
NRP	Number of register file read ports
NPIPE	Number of pipeline stages between operand read and write-back

4.1 Incorporating the bypassing mechanisms to the ByoRISC processor

A more detailed view of a 6-stage pipeline ByoRISC architecture is shown in Fig. 4. In the figure, the bypass network part of the forwarding logic (termed as forwarding unit here according to common parlance [15]) and the data forwarding multiplexers as well as their associated interconnections can be easily identified. The multi-port register file has 3 read ports and 2 write ports and is implemented in 6 embedded memories (Xilinx block RAMs). The pipeline stage registers are used to appropriately pass the read data vector ($rdata0$ to $rdata2$), the read operand addresses ($raddr0$ to $raddr2$), and the write operand addresses ($waddr0$ to $waddr1$). The write data vector ($wdata0$ to $wdata1$) is propagated accordingly following its generation at the EX stage of the processor pipeline.

The VHDL description of the forwarding logic is generated with the help of a custom RTL generator accepting the NWP,NRP and $NPIPE$ param-

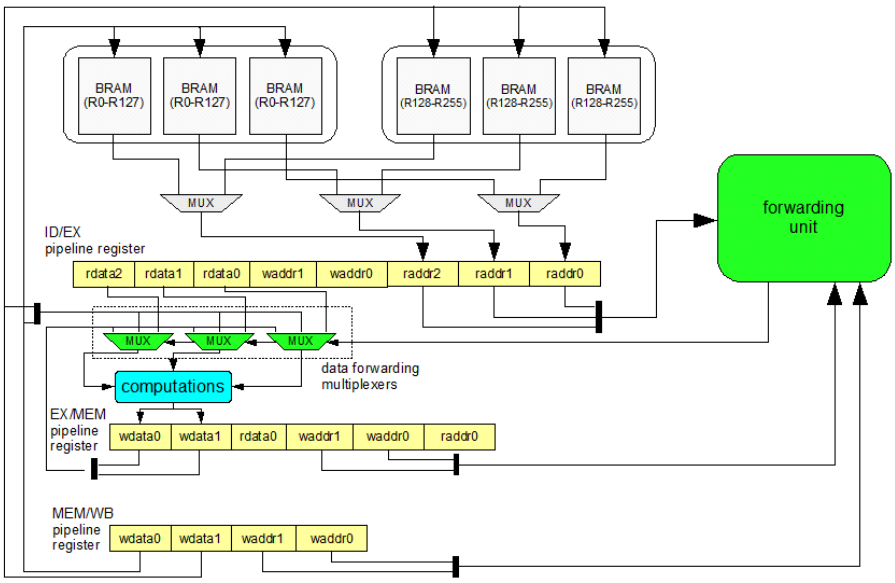


Fig. 4. A more detailed model of a ByoRISC processor instance, clearly depicting the logic associated with forwarding.

eters. The generator produces VHDL'93-compliant code for the forwarding unit (Fig. 5) and the data forwarding multiplexers (Fig. 6) as shown in the corresponding listings.

5 Performance evaluation of the scalable register bypassing architecture

As it has been mentioned before, the Scalable Register Bypassing architecture is part of the ByoRISC processor design [13]. ByoRISC has been designed as a configurable and extensible architecture. It has 32-bit instruction and data word width, Harvard memory architecture, and can be configured for either a 5- or a 6-stage pipeline with full data forwarding. The processor can be extended via ASHEs, that can service possibly multi-cycle custom instructions with an arbitrary number of loads and stores from the data memory. Other important characteristic of the processor is that there is no centralized decoder, but rather a form of distributed decoding mechanisms to each pipeline stage is used, similar to application-specific processors designed with the ASIPMeister tool [17]. Further, for custom instructions with a large number of read and write operands (have been proved for up to 8 read and 8 write operands on recent FPGA architectures), the additional register addresses that do not fit in the instruction encoding limits can be obtained from a LUT, that resides in an interim pipeline stage prior main instruction decode. This scheme is termed as secondary instruction decode (SID).

```
...
entity forwarding_unit is
  -- generics: NWP, NRP, RAW (register address width)
  ...
  port (
    raddr_v_idex     : in  std_logic_vector(NRP*RAW-1 downto 0);
    waddr_v_exmem    : in  std_logic_vector(NWP*RAW-1 downto 0);
    waddr_v_memwb    : in  std_logic_vector(NWP*RAW-1 downto 0);
    RegWrite_v_exmem : in  std_logic_vector(NWP-1 downto 0);
    RegWrite_v_memwb : in  std_logic_vector(NWP-1 downto 0);
    alu_opd_v_sel    : out std_logic_vector(((LOG2(NWP)+2)*NRP)-1 downto 0)
    -- LOG2() is a ceiling log2 function
  );
end forwarding_unit;

architecture rtl of forwarding_unit is
...
  FW_NRPI_GEN : for i in 0 to NRP-1 generate
    L_FW_NRPI_GEN:
      alu_opd_v_sel((LOG2(NWP)+2)*(i+1)-1 downto (LOG2(NWP)+2)*i) <= "010"
      when (RegWrite_v_exmem(0) = '1' and waddr_v_exmem(RAW-1 downto 0) =
      raddr_v_idex(RAW*(i+1)-1 downto RAW*i))
      else "001"
      when (RegWrite_v_memwb(0) = '1' and waddr_v_memwb(RAW-1 downto 0) =
      raddr_v_idex(RAW*(i+1)-1 downto RAW*i))
      else "110"
      when (RegWrite_v_exmem(1) = '1' and waddr_v_exmem(2*RAW-1 downto RAW) =
      raddr_v_idex(RAW*(i+1)-1 downto RAW*i))
      else "101"
      when (RegWrite_v_memwb(1) = '1' and waddr_v_memwb(2*RAW-1 downto RAW) =
      raddr_v_idex(RAW*(i+1)-1 downto RAW*i))
      else "000";
  end generate FW_NRPI_GEN;
end rtl;
```

Fig. 5. VHDL description for the forwarding unit (BypassNetwork) generated for: `NWP=2,NRP=3,NPIPE=2`.

For assessing the performance of the SRB architecture, we evaluate the bypass network over the entire parameter set for a range of values; $NWP : 1-8$, $NRP : 2-8$ and $NPIPE : 1-3$. A specific case of an entire ByoRISC processor with full data forwarding in respect to different values of read and write register file ports is also considered ($NPIPE = 2$). For each case, the timing (either combinational propagation delay or maximum clock frequency) and area requirements are measured for a representative FPGA process. The logic synthesis tool used is Xilinx Webpack ISE 9.2.

Throughout the evaluations, the XC4VLX25 device (FF668 package and '-10' speed grade) which is one of the smallest available Virtex-4 devices. The maximum capacity of XC4VLX25 is 10,572 slices, 72 18-kbit block RAMs (BRAMs) and 48 DSP48 datapath blocks, each one containing an 18×18-bit embedded multiplier.

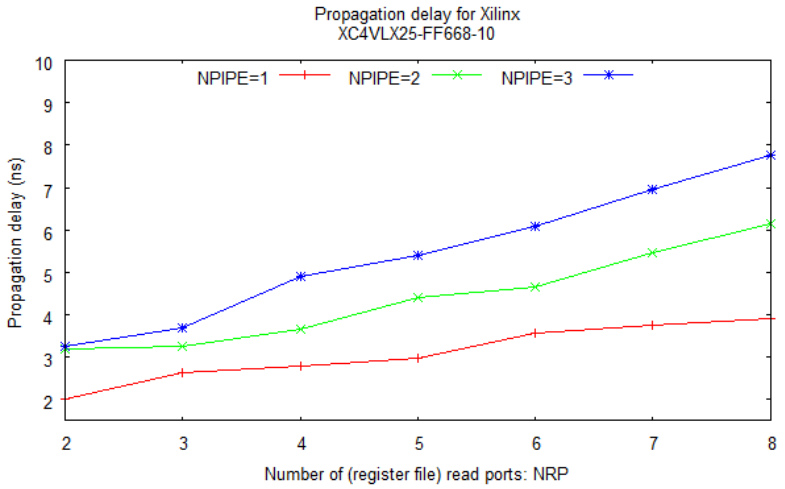
```
...
entity fwdmuxes is
  -- generics: NWP, NRP, RAW
  ...
  port (
    regout_v_ex    : in  std_logic_vector(NRP*32-1 downto 0);
    alu_opd_v_sel : in  std_logic_vector(((LOG2(NWP)+2)*NRP)-1 downto 0);
    rf_wbdata      : in  std_logic_vector(NWP*32-1 downto 0);
    result_v       : in  std_logic_vector(NWP*32-1 downto 0);
    regout_v_fwd   : out std_logic_vector(NRP*32-1 downto 0)
  );
end fwdmuxes;

architecture rtl of fwdmuxes is
begin
  FW_NRPI_GEN : for i in 0 to NRP-1 generate
    L_FW_NRPI_GEN:
      regout_v_fwd(32*(i+1)-1 downto 32*i) <=
        rf_wbdata(32*(NWP-1)-1 downto 32*(NWP-2)) when
          (alu_opd_v_sel((log2(NWP)+2)*(i+1)-1 downto (log2(NWP)+2)*i) =
          conv_std_logic_vector(4*(NWP-2)+1,log2(NWP)+2))
        else
        result_v(32*(NWP-1)-1 downto 32*(NWP-2)) when
          (alu_opd_v_sel((log2(NWP)+2)*(i+1)-1 downto (log2(NWP)+2)*i) =
          conv_std_logic_vector(4*(NWP-2)+2,log2(NWP)+2))
        else
        rf_wbdata(32*(NWP)-1 downto 32*(NWP-1)) when
          (alu_opd_v_sel((log2(NWP)+2)*(i+1)-1 downto (log2(NWP)+2)*i) =
          conv_std_logic_vector(4*(NWP-1)+1,log2(NWP)+2))
        else
        result_v(32*(NWP)-1 downto 32*(NWP-1)) when
          (alu_opd_v_sel((log2(NWP)+2)*(i+1)-1 downto (log2(NWP)+2)*i) =
          conv_std_logic_vector(4*(NWP-1)+2,log2(NWP)+2))
        else
        regout_v_ex(32*(i+1)-1 downto 32*i);
  end generate FW_NRPI_GEN;
end rtl;
```

Fig. 6. VHDL description for the data forwarding multiplexers generated for: `NWP=2,NRP=3,NPIPE=2`.

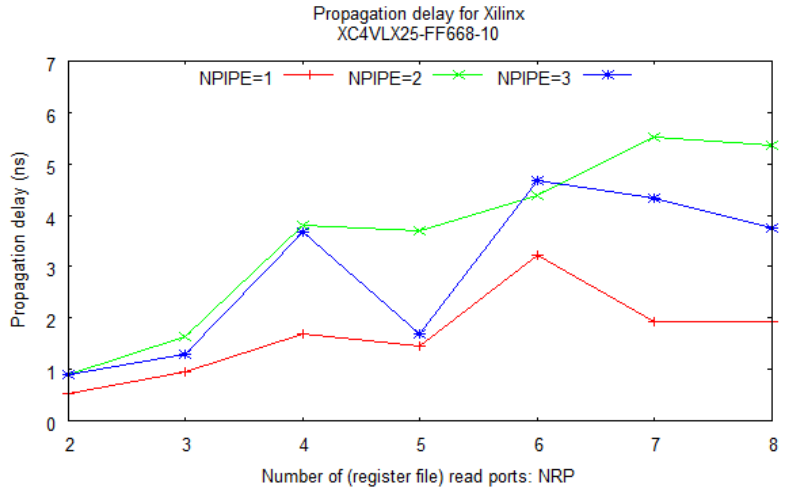
5.1 The SRB bypass network

The bypass network —forwarding unit (fwdunit) and data forwarding multiplexers (fwdmuxes) —of the SRB architecture has been written in VHDL and synthesized for the XC4VLX25 FPGA device. Fig. 7 depicts the delay estimates for different number of supported write ports (NWP=$\{1 \ldots 8\}$) and for pipeline configurations with $NPIPE = 1$ (common EX, MEM stage), $NPIPE = 2$ (separate EX and MEM stages), and $NPIPE = 3$ (two execution stages: EX1, EX2 and MEM stage). A complete set of measurements has been obtained while altering the number of read ports (NRP=$\{2 \ldots 8\}$). The effect of different NWP values is minimal, having an effect of 3% on the estimated propagation delay due to the fact that, in Virtex-4, wide multiplexers are available for accessing the entries read from the register files and thus the need for additional hardware is eliminated.

The chip area requirements are shown in Fig. 8.



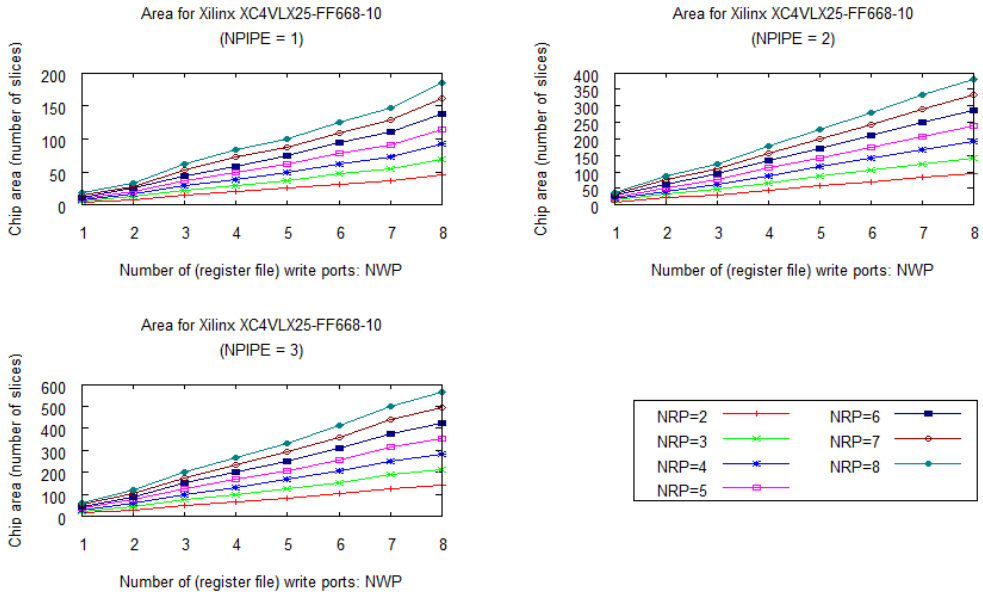
(a) Forwarding unit.



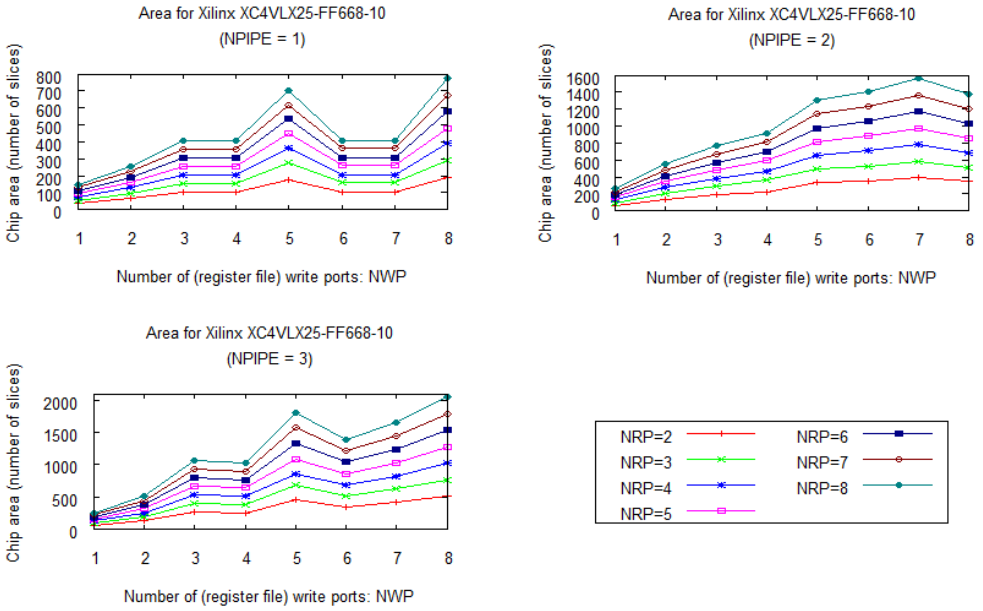
(b) Data forwarding multiplexers

Fig. 7. Propagation delay for the SRB bypass network.

In more detail, 8-to-1 multiplexers can be implemented with one level of logic in Virtex-4 FPGAs. More specifically, pairs of adjacent slices are used, from which a MUXF6 is used to combine the outputs of MUXF5 resources. The effect of increasing the number of read ports is clearly visible with a negative impact of about 2 times the minimum value for the FPGA process. Recent soft-core FPGA-based processors feature maximum clock frequencies in the range of 80-200MHz [4, 5] so that a less than 7ns bypassing delay could prove reasonable especially for a moderate number of write ports. The corresponding area requirements range from negligible to a significant contribution of 12.7% of the device slices for the FPGA. Thus, even though for an FPGA device the chip resources are constrained by default, constructing a bypass network for full data forwarding is feasible.



(a) Forwarding unit.



(b) Data forwarding multiplexers.

Fig. 8. Chip area in number of slices for the SRB bypass network.

5.2 Effect of the SRB architecture on the ByoRISC processor

The bypass network has been incorporated in a synthesizable VHDL description of a complete architectural testbed (featuring more than 20 configurable options), the ByoRISC processor [13]. The multi-port register file of ByoRISC which is related to the operation of the SRB architecture comprises of $NWP \times NRP$ block RAMs, and its design follows the approach described

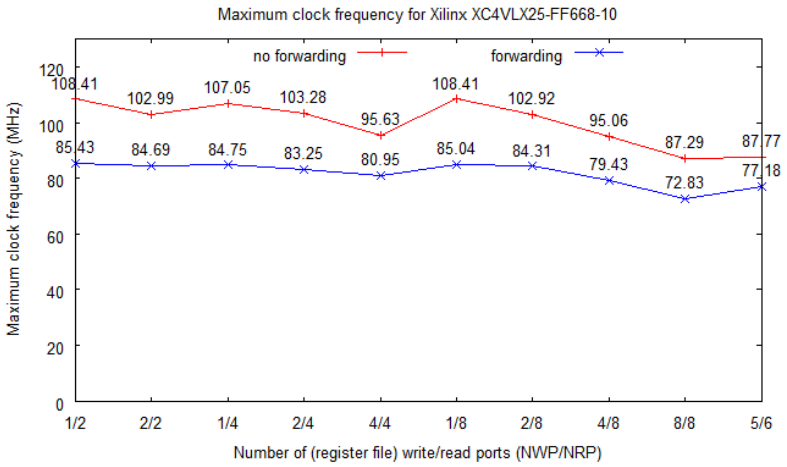


Fig. 9. Propagation time delay for ByoRISC.

in detail in [18]. The testbed processor has been configured for the following options:

- Support for custom instructions, hardware multiplier (pipelined to 4 stages), hardware variable shifter, load-store unit for byte, halfword and word data types.
- Instruction and data memory sizes set to 8KB, respectively.
- Opcode width set to 8.
- 32 general-purpose registers and register address width set to 5.

The exploration procedure has been applied for enabling/disabling full data forwarding, and for different values of read/write ports (NWP={1, 2, 4, 8} and NRP={2, 4, 8} with $NRP > NWP$). Fig. 9 depicts the delay measurements for different numbers of supported read and write ports. The chip area requirements are shown in Fig. 10. In the latter figure, only the data values obtained with forwarding configured are shown. The number of execution pipeline stages has been set to 2. An indicative combination of write/read ports (5/6) has been actually used for an incarnation of ByoRISC in an embedded system for image processing applications (Floyd-Steinberg dithering, halftone image packing/unpacking and encyrption/decryption with the XTEA block cipher) [13].

Studying the data visualized by Fig. 10 it is derived that the number of read-/write ports escalates the required area for the selected Virtex-4 FPGA device by about an order of magnitude, and more specifically from 6% to 50% of the device resources. Also, the use of full data forwarding reduces the maximum achievable clock frequency by 17.9% on average, compared to the corresponding cases without forwarding. Since the power budget of the FPGA device is limited, a full data forwarding solution for a RISC processor able to execute MIMO instructions can be used. MIMO instructions can provide many-fold acceleration of applications: more than 5 × for some MiBench [19] applications

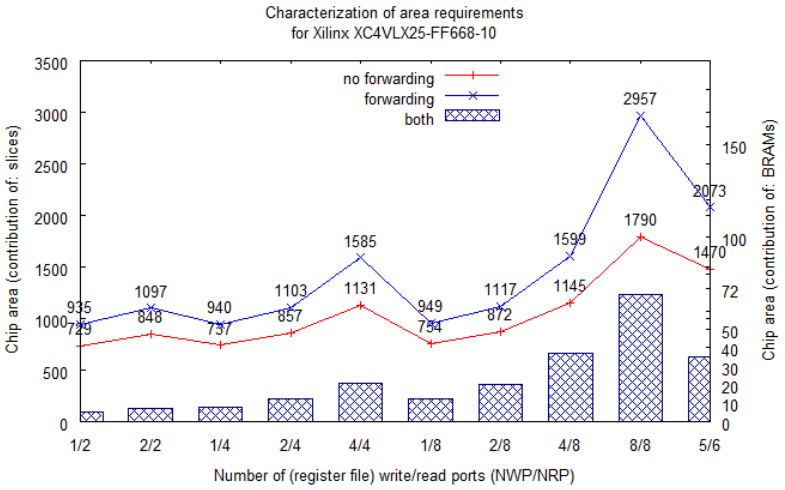


Fig. 10. Chip area for ByoRISC.

as discussed in [20]; for the image processing application set mentioned in this text, up to 9 × cycle acceleration has been observed with a maximum requirement of 5 write and 6 read ports. Comparing the extreme configurations (1/2 vs. 8/8) with forwarding enabled, a moderate performance degradation of 14.75% on the maximum clock frequency is observed. For the extreme configuration (8/8), the use of forwarding has a net effect of about 16.6% compared to its absence for the same number of write/read ports.

5.3 Effect of the data forwarding mechanisms on application speedup

In order to evaluate the performance of the ByoRISC architecture on realistic applications, an image processing flow (IPF) has been used. The IPF which is shown in Fig. 11 processing 256-level greyscale images, comprises of five application kernels:

- *fsdither*: Floyd-Steinberg dithering by error diffusion to a bilevel image
- *htpack*: halftone image packer for eightfold lossless compression of a bilevel image
- *htunpack*: the corresponding unpacking/expanding application to *unpack*
- *xteaenc* and *xteadec*: XTEA encryption and decryption [21], respectively

With the help of the YARDstick toolset [22], application analysis and custom instruction (CI) generation and selection have been applied on the IPF application set. First, the critical basic blocks of the applications have been identified (Table 2). 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 maximizing the cycle-gain priority metric has been

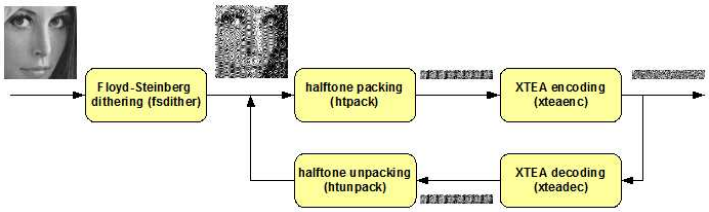


Fig. 11. An image processing flow comprising of dithering, data packing/unpacking and XTEA encryption/decryption.

used.

Table 2
'Hot' basic blocks for some of the IPF applications running on 64 × 64 test images (for $NPIPE = 2$).

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
htunpack.2	88	44032	99.98
xteaenc.2	44	638976	95.63
xteadec.2	44	638976	94.76

A summary of the identified CIs is given in Table 3. 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 and operation issue slots. 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× is expected over the application set based on the estimations (averaging the results of column 'Increm. speedup' over the three applications of IPF) made by YARDstick.

Table 3
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	Increm. 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
htunpack0	5/0/12	35328	7.25	81	9	1.33
xteaenc0	6/5/7	540672	4.55	38	5	0.83
xteadec0	6/5/7	540672	4.55	38	5	0.78

In order to evaluate the effect on the full register bypassing mechanisms on the single-issue width ByoRISC ASIP (Application-Specific Instruction-set Processor), targeting the IPF applications, its cycle-by-cycle behavior has been simulated based on an ArchC [23] model of the processor. A set of measurements was obtained while setting the number of pipeline stages to the values

Table 4
Cycle ($NPIPE$ = 1..3) and timing speedups ($NPIPE$ = 2) for the ByoRISC processor for the baseline configuration (*Baseline*), with custom instructions and partial (*CI + partial bypassing*) or full bypassing (*CI + full bypassing*).

	fsdither	htpack	htunpack	xteaenc	xteadec
Cycle measurements for `NPIPE=1`					
Baseline	18219695	3079201	3464131	40697881	38731798
CI + partial bypassing	6818325	655375	688143	20774934	18808854
CI + full bypassing	6031893	589839	589839	14483478	12517398
Speedup (Baseline vs CI + partial bypassing)	2.67	4.70	5.03	1.96	2.06
Speedup (Baseline vs CI + full bypassing)	3.02	5.22	5.87	2.81	3.09
% difference due to bypassing	11.53	10.00	14.29	30.28	33.45
Cycle measurements for `NPIPE=2`					
Baseline	15990089	3200981	3585911	39616537	37650454
CI + partial bypassing	7080981	655375	688143	20742166	18776086
CI + full bypassing	6294549	589839	589839	14450710	12484630
Speedup (Baseline vs CI + partial bypassing)	2.26	4.88	5.21	1.91	2.01
Speedup (Baseline vs CI + full bypassing)	2.54	5.43	6.08	2.74	3.02
% difference due to bypassing	11.11	10.00	14.29	30.33	33.51
Timing measurements (in μs) for `NPIPE=2`					
Baseline	187.24	37.48	41.99	463.91	440.89
CI + partial bypassing	80.65	7.46	7.84	236.25	213.86
CI + full bypassing	81.58	7.64	7.64	187.28	161.8
Speedup (Baseline vs CI + partial bypassing)	2.32	5.02	5.36	1.96	2.06
Speedup (Baseline vs CI + full bypassing)	2.30	4.90	5.49	2.48	2.72
% difference due to bypassing	-1.15	-2.41	2.47	20.73	24.34
Cycle measurements for `NPIPE=3`					
Baseline	16906211	3617673	4002603	40763417	38797334
CI + partial bypassing	6032917	491535	524303	19791894	17825814
CI + full bypassing	5246485	425999	425999	13500438	11534358
Speedup (Baseline vs CI + partial bypassing)	2.80	7.36	7.63	2.06	2.18
Speedup (Baseline vs CI + full bypassing)	3.22	8.49	9.40	3.02	3.36
% difference due to bypassing	13.04	13.33	18.75	31.79	35.29

$NPIPE$ = 1..3. Table 4 summarizes the number of cycles, and percentage differences of cycle speedups for three distinct cases:

(1) Baseline processor (`NWP=1,NRP=2`) with full bypassing
(2) Custom instructions and partial bypassing (bypassing organization of the baseline processor)
(3) Custom instructions and full bypassing

Especially for the case of $NPIPE = 2$, the processor has been synthesized on Xilinx ISE Webpack 9.2 and its timing characteristics (minimum clock period) have been measured. For this case, along with the cycle speedups, speedups and associated percentage differences taking into account the minimum clock period that can be achieved are shown in Table 4. The clock period estimates are shown in Fig. 9.

The results reveal cycle speedups of 2.5× (fsdither, $NPIPE = 2$) to 9.44× (htunpack, $NPIPE = 3$) and 3.9× ($NPIPE = 1$) to 5.5× ($NPIPE = 3$) in average. For the specific case of $NPIPE = 2$ for which the ByoRISC processor has been successfully tested in both simulation and FPGA environments, the minimum clock period is taken into account to obtain the execution time estimates. In this case, the maximum average speedups is only reduced by 9.65% in order to obtain a 3.58× speedup over the "baseline" case when using

custom instructions and full forwarding.

Multi-cycle register files could be used in order to sequentially pass register file write operands, and to permit the reading of values in chunks of one or two read values for the same multi-input/multi-output custom instruction. The timing overhead of having 2 read/1 write port register file and using partial forwarding to 8 read/8 write multi-port register file and support of full forwarding (which is the maximum supported by the ByoRISC architecture) is 32.8% when $NPIPE = 2$. In all cases examined (for different $NPIPE$ values), the speedup due to the multi-operand custom instructions ranges from 3.3× to 5.5× for the test applications, thus in order to obtain the execution time speedups these values would have to be reduced by about one third. This is a pessimistic view due to two distinct reasons. Firstly, there is an additional cycle overhead to the baseline case due to always having multiple cycles for reading and writing multiple operands. Secondly, opportunities for data forwarding will always be missed due to pipeline bubbles introduced by stalling the pipeline at the instruction decode stage for writing or reading values to or from the multi-cycle limited-port register file.

6 Conclusions

In this paper, the application of a scalable register bypassing scheme on contemporary soft-core processors has been examined in detail. The Scalable Register Bypass (SRB) architecture can be configured for different number of general-purpose register file read (NRP) and write (NWP) ports and can be incorporated in a wide range of pipeline processors with a number of execution pipeline stages ($NPIPE$) subject to quite generic assumptions. An RT-level specification, fully describing the SRB, has been developed and is described herein. Its viability has been proved, for contemporary FPGA solutions by assessment of logic synthesis estimates based on the RTL VHDL description of the SRB bypass network as well as the description of a complete processor using scaled forms of the SRB. For the examined case of an image processing application set with average speedup up to 5.5×, the negative effect of applying full forwarding on the maximum clock frequency of a Virtex-4 FPGA device when using 8 read and 8 write ports is less than 15%.

References

[1] ARC cores.
URL `http://www.arc.com`

[2] R. Gonzalez, Xtensa: A configurable and extensible processor, IEEE Micro 20 (2) (2000) 60–70.

[3] Gaisler research.
URL `http://www.gaisler.com`

[4] Xilinx home page.
URL `http://www.xilinx.com`

[5] Altera Nios II home page.
URL `http://www.altera.com/products/ip/processors/nios2/`

[6] A. Abnous, N. Bagherzadeh, Pipelining and bypassing in a VLIW processor, IEEE Transactions on Parallel and Distributed Systems 5 (6) (1994) 658–664.

[7] S. A. Trainis, Modelling the hardware cost of full register bypassing in a multiple instruction issue processor, Journal of Systems Architecture 43 (1997) 39–46.

[8] K. Fan, N. Clark, M. Chu, K. V. Manjunath, R. Ravindran, M. Smelyanskiy, S. Mahlke, Systematic register bypass customization for application-specific processors, in: Proceedings of the 14th International Conference on Application-specific Systems, Architectures and Processors, The Hague, The Netherlands, 2003, pp. 64–74.

[9] A. Shrivastava, S. Park, E. Earlie, N. Dutt, A. Nicolau, Y. Paek, Retargetable pipeline hazard detection for partially bypassed processors, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 26 (10).

[10] M. Sami, D. Sciuto, C. Silvano, V. Zaccaria, R. Zafalon, Low-power data forwarding for VLIW embedded architectures, IEEE Transactions on Very Large Scale Integration (VLSI) Systems 10 (5) (2002) 614–622.

[11] N. Goel, A. Kumar, P. R. Panda, Power reduction in VLIW processor with compiler driven bypass network, in: Proceedings of the 20th International Conference on VLSI Design (VLSI Design 2007), Sixth International Conference on Embedded Systems (ICES 2007), Bangalore, India, 2007, pp. 233–238.

[12] A. S. Terechko, Clustered VLIW architectures: a quantitative approach, Ph.d. thesis, Technical University of Eindhoven, Eindhoven, The Netherlands (February 2007).

[13] N. Kavvadias, S. Nikolaidis, The ByoRISC configurable processor family, in: accepted for publication in the proceedings of the IFIP/IEEE VLSI-SoC 2008 - International Conference on Very Large Scale Integration, Rhodes Island, Greece, 2008.

[14] L. Pozzi, K. Atasu, P. Ienne, Exact and approximate algorithms for the extension of embedded processor instruction sets, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 25 (7) (2006) 1209–1229.

[15] J. Hennessy, D. Patterson, Computer Architecture: A Quantitative Approach, 2nd Edition, Morgan Kaufmann Publishers, San Francisco, CA, 1996.

[16] D. E. Knuth, MMIXware: A RISC Computer for the Third Millennium, Springer-Verlag, 1999.
URL http://www-cs-faculty.stanford.edu/~knuth/mmix-news.html

[17] ASIPMeister application specific instruction-set processor design system.
URL http://www.eda-meister.org/asip-meister/

[18] M. A. R. Saghir, R. Naous, A configurable multi-ported register file architecture for soft core processors, in: Proceedings of the 2007 International Workshop on Applied Reconfigurable Computing (ARC 2007), Mangaratiba, Rio de Janeiro, Brazil, 2007, pp. 14–25.

[19] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, R. B. Brown, MiBench: A free, commercially representative embedded benchmark suite, in: Proceedings of the 4th annual IEEE International Workshop on Workload Characterization, Austin, Texas, USA, 2001.

[20] N. Kavvadias, S. Nikolaidis, A flexible instruction generation framework for extending embedded processors, in: Proceedings of the 13th IEEE Mediterranean Electrotechnical Conference (MELECON 2006), Benalmadena (Malaga), Spain, 2006, pp. 125–128.

[21] R. M. Needham, D. J. Wheeler, TEA extensions, Technical report, Computer Laboratory, University of Cambridge (Oct. 1997).

[22] N. Kavvadias, 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, 2007.

[23] The ArchC resource center.
URL http://www.archc.org