*Int. J. Engng Ed.* Vol. 26, No. 3, pp. 584–592, 2010
Printed in Great Britain.

0949-149X/91 $3.00+0.00
© 2010 TEMPUS Publications.

# A Balanced HW/SW Teaching Approach for Embedded Microprocessors*

U. MEYER-BAESE[1], GUILLERMO BOTELLA[2], ENCARNACIÓN CASTILLO[3] and ANTONIO GARCÍA[3]

*[1] Florida State University, Department of Electrical and Computer Engineering, Tallahassee, Florida, USA. E-mail: umb@eng.fsu.edu*

*[2] Department of Computer Architecture and Automation of Complutense University of Madrid, Spain and Florida State University, Department of Electrical and Computer Engineering, Tallahassee, Florida, USA.*

*[3] Departmento de Electrónica y Tecnología de Computadores, Universidad de Granada, Campus Universitario Fuentenueva, 18071 Granada, Spain.*

*Currently popular textbooks on Embedded Microprocessors are analysed in depth and reveal the inherent weaknesses of these books. For example, while even advanced hardware (HW) concepts are presented, the textbooks fail to provide descriptions about the development of software (SW) tools that put these microprocessors (μPs) to work. Conversely, we provide an intimate knowledge of the close relationship between the μP design and the development tools with two teaching modules based on an architectural description language (ADL). The URISC model (popular since the IJEE paper 20 years ago) and the Educational RISC (ERISC) process models are developed in an iterative refinement of the instruction set. For all intermediate steps, development tools (assembler, linker, loader, C compiler) are generated which teach students the basics of embedded processor theory and design procedure. The processors are debugged using three example programs, synthesized to HDL for ASIC/FPGA designs, and are tested on popular Altera and Xilinx University development boards to offer hands-on design experience.*

**Keywords:** RISC; embedded microprocessor; FPGAs; ADL

## 1. INTRODUCTION

MOST OF TODAY'S MICROPROCESSORS are employed in embedded systems. Embedded systems are usually characterized as those that include a microprocessor but do not have the typical components of a computer, such as a keyboard, monitor, or mouse. Examples of embedded systems range from cellular phones and digital clocks to GPS devices, video recorders, www routers to households, and other electronic entertainment devices [1]. A modern car, for example, typically uses 50–100 microprocessors [2]. Embedded systems are not usually real-time systems necessitating that certain computation be accomplished by a certain deadline, like in a ABS car system. Embedded systems are often resource-limited by price, power dissipation, memory or storage. Although many embedded systems require low-power dissipation, the implemented algorithms, like the error turbo correction coding used in UMTS phones, are computationally demanding. Nevertheless, embedded processors currently perform sophisticated tasks and run these complex algorithms. The microprocessors in a car use an estimated 100 million lines of code; GPS and radio alone account for 20 million lines of code [2].

Given the wide range of embedded systems applications, it is unsurprising that not a single microprocessor can cover all of the requirements in HW and SW, so customization is necessary. This is exactly the job description of a modern embedded system engineer: to have an intimate knowledge of the hardware (e.g. the microprocessor and its peripherals) and the software (e.g. the algorithms and coding in a computer language such as assembler or C).

Today, the embedded system design course and laboratories (C&L) offered at universities depends on the student level and department target, and can take on many different HW/SW mixture forms. For Computer Science (CS) students an Embedded System course usually focuses on C++, JAVA or UML in combination with some facts about the underlying hardware components. Twelve books for a typical CS course are listed in the literature survey [22–34]. A typical Embedded Microprocessor Design course taught in Electrical Engineering (EE) will focus on the design of a microprocessor in hardware description language, such as VHDL or Verilog, only leaving out the SW part almost completely. Seven popular books used in EE are listed [15–22]. In the middle of these extremes will be an Embedded System Design course for a student in a Computer Engineering (CE) Department focusing on the system design and integration of commercial, off-the-shelf

* Accepted 26 December 2009.

(COTS) components. Here a hardcore (fixed netlist) embedded microprocessor (e.g. PowerPC, PIC, ARM) is augmented with peripheral (memory, USB, I2C etc.) components and programs in C or assembler are developed [12–14].

Our C&L teaching modules try to bridge between the HW-only approach in EE and the SW-only approach in CS. But the main advantage of using an architecture description language (ADL) would be that there is no restriction, as in a current CE course, to a COST microprocessor—the ADL allows design of any kind of microprocessor with full control over the instruction set of the microprocessor, ranging from low-cost 8-bit microcontroller to high performance 128-bit VLIW microprocessors. The use of a mixed level ADL in addition will allow to generate high quality software tools. A survey of three commercial and three public domain ADL tools is given later.

Most educational materials and books used today do not provide the intimate knowledge of HW and SW working together. This is corroborated by the fact that most logic or computer design books [16] teach the HW design of a microprocessor in all details—including how to design a CISC and RISC [18] or URSIC $\mu$P [15], instruction pipelines [19,20], register allocation, caches [12], and memory management units [13, 14, 21, 22]—but contain little or no coverage on the SW tool development that put these parts to work. Figure 1 shows the HW design and SW tool imbalance in the current textbooks available for an embedded microprocessor course. The exploded pie part shows the pages spent on SW development tool description in theses textbooks. As is evident, 94% of the information within the textbooks describes HW design. Books with limited coverage and description using standard software without describing development of HW or SW (e.g. [23–34]) are not included. Even books with titles like 'Embedded System' [13, 14, 26] fail almost completely in this integral part of embedded system design.

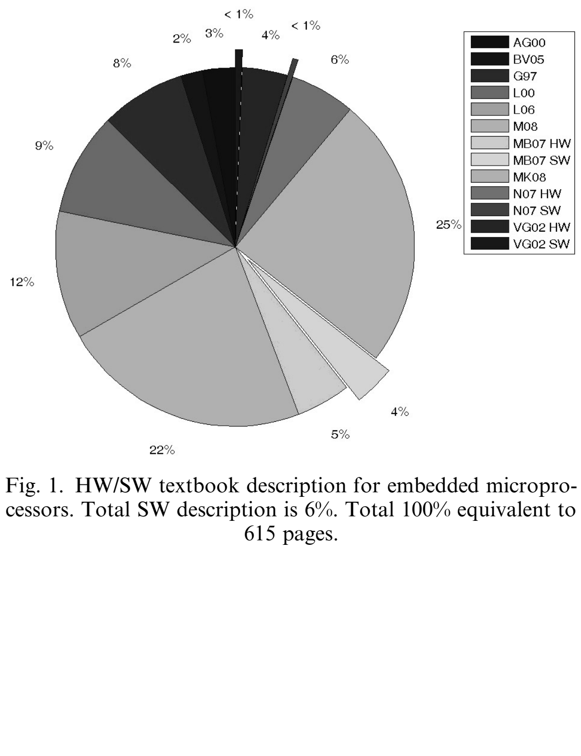


Fig. 1. HW/SW textbook description for embedded microprocessors. Total SW description is 6%. Total 100% equivalent to 615 pages.

Clearly, developing a set of quality SW tools for a processor is a challenging task, and perhaps many authors feel more comfortable covering topics like lexical analysis, compiler-compiler and grammar [7]. But nonetheless, a close understanding of the relationship between the development software tools and the underlying hardware is essential for those learning embedded system design.

## 2. DESIGNING APPLICATION-SPECIFIC PROCESSORS WITH HDL

In the classical embedded microprocessor approach, we start with an architecture description and then develop the instruction set and architecture. We then write the HDL code for the processor and write, based on this developed architecture, the development tools (e.g. the instruction set simulator (ISS), the C compiler, the assembler, etc.). While this hand-coded HDL may allow us to obtain an extremely small core size by taking advantage of the underlying logic blocks (e.g. Pico Blaze by Xilinx used the $32 \times 1$ LUT to implement the processor registers and save many resources), the disadvantage is that any change in hardware also needs to be coded in all development tools. This is considered a major source of cost and inefficiency in embedded processor design when using HDL.

This classical approach has been used in a computer architecture undergraduate course (since 2001) and an ASIC design graduate course (since 2006) at the FAMU-FSU College of Engineering in Tallahassee, resulting in many functional $\mu$Ps. However, the software development tools were far from satisfactory.

## 3. DESIGN APPROACH USING ADL

Embedded processor designs typically begin at a level of abstraction far beyond the instruction set architecture (ISA) and require several architecture exploration cycles before the optimum hardware/software partitioning for a particular application is found. This process requires a number of tools for software development and profiling. These are normally written manually—a time consuming, inefficient and error-prone task. With the introduction of so-called Architecture Description Language (ADL), the design process can be efficient and reliable [3]. Early ADL was either structure-orientated (MIMOLA, UDL/I) or behaviour-orientated (Valen-C or ISDL). Later, mixed ADLs such as nML, LISA, HMDES, ASIP Meister, Flexware, TDL and EXPRESSION adopted an integrated approach—the language captures both the structural as well as the behavioural design of the embedded processor also called application-specific integrated processor (ASIP). Several of these tools have been developed in academia, and some have become commercial tools such as LISA tools developed at ISS RWTH Aachen and now the Processor Designer product of Coware Inc. (CA, US) [9–11] or nML developed at TU Berlin and now a Target Compiler Technology (Belgium) product [4]. The ASIP development flow for the LISA tool set is similar to the classic approach, the only difference is that one LISA 2.0 based processor description is used to specify the behavior of the microprocessor as well as the generated development tools.

The nML comes with a Chess/Checkers retargetable C compiler, an RTL synthesis generator GO, and RISK—a test-program generator. These commercial tools have been used in a wide variety of products, such as portable audio and hearing instruments by CoolFlux, Wireline modems ADSL2+ by STMicroelectronics, wireless modems HSDPA by Nokia, and TI video accelerators and network processors [4]. A brief comparison of tools properties that can be used for an ADL University course is given in Table 1.

ASIP Meister is a GUI-based processor system and was used by 180 academic institutions in 37 countries during 2002–2005. Since 2006, ASIP Solution Inc. has taken over the maintenance and further development of ASIP Meister. The GUI-based platform is somehow more restrictive than nML or LISA, but allows for a much improved development time. An MIPS processor, for instance, could be developed in eight hours, and DLX in only 3.5 hours.

The academic ADLs most often focus on special features. EXPRESSION—developed by UC, Irvine—is a popular research language that allows the exploration of memory hierarchies. MADL, developed at Princeton, allows the design of superscalar processors and has an easy C-compiler interface. MAML is an XML-based ADL that has the support to build VLIW and multicore processors. However, the quality of the generated HDL (if any) of the academic ADLs may not be sufficient for industry products [4].

## 4. APPROACH TO TEACHING A LISA-BASED MICROPROCESSOR COURSE

The CoWare Processor Designer, formerly known as the LISAtek processor design platform (LPDP), was originally developed at RWTH Aachen and is now a product of CoWare Inc. This design flow was used to develop the embedded processors. The LISA language supports a profile-based and stepwise refinement of processor models down to cycle-accurate and even VHDL or Verilog RTL synthesis models for fast custom VLSI implementation. In a very elegant way, it avoids model inconsistencies otherwise inevitable in traditional design flows. Microprocessors from simple RISC to highly complex VLIW processors have been described and successfully implemented using the Processor Designer for FPGAs and cell-based ASICs. There are more than 40 LISA models in the industry and academia from different architectural categories (RISC, PDSP and ASIP). These include different ARM and MIPS models, PDSP from TI and StarCore, as well as ASIPs from Infineon (ICORE), STMicroelectronis, etc. CoWare provides 14 different example/starting-point models. This includes seven tutorial models that are used as part of CoWare training material. Some have multiple versions that contain over 10 different designs as seen in the QSIP_X model. Four starting point models are provided and used as skeletons for starting a new architecture. Three different IP models for classic architectures are also included. All models are instruction accurate, and most of the models are Harvard-type RISC models that are also cycle accurate. Pipeline stages vary from three to five. All types of modern processor are provided from simple RISC (QSIP), over PDSP like LT_DSP_32p3 to VLIW LT_VLIW_32p4, to special processors like a 16- to 4096-point FFT processor LT_FFT_48p3.

### *4.1 The Lisa language*

Processor design using LISA is organized in different sections [9–11] beginning with a resource section to specify the program, data memory, registers, program counters and the pipeline of processors. A discussion of a partial resource section from the URISC processor model follows:

```
RESOURCE {
  MEMORY_MAP {PAGE(0),RANGE(0x00000,
0x007F) -> prog_mem[(15..0)];}
  RAM uint32 prog_mem { . . . . .
  };
  PROGRAM_COUNTER TClocked<uint8> PC;
// Program counter
```

Table 1: Comparison of commercial tools and tools from academia

| | Advantage | Disadvantage |
|---|---|---|
| **Professional tool:** **LISA, nML, ASIP Meister** | • Variety of Training μP models<br>• FPGA HDL code generation<br>• Vendor tutorials<br>• bug fix | • Expensive |
| **Public Domain:** **EXPRESSION, MADL, MAML** | • Free<br>• Support of special features | • No production quality HDL generation<br>• Limited bug fix<br>• No library training of μP models |

```
  REGISTER TClocked<int8> r[0..15]; //
Register file
  PIN IN uint8 iport; // Input PINs of the
core
  . . . . .
  PIPELINE pipe = {FET; EXE};
  PIPELINE_REGISTER IN pipe {
    uint8 PPC;
    uint16 PIW;
  };
  . . . . .
  };
  UNIT PIPE_FD { fetch; };
  UNIT PIPE_EW {URISC; };
  } // End of resources
```

The behaviour and timing of the processor are described by LISA operations. The instructions encoding can be arranged in a hierarchical tree so that general behaviour is specified in the parent operation, while specialized behaviour is described in child operations. For example, consider addressing modes of instructions. Absolute or relative addressing is often used by several parent operations, and reusing the addressing modes makes the processor description shorter and easier to read.

Each LISA operation includes just enough information that HW architecture and SW tools can be generated. The elements are defined as follows and include examples:

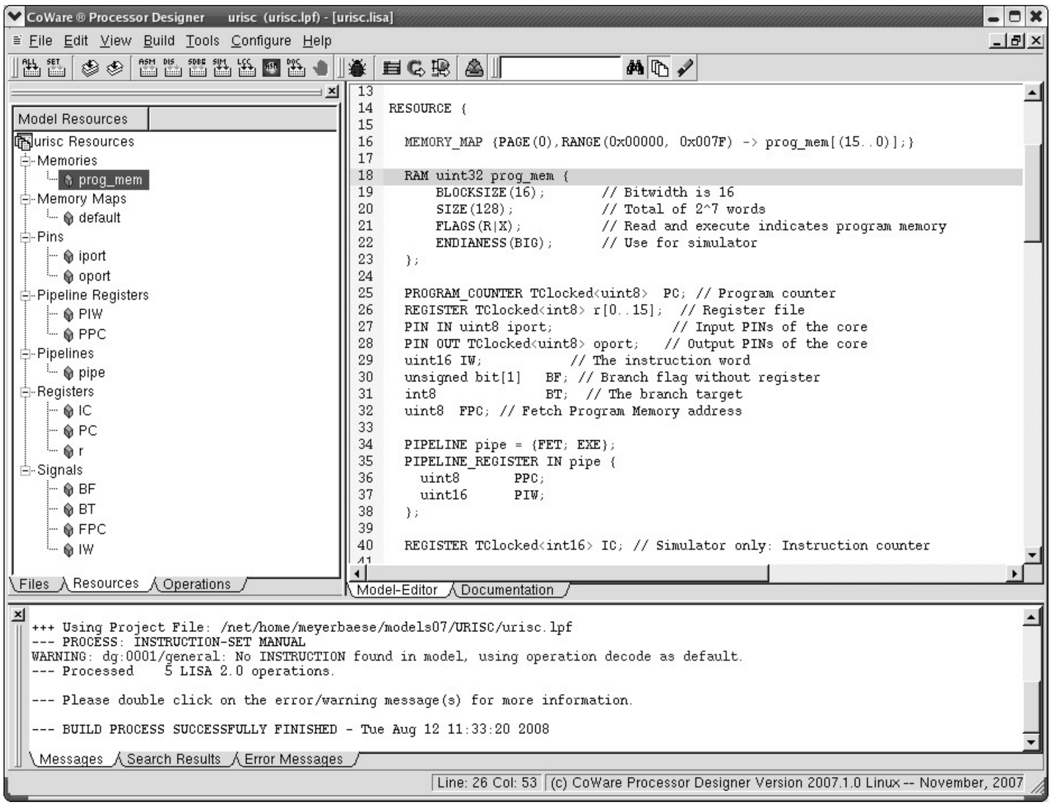
- The DECLARE element is used to reference elements from other operations and to define used resources.
```
DECLARE{LABEL src, dst;
        INSTANCE addr;
        GROUP mode = { absolute_
        addressing || relative_
        addressing };}
```
- The CODING includes the binary coding of the instructions as a sequence of coding fields. The value can be "0","1" or X (don't care) or can have a reference to the coding field of other LISA operations.
```
CODING { dst=0bx[4] src=0bx[4] mode
addr }
```
- The SYNTAX describes the assembler syntax that references variables via labels.
```
SYNTAX { ''URISC'' '' r['' dst ''],
r['' src ''], '' addr }
```
- The BEHAVIOR description is included in the data path function of the processor. Coding is sequential just as in regular C coding. Resources such as registers, memories, flags and pins can be accessed, modified and stored.
```
BEHAVIOR { BF = 0; // reset branch flag
          s0 = (int8) s0 * s1; //
          compute product
          s1 = s2; s2 = s3; s3 = 0;} //
          pop stack by one
```
- The ACTIVATION allows a LISA operation to activate another operation or a group of operations typically for the next pipeline stage. Here is an example from the ERISC processor that also includes the coding root tree specification with CODING AT.
```
OPERATION decode IN pipe.EXE {
  DECLARE { GROUP instruction = {
  JMP||BEQ||BNE||PRINT||SCAN||
  PUSH||PUSHI||POP||MUL||SUB||
  NEG||ADD||NOP};}
  CODING AT (IN.PPC) { IN.PIW ==
  instruction }
  SYNTAX { instruction }
  ACTIVATION { instruction }
    }
```
- The DOCUMENTATION allows specification of a description that will be included in the instruction set manual generated automatically by the processor designer.
```
DOCUMENTATION (''decode''){
  The decode operation specifies the
  coding root with AT and calls the
  instructions. }
```

The Processor Designer has a comfortable model editor with colour coding that allows easy verification of the language elements in use (e.g. key words appear in blue, values in red, comments in green, see Fig. 2). After successful specification of the processor, the Processor Designer automatically generates the SW development tools. The functionality of an application can then be verified by a run through the debugger. Figure 3 shows the Processor Debugger with Disassembler, Profiler, Register, and Memory display. After successful debugging, the processor is ready for synthesis to measure performances like the area, power dissipation or speed. Along with the Verilog or VHDL code for cell-based ASIC or FPGAs, Synopsys synthesis scripts and Xilinx ModelTech simulation scripts can also be generated. Files could also be run through Altera Quartus without problems since only elementary standard HDL constructs are used. Figure 4, for instance, shows a simulation done with Altera Quartus software of the I/O program.

After we have presented the design flow using a mixed ADL, we can then start to develop two educational models to be used in class or lab. For educational purposes, we have developed two processor models that have previously been used in traditional μP design courses [5, 6, 12, 15].

### *4.2 URISC processor model*

The URISC processor model, introduced 20 years ago in another IJEE paper, is a popular architecture that has been used for many years in Computer Architecture [5, 6] and in HDL courses [15]. It shows the ultimate limits of the reduced instruction set computer (RISC) approach, i.e. a microprocessor with a single instruction. The instruction subtracts source 1 operand from operand 2, replaces source 2 with the results, and jumps to a target address if the result of the subtraction is negative. At the time URISC was introduced



```

RESOURCE {

  MEMORY_MAP {PAGE(0),RANGE(0x00000, 0x007F) -> prog_mem[(15..0)];}

  RAM uint32 prog_mem {
      BLOCKSIZE(16);       // Bitwidth is 16
      SIZE(128);           // Total of 2^7 words
      FLAGS(R|X);          // Read and execute indicates program memory
      ENDIANESS(BIG);      // Use for simulator
  };

  PROGRAM_COUNTER TClocked<uint8>  PC; // Program counter
  REGISTER TClocked<int8> r[0..15];  // Register file
  PIN IN uint8 iport;                // Input PINs of the core
  PIN OUT TClocked<uint8> oport;   // Output PINs of the core
  uint16 IW;           // The instruction word
  unsigned bit[1]   BF; // Branch flag without register
  int8              BT;  // The branch target
  uint8  FPC; // Fetch Program Memory address

  PIPELINE pipe = {FET; EXE};
  PIPELINE_REGISTER IN pipe {
    uint8      PPC;
    uint16     PIW;
  };

  REGISTER TClocked<int16> IC; // Simulator only: Instruction counter
```

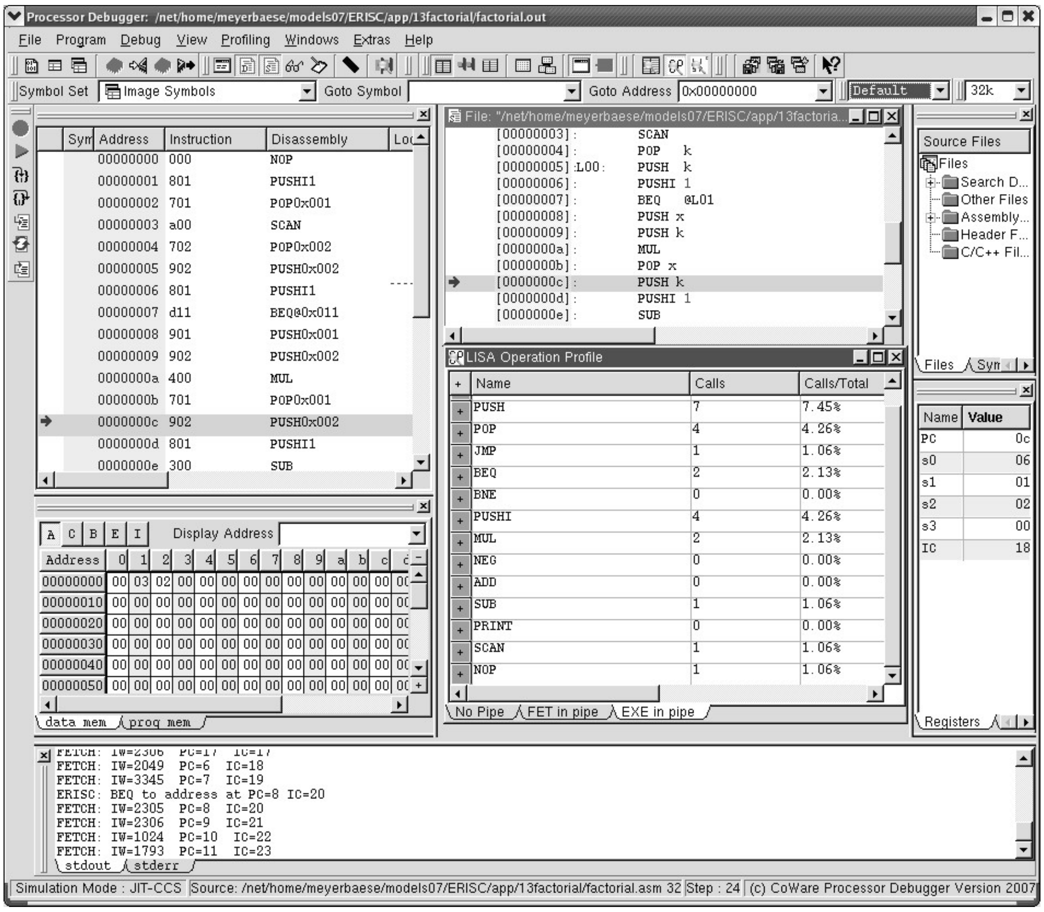
```
+++ Using Project File: /net/home/meyerbaese/models07/URISC/urisc.lpf
--- PROCESS: INSTRUCTION-SET MANUAL
WARNING: dg:0001/general: No INSTRUCTION found in model, using operation decode as default.
--- Processed    5 LISA 2.0 operations.

--- Please double click on the error/warning message(s) for more information.

--- BUILD PROCESS SUCCESSFULLY FINISHED - Tue Aug 12 11:33:20 2008
```

Line: 26 Col: 53 | (c) CoWare Processor Designer Version 2007.1.0 Linux -- November, 2007

Fig. 2. The Processor Designer model editor.



Goto Address: 0x00000000

| Sym | Address | Instruction | Disassembly | Loc |
|---|---|---|---|---|
| | 00000000 | 000 | NOP | |
| | 00000001 | 801 | PUSHI1 | |
| | 00000002 | 701 | POP0x001 | |
| | 00000003 | a00 | SCAN | |
| | 00000004 | 702 | POP0x002 | |
| | 00000005 | 902 | PUSH0x002 | |
| | 00000006 | 801 | PUSHI1 | |
| | 00000007 | d11 | BEQ@0x011 | |
| | 00000008 | 901 | PUSH0x001 | |
| | 00000009 | 902 | PUSH0x002 | |
| | 0000000a | 400 | MUL | |
| | 0000000b | 701 | POP0x001 | |
| ➔ | 0000000c | 902 | PUSH0x002 | |
| | 0000000d | 801 | PUSHI1 | |
| | 0000000e | 300 | SUB | |

```
Address   0  1  2  3  4  5  6  7  8  9  a  b  c  d
00000000 00 03 02 00 00 00 00 00 00 00 00 00 00 00
00000010 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000020 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000030 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000040 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000050 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

File: "/net/home/meyerbaese/models07/ERISC/app/13factoria...

```
  [00000003]:        SCAN
  [00000004]:        POP   k
  [00000005]:L00:    PUSH  k
  [00000006]:        PUSHI 1
  [00000007]:        BEQ   @L01
  [00000008]:        PUSH x
  [00000009]:        PUSH k
  [0000000a]:        MUL
  [0000000b]:        POP x
➔ [0000000c]:        PUSH k
  [0000000d]:        PUSHI 1
  [0000000e]:        SUB
```

LISA Operation Profile

| Name | Calls | Calls/Total |
|---|---|---|
| PUSH | 7 | 7.45% |
| POP | 4 | 4.26% |
| JMP | 1 | 1.06% |
| BEQ | 2 | 2.13% |
| BNE | 0 | 0.00% |
| PUSHI | 4 | 4.26% |
| MUL | 2 | 2.13% |
| NEG | 0 | 0.00% |
| ADD | 0 | 0.00% |
| SUB | 1 | 1.06% |
| PRINT | 0 | 0.00% |
| SCAN | 1 | 1.06% |
| NOP | 1 | 1.06% |

| Name | Value |
|---|---|
| PC | 0c |
| s0 | 06 |
| s1 | 01 |
| s2 | 02 |
| s3 | 00 |
| IC | 18 |

```
FETCH: IW=2306  PC=17  IC=17
FETCH: IW=2049  PC=6  IC=18
FETCH: IW=3345  PC=7  IC=19
ERISC: BEQ to address at PC=8 IC=20
FETCH: IW=2305  PC=8  IC=20
FETCH: IW=2306  PC=9  IC=21
FETCH: IW=1024  PC=10  IC=22
FETCH: IW=1793  PC=11  IC=23
```

Simulation Mode : JIT-CCS | Source: /net/home/meyerbaese/models07/ERISC/app/13factorial/factorial.asm 32 | Step : 24 | (c) CoWare Processor Debugger Version 2007

Fig. 3. ERISC debug of factorial program.

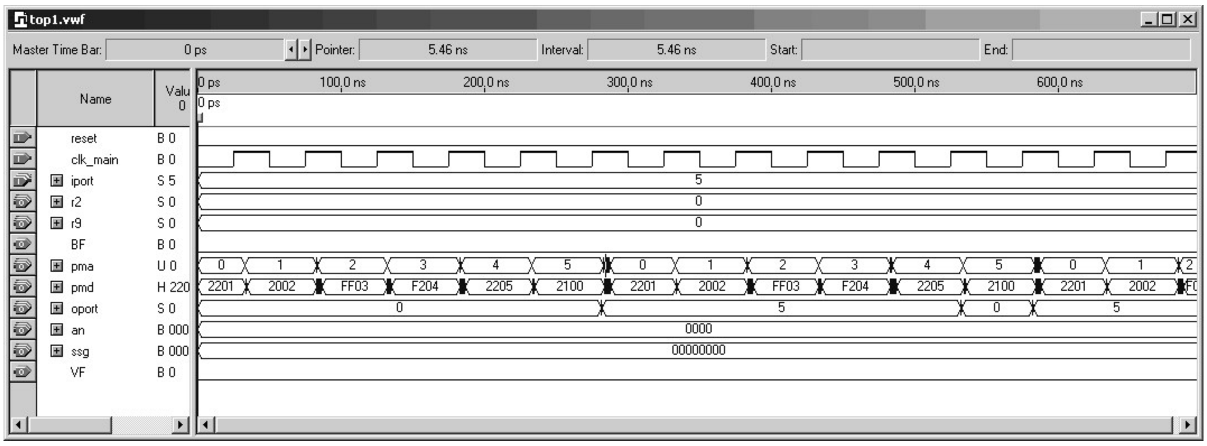


Fig. 4. URISC test program I/O on Altera Quartus. Values 5 is taken from the iport and displayed on the oport.

(1988), memory was small and expensive, and the original design tried to use only one memory device, which resulted in many microinstructions per operation. Today, memory is large and inexpensive compared with the original multi-micro step URSIC design by Parhami, et al. [5]. For this reason, we made a couple of small modifications to reflect today's RISC/FPGA design principles:

1. Use 16 registers and not memory to implement the `dst-= src` instruction.
2. Initialize all registers to –1 at reset.
3. Use r[0] as input port (iport); r[1] keep at –1; r[15] as output port.
4. Allow monitoring of r[2] (temp), r[9] (loop-counter), valid flag (VF) and branch flag (BF). Reset the valid flag after the register is cleared.
5. Use only one pipeline register for a program counter (PC=Program memory address) and instruction word (IW).
6. Use single program memory (127 = $2^7$ words $\times$ 16 bits width), no data memory.
7. Allow a relative and absolute PC update.

The URISC models use 8 LISA 2.0 operations and can be designed in two phases: In the first phase a simple I/O processor (still synthesizable to FPGA) is designed. In the second step, the full functional processor is designed. We have implemented three software examples that use only the basic feature available in all university development boards—such as FPGA, LEDs and switches:

- The first program is a simple I/O test that reads data from the input DIP switch and displays the results on the LEDS of the FPGA board, see Fig 4.
- The second example computes are based on the DIP switch and the Fibonacci number based on the iterative procedure $F_k = F_{k-1} + F_{k-2}$.
- The third example is a factorial program. Depending on the input DIP switch, the factorial is computed. This is more challenging to the programmer since a multiplier must to be translated into a series of additions, since URISC does not include a multiplier.

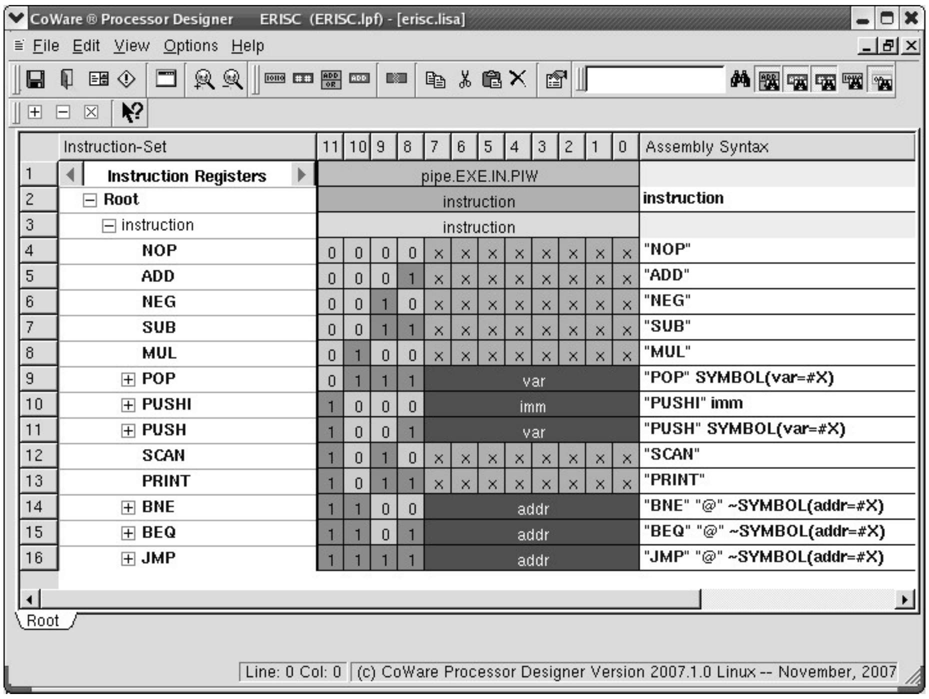
All three programs have been tested on the Altera and Xilinx development boards. Programming files can be downloaded from a companion web page [8]. Synthesis results for the URISC processors as examples are given in Table 2. As is evident, the designs are small and should fit on any popular university board system, even outdated systems like the Altera UP2. The speed data is sufficiently large enough for the on-board oscillator to be used directly to run the processors. No clock dividers are required.

### *4.3 ERISC processor model*

The ERISC processor model is an 8-bit basic µP that has allowed us to develop a full set of developmental tools, including a C compiler. An iterative refinement method can be used to develop the processor step-by-step and verify via a test program or HDL simulation. Five different versions were designed where the version number corresponds to the number of LISA 2.0 operations in use.

Table 2. Synthesis results for URSIC and ERISC processor models

| | **Altera DE2** | | | **Xilinx Nexys S200** | | |
|---|---|---|---|---|---|---|
| | Les | Multipliers | Speed | LUTs | Multipliers | Speed |
| **URISC** | 326 | 0 | 94.74 MHz | 284 | 0 | 78.9 MHz |
| **ERISC3** | 64 | 0 | 369.69 MHz | 58 | 0 | 234.7 MHz |
| **ERISC13** | 390 | 1 | 151.38 MHz | 347 | 1 | 96.4 MHz |



CoWare ® Processor Designer ERISC (ERISC.lpf) - [erisc.lisa]

| | Instruction-Set | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | Assembly Syntax |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | Instruction Registers | pipe.EXE.IN.PIW | | | | | | | | | | | | |
| 2 | Root | instruction | | | | | | | | | | | | instruction |
| 3 | instruction | instruction | | | | | | | | | | | | |
| 4 | NOP | 0 | 0 | 0 | 0 | x | x | x | x | x | x | x | x | "NOP" |
| 5 | ADD | 0 | 0 | 0 | 1 | x | x | x | x | x | x | x | x | "ADD" |
| 6 | NEG | 0 | 0 | 1 | 0 | x | x | x | x | x | x | x | x | "NEG" |
| 7 | SUB | 0 | 0 | 1 | 1 | x | x | x | x | x | x | x | x | "SUB" |
| 8 | MUL | 0 | 1 | 0 | 0 | x | x | x | x | x | x | x | x | "MUL" |
| 9 | POP | 0 | 1 | 1 | 1 | var | | | | | | | | "POP" SYMBOL(var=#X) |
| 10 | PUSHI | 1 | 0 | 0 | 0 | imm | | | | | | | | "PUSHI" imm |
| 11 | PUSH | 1 | 0 | 0 | 1 | var | | | | | | | | "PUSH" SYMBOL(var=#X) |
| 12 | SCAN | 1 | 0 | 1 | 0 | x | x | x | x | x | x | x | x | "SCAN" |
| 13 | PRINT | 1 | 0 | 1 | 1 | x | x | x | x | x | x | x | x | "PRINT" |
| 14 | BNE | 1 | 1 | 0 | 0 | addr | | | | | | | | "BNE" "@" ~SYMBOL(addr=#X) |
| 15 | BEQ | 1 | 1 | 0 | 1 | addr | | | | | | | | "BEQ" "@" ~SYMBOL(addr=#X) |
| 16 | JMP | 1 | 1 | 1 | 1 | addr | | | | | | | | "JMP" "@" ~SYMBOL(addr=#X) |

Line: 0 Col: 0 | (c) CoWare Processor Designer Version 2007.1.0 Linux -- November, 2007

Fig. 5. Instruction set of final ERISC version.

- Version 1: Very simple version with only one NOP operation and PM. Can generate: ASM, DIS, ISS. No HDL
- Version 3: HDL generation modifications:
  - `SCAN` and `PRINT` instruction to check I/O
  - Add 2 stage pipeline (design choice):
    1. stage load the instruction word
    2. Do the operation in this stage: Changes to all operations
  - Use a two instruction sequence for branch
- Version 8: Add the remaining ALU operations: `ADD`, `NEG`, `SUB`, `MUL`, and `PUSHI`
- Version 11: Add 3 instructions for loop control:
  - CONDITIONAL BRANCH (NOT) EQUAL: BEQ, BNE
  - Jump (JMP) unconditional
- Version 13: Add a data memory interface
  - PUSH on TOS and
  - POP TOS

Figure 5 shows a complete instruction set for version 13.

## 5. CONCLUSIONS

We have developed a balanced HW/SW teaching approach for embedded microprocessors to bridge the gap between the microprocessor specification and the software development tool by using architecture description language. We have provided two popular processor models that allow a step-by-step processor instruction set refinement and software development tools generation in each design step. Example designs have been synthesized for Xilinx and Altera development boards to provide hands-on lab experience. Three application programs have been developed for each processor, and all material is available online at http://www.eng.fsu.edu/~umb

Although we have developed teaching modules for the LISA language, it is hoped that in the future similar teaching modules will be developed and made available at no charge for other popular ADLs such as nML, ASIP Meister, EXPRESSION, MADL or MAML.

*Acknowledgments*—The authors gratefully acknowledge support for this work from the Council on Research and Creativity (CRC) at Florida State University and the Humboldt research foundation. The authors would like to thank Xilinx Inc, Altera Inc. and CoWare Inc. for their support under the University programmes. Any opinions, findings and conclusions or recommendations expressed in this paper are those of the authors, and do not necessarily reflect the views of the sponsors. Many thanks also to Carissa Neff for the English editing of this article.

## REFERENCES

1. M. Anderson, Help wanted: embedded engineers: why the United States is losing its edge in embedded systems *IEEE USA Today's Engineer Digest*, Mar. 2008.
2. R. Charette, This Car Runs on Code, *IEEE Spectrum Online*, Feb. 2009.
3. P. Ienne, R. Leupers, *Customizable Embedded Processors*, Morgan Kaufmann Publishers, Amsterdam, 2007.

4. P. Mishra, N. Dutt, *Processor Description Languages*, Morgan Kaufmann Publishers, Amsterdam, 2008.
5. F. Mavaddat, B. Parhami, URSIC The ultimate reduced instruction set computer*, Int. J. Eng. Educ.*, **25**, 1988, pp. 327–334.
6. B. Parhami, *Computer Architecture: From Microprocessor to Supercomputers*, Oxford University Press, New York, 2005.
7. A. Aho, R. Sethi, J. Ullman: *Compilers: Principles, Techniques, and Tools*, Addison Wesley Longman, Reading, Massachusetts, 1988.
8. U. Meyer-Baese, *LISA online resource* http://www.eng.fsu.edu/~umb/lisa (Accessed 21 December 2009).
9. A. Hoffmann, H. Meyr, R. Leupers, *Architecture Exploration for Embedded Processors with LISA*, Kluwer Academic Publishers, Boston, 2002.
10. R. Zurawski (ed.), A *Novel Methodology for the Design of Application-Specific Instruction-Set Processors*, Embedded System Handbook, CRC, Boca Raton, FL, 2006.
11. A. Hoffmann, H. Meyr, R. Leupers, *Optimized ASIP Synthesis from Architecture Description Language Models*, Springer, Dordrecht, The Netherlands, 2007.
12. U. Meyer-Baese, Digital *Signal Processing with Field Programmable Gate Arrays*, 3nd ed., Springer-Verlag, Berlin, 2007.
13. Z. Navabi, *Embedded Core Design with FPGAs*, 2nd ed., McGraw-Hill, New York, 2007.
14. F. Vahid, T. Givargis, *Embedded System Design*, John Wiley & Sons, Inc., Hoboken, New Jersey, 2002.
15. J. Armstrong, F. Gray, *VHDL Design Representation and Synthesis*, Prentice Hall, Upper Saddle River, NJ, 2000.
16. S. Brown, Z. Vranesic, *Fundamentals of Digital Logic with VHDL Design*, 2nd ed., McGraw-Hill, New York, 2005.
17. J. Catsoulis, C. John, *Designing Embedded Hardware*, O'Reilly Media, Beijing, 2005.
18. D. Gajski, *Principles of Digital Design*, Prentice Hall, Upper Saddle River, NJ, 1997.
19. W. Lee, *VHDL Coding and Logic Synthesis with SYNOPSYS*, Academic Press, London, UK, 2000.
20. S. Lee, *Advanced Digital Logic Design*, Thomson, Ontario, Canada, 2006.
21. M. Mano, C. Kime, *Logic and Computer Design Fundamentals*, Pearson Prentice Hall, Upper Saddle River, NJ, 2008.
22. A. Marcovitz, *Introduction to Logic and Computer Design*, McGraw-Hill, New York, 2008.
23. M. Bates, *Programming 8-bit PIC Microcontrollers in C: with Interactive Hardware Simulation*, Newnes, Elsevier, Oxford, UK, 2008.
24. R. Barnett, S. Cox, L. O'Cull, *Embedded C Programming and the Atmel AVR*, Delmar Cengage Learning, 2nd. ed. 2006.
25. M. Barr, A. Massa, *Programming Embedded Systems: With C and GNU Development Tools*, O'Reilly Media, Beijing, 2006.
26. S. Barrett, D. Pack, Embedded *Systems: Design and Applications with the 68HC12 and HCS12*, Pearson Prentice Hall, Upper Saddle River, NJ, 2005.
27. F. Cady, *Microcontrollers and Microcomputers: Principles of Software and Hardware Engineering*, Oxford University Press, Oxford, NY, 1997.
28. F. Cady, J. Sibigtroth, *Software and Hardware Engineering: Motorola M68HC12*, Oxford University Press, Oxford, NY, 2000.
29. C. Hallinan, *Embedded Linux Primer: A Practical Real-World Approach*, Prentice Hall PTR, Upper Saddle River, NJ, 2006.
30. C. Hellebuyck, *Beginner's Guide To Embedded C Programming: Using The PIC Microcontroller And The Hitech Picc-Lite C Compiler*, Create Space, Milford, MI, 2008.
31. D. Simon, *An Embedded Software Primer* Addison-Wesley, Reading, MA, 1999.
32. M. Samek, *Practical UML Statecharts* in C/C++, 2nd, ed., Event-Driven Programming for Embedded Systems, Newnes, Elsevier, Oxford, UK, 2008.
33. W. Wolf, *Computers as Components*, Academic Press, San Diego, CA, 2000.
34. K. Yaghmour, J. Masters, *Building Embedded Linux Systems*, O'Reilly Media, Bejing, 2008.

**Uwe Meyer-Baese** received the B.S.E.E., M.S.E.E., and Ph.D. (summa cum laude) degrees from the Darmstadt University of Technology, Darmstadt, Germany, in 1987, 1989 and 1995, respectively. He is currently a Professor in the Electrical and Computer Engineering Department, Florida State University, Tallahassee. In 1994 and 1995, he held a Post-doctoral Position in the Institute of Brain Research, Magdeburg, Germany. In 1996 and 1997, he was a Visiting Professor at the University of Florida, Gainesville. From 1998 to 2000, he worked as a Research Scientist in the ASIC industry, where he was responsible for development of high-performance architectures for digital signal processing. During his graduate studies, he worked part-time for TEMIC, Siemens, Bosch and Blaupunkt. He holds three patents, has published over 80 journal and conference papers and has supervised more than 60 master thesis projects in the DSP/FPGA area. He is author of the best selling Springer textbook on DSP with FPGAs. He was a recipient of the Habilitation (*venia legendi*) by the Darmstadt University of Technology in 2003, the Max-Kade Award in Neuroengineering in 1997, the Humboldt Research Award in 2006 and a FAMU-FSU College of Engineering Teaching Award 2007.

**Guillermo Botella** received the M.A. Sc. degree in Physics in 1998, the M.Sc. degree in Electronic Engineering in 2001 and the Ph.D. degree in 2007, all from the University of Granada. From 2002 to 2005 he was a research European Fellow at the Department of

Architecture and Computer Technology of the Universidad de Granada and the Vision Research Laboratory at University College London. After that he joined as Assistant Professor at the Department of Computer Architecture and Automation of Complutense University of Madrid. He has been visiting professor in 2008 and 2009 at the Department of Electrical and Computer Engineering, Florida State University, Tallahassee. He has authored more than 20 technical papers in international journals and conferences. His current research focuses on image, video and signal processing on FPGAs, vision algorithms and design automation for High Level Synthesis.

**Encarnación Castillo** received the M.A.Sc. degree and Ph.D. degree in electronic engineering from the University of Granada, Spain, in 2002 and 2008, respectively. From 2003 to 2005 she was a research Fellow at the Department of Electronics and Computer Technology at the University of Granada, where she is now an Assistant Professor. During her research fellowship, she carried out part of her work at the Department of Electrical and Computer Engineering, Florida State University, Tallahassee. Her research interests include the protection of IP protection of VLSI and FPGAs-based systems, as well as Residue Number System arithmetic, high-performance digital signal processing and VLSI and FPL signal processing systems. She has authored more than 20 technical papers in international journals and conferences.

**Antonio García** received the M.A.Sc. degree in Electronic Engineering (obtaining the Nation Best Academic Record award) in 1995, the M.Sc. degree in Physics (majoring in Electronics) in 1997 and the Ph.D. degree in Electronic Engineering in 1999, all from the University of Granada. He was an Associate Professor at the Department of Computer Engineering of the Universidad Autónoma de Madrid before joining the Department of Electronics and Computer Technology at the University of Granada, where he serves as Professor. He has authored more than 70 technical papers in international journals and conferences and serves regularly as reviewer for several IEEE and IEE journals. He is also part of the Program Committee for several international conferences on programmable logic. His current research interests include IP protection of VLSI and FPGA-based systems, low-power and high-performance VLSI signal processing systems and the combination of digital and analogue programmable technologies for smart instrumentation. He is a member of IEEE and a C&S and SP Society member.