Architectural Specification of a Microcontroller by Using ArchC

Maicon Carlos Pereira, Cesar Albenes Zeferino
{maicon, zeferino}@univali.br

Universidade do Vale do Itajaí
Centro de Ciências Tecnológicas da Terra e do Mar
Grupo de Sistemas Embarcados e Distribuídos

Abstract

In this paper, it is presented the architectural specification of a basic microcontroller, named µBIP, developed to be used in the teaching of concepts related to the design of embedded systems, at the software and the hardware levels. The microcontroller architecture is derived from a family of basic processors (BIP – Basic Instruction-set Processor) specially designed focusing on students' learning. In this paper, it is described the architecture of µBIP, which was specified by using ArchC, an architecture description language based on SystemC.

1. Introduction

The increasing demand for products based on embedded systems has carried out to the need for engineers able to work on the different phases of the design of such computing systems. However, some of the basic concepts in this field have not been discussed in most of undergraduate courses in Computer Science [HEN 07].

Computer Architecture and Organization disciplines play a major role in the development of abilities and skills of students of Computer Science and Computer Engineering courses, especially for the ones which will work on the design of embedded systems. However, many students have difficulties in understanding the first concepts taught in the introductory disciplines in this domain. To minimize this problem, teachers continuously work in the search of new methods to help the students in understanding such concepts, by making a link with other disciplines related to programming, compilers, and operating systems [MOR 06].

In this way, researchers of the Embedded and Distributed Systems Group (GSED) of University of Vale do Itajaí (Univali) made a specification of a processor family named BIP – Basic Instruction-set Processor. This family aims at facilitating the understanding of introductory concepts on Computer Architecture and Organization in the first terms of an undergraduate course. A second goal of BIP family is to promote an interdisciplinary approach, allowing teachers of other disciplines to use BIP specification in practical activities.

Current BIP specification is based on the simplest architectural alternatives in order to ease the programming and the building of a processor: (*i*) it is an accumulator-based architecture; (*ii*) all the instructions are based on a single instruction format; and (*ii*) there are only two addressing modes (Immediate and Direct). The first version of BIP (BIP I) is an "as simple as possible" architecture, with a very small Instruction Set Architecture (ISA), including only load, store and arithmetic instructions (add and subtract). The second version (BIP II) adds conditional and unconditional branch instructions.

BIP I and BIP II were already applied in several disciplines in undergraduate courses in the Computer Science field, allowing the students to improve their comprehension about several concepts. However, due to their simplicity, the applicability of these CPUs shown to be limited to be used in more advanced disciplines, because of the lack of several features of real processors applied in embedded systems, like I/O ports, timers, and other peripherals.

This paper presents results of an ongoing project that aims at the design of a simplified microcontroller named µBIP (micro BIP). This microcontroller is based on the BIP II specification and extends its architecture by adding new instructions. It also includes support for the integration of peripherals to the CPU. In this paper, it is described the architectural specification, which was done with the aid of ArchC, an Architecture Description Language – ADL developed at the Instituto de Computação of the University of Campinas (IC-UNICAMP) [ARC 07a].

This text is organized in seven sections. Section 2 presents some concepts on ArchC, while Section 3 describes the µBIP architecture. The ArchC µBIP model is presented is Section 4, followed by the description of the method used to validate the specified architecture, in Section 5. Concluding, in Section 6, is done a briefly discussion about the use of µBIP in Education, and, in Section 7, they are presented the final remarks.

2. ArchC

An ADL is a computer language used to describe software and/or system architectures. As defined by [BAL 05], given a description of a processor architecture, an ADL allows to automatically generate software tools (ISA simulator, compiler, assembler, linker and debugger). ArchC is an ADL which allows describing the processor architecture and the memory hierarchy. In its current version (v.2.0), it also allows generating ISA simulator, linker and debugger tools.

The description of an architecture using ArchC is composed by two main blocks [ARC 07b]: Architecture Resources (AC_ARCH) and Instruction Set Architecture (AC_ISA). The first one includes the description of the amount and type of memory, the amount of registers and the pipeline. The second one includes information about the instruction set, as: (*i*) instruction width; (*ii*) instruction format; (*iii*) opcodes; (*iv*) decoding; and (*v*) instruction behavior.

The modeling of a processor in ArchC can be done at different levels of abstraction. At the functional level, a model includes only a few information, but they allow generating tools to analyze the instruction set details about timing or pipeline operation. At the cycle-accurate level, it is possible to build a model closer to a realistic implementation, including information about timing and pipeline.

3. µBIP Architecture

In µBIP microcontroller, instructions and data are 16-bit wide. There is only one instruction format, shown in Fig. 1. This instruction format considers one implicit operand, the ACC (accumulator) register, and one explicit operand (the *operand* field), which can be a constant (in immediate addressing mode), a variable (in direct addressing mode) or a vector index (in indirect addressing mode).

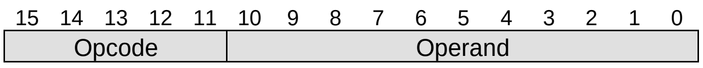

Fig. 1 – Instruction format

The instruction set includes 29 instructions (BIP II had only 15), which are organized in the nine classes shown in Tab. 1.

Tab. 1 – Instruction-set

Class	**Instructions**
Store	`STO`
Load	`LD, LDI`
Arithmetic	`ADD, ADDI, SUB, SUBI`
Logical	`AND, OR, XOR, ANDI, ORI, XORI, NOT`
Control	`HLT`
Branch	`BEQ, BNE, BGT, BGE, BLT, BLE, JMP`
Logical Shift	`SLL, SRL`
Vector access	`STOV, LDV`
Procedures	`RET, RETI, CALL`

µBIP has Harvard architecture, program and data memories. Address space is organized as a 2-Kword program space and a 2-Kword data space. I/O is memory mapped and occupies half of the data memory space.

The first version of µBIP uses a mono-cycle organization and includes the following peripheral and hardware features: (a) two 16-bit I/O ports with individual direction control for each pin; (b) a 16-bit timer; (c) an interrupt controller; and (d) hardware support for procedure calls.

4. Modeling µBIP with ArchC

Since µBIP uses a mono-cycle organization, functional and cycle-accurate models in ArchC are equivalent. Its ArchC description is composed by the following files: (a) *ubip.ac*; (b) *ubip_isa.ac*; (c) *ubip_isa.cpp*; and (d) *ubip_address.h*. The first file (*ubip.ac* file) describes the architecture resources: memories, registers, word size and endianness (see Fig. 2).

```
AC_ARCH(ubip){
  ac_mem memRAM:4K; //4Kbyte/2K-Words
  ac_mem memROM:4K; //4Kbyte/2K-Words
  ac_reg PC, ACC, STATUS;
  ac_wordsize 16;
  ARCH_CTOR(ubip) {
    ac_isa("ubip_isa.ac");
    set_endian("big");
  };
};
```

Fig. 2 – Description of architecture resources (*ubip.ac*)

The second file (*ubip_isa.ac*), partially shown in Fig. 3, is where the instruction format and its fields (*op* and *operand*) are specified. Following, the instructions set is declared and it is defined a map (*ac_asm_map*) describing the allowed values for the instruction operands. Two functions are used to define the syntax (*set_asm*) and the opcode (*set_decoder*) of a given instruction. Another function (*pseudo_instr*) allows defining pseudo-instruction which, at the assembling process, is replaced by a sequence of native instructions.

```
AC_ISA(ubip){
  ac_format INS_FORMAT  = "%op:5 %operand:11";

  ac_instr<INS_FORMAT> add, addi, sub, subi;

  ac_asm_map ubipSFR {
    "$"[0..2047]  = [0..2047];
    "$port0_dir"  = 1024;
    "$port0_data" = 1025;
    "$tmr0_config"= 1040;
    ...
  }
  ISA_CTOR(ubip){
    //--- ASSEMBLY ---
    add.set_asm("add %ubipSFR", operand);
    add.set_asm("add %exp", operand);
    add.set_decoder(op=0x04);
    ...
    //Pseudo-Instructions
    pseudo_instr("psto %ubipSFR, %imm"){
      "ldi %1";
      "sto %0";
    }
    ...
  };
};
```

Fig. 3 – Description of Instruction-Set Architecture (*ubip_isa.ac*)

The third file (*ubip_isa.cpp*) describes the behavior of instructions and peripherals. Fig. 4 and Fig. 5 present part of this file, where they are shown the behavior of *ADD* instruction (*behaviour*(*add*)) and part of the common behavior for all the instructions (*behavior*(*instruction*)). Two functions (*DataMemoryRead* and *DataMemoryWrite*) were implemented to deal with word addressing, since ArchC addressing is byte-oriented. The same problem had to be solved in the functions related with PC register (*inc_PC*, *set_PC* and *get_PC*). A number of constants used in *ubip_isa.cpp* are defined in the last of the four files which compose the ArchC µBIP model.

```
void ac_behavior( add ){
  ac_word operand1 = ACC.read();                        //Fetching operand 1
  ac_word operand2 = DataMemoryRead(memRAM,operand);    //Fetching operand 1
  ac_word result   = operand1 + operand2;               //Executing the operation
  ACC.write(result);                                    //Writtng result into ACC
  set_status(STATUS, Z, check_zero(result));            //Updating STATUS flags
  set_status(STATUS, N, check_negative(result));
  set_status(STATUS, C, check_carry_out(operand1, operand2));
  inc_PC(ac_pc);                                        //Updating PC
}
```

Fig. 4 – Behavior of ADD instruction (*ubip_isa.cpp*)

```
// For all the instruction do:
void ac_behavior( instruction ){
   //Check for interrupt requests on PORT0 and on Timer0
   bool bIntr = false;
   bIntr = bIntr | CheckInterruptPA0(memRAM, ac_pc, tempMem[PORT0_DATA],
                              DataMemoryRead(memRAM,PORT0_DATA));
   bIntr = bIntr | (trm00_inc(memRAM, ac_pc, ac_instr_counter)); //Update Timer0
   if (bIntr) ac_annul();
   (...)
}
```

Fig. 5 – Common behavior for all the instructions (*ubip_isa.cpp*)

5. Validation

For the validation of the developed ArchC model, it was defined a test plan which were applied by using software tools (ISA simulator, assembler and linker) automatically generated by ArchC. This test plan included a set of unitary tests for each instruction, which verified the correctness of these instructions regarding the operation to be executed and the state of registers and data memory, before and after the instruction execution.

The validation process will be extended with the use of test bench codes based on the benchmarks available in the Dalton Project site [DAL 01].

6. Discussion

While BIP I and BIP II were specified to be used in introductory classes on Computer Architecture, µBIP is intended to be applied in courses focusing in Digital System Design and Embedded System Design. We envision that it could become a useful tool to aid students in learning issues related to these disciplines.

Advanced studies can be carried out, like, for instance, adding new peripheral or implementing µBIP in a pipeline-based organization.

7. Conclusions

In this paper, it was presented some results of an ongoing project which aims at the development of a basic microcontroller to be used in Education, especially in advanced disciplines of undergraduate courses in Computer Science and Computer Engineering. But it could also be used in graduate courses.

In the first phase of this development, ArchC ADL was used to aid the architecture specification and validation. The software tools automatically generated by ArchC were very useful in this process. In the next phase, it will be implemented the VHDL model for the monocycle organization of µBIP, which will be synthesized and validated in FPGA.

8. References

[ARC 07a] THE ARCHC TEAM. **The ArchC architecture description language v2.0: reference manual**. Campinas: The ArchC Team, 2007.

[ARC 07b] THE ARCHC TEAM. **The ArchC project home page**. Disponível em: <http://www.archc.org>.

[BAL 05] BALDASSIN, Alexandro. **Geração automática de montadores em ArchC**. 2005. Dissertação (Mestrado em Ciência da Computação)-Programa de Pós-Graduação em Ciência da Computação, Universidade Estadual de Campinas, Campinas, 2005.

[DAL 01] THE DALTON PROJECT TEAM. **The UCR Dalton Project**. University of California – Riverside, 2001. Disponível em: < http://www.cs.ucr.edu/~dalton/ >.

[HEN 07] HENZINGER, Thomas A; SIFAKIS, Joseph. **The Discipline of Embedded Systems Design**. Computer. v. 40, n.10, p. 32-40. out. 2007.

[MOR 06] MORANDI, Diana; PEREIRA, Maicon Carlos; RAABE, André Luis Alice; ZEFERINO, Cesar Albenes . **Um processador básico para o ensino de conceitos de arquitetura e organização de computadores**. Hífen, Uruguaiana, v. 30, p. 73-80, 2006.