FORMAL MODELLING OF THE ADSP-2100 PROCESSOR USING HOL

Ali Habibi[1], Sofiène Tahar[1] and Adel Ghazel[2]
[1]*Concordia University, ECE Dept., Montreal, Quebec, H3G 1M8 Canada*
[2]*Ecole Supérieure des Communications de Tunis, 2083 Ariana, Tunisia*
[1]{habibi,tahar}@ece.concordia.ca
[2]{adel.ghazel}@supcom.rnu.tn

Abstract

In this paper, we describe formal modelling of the digital signal processors of the family ADSP-2100 using the HOL (Higher Order Logic) theorem prover. While specifying the behavior and implementation of the processor, we solved the problem of complexity related to the large number of parameters by using a structured method based on our knowledge about the processor architecture. We show details of the specification strategy used and display few illustrative examples.

1. INTRODUCTION

Hardware and software systems are growing everyday in scale and functionality. This increase in complexity increases the number of subtle errors. Moreover, some of these errors may cause catastrophic loss of money, time, or even in many cases human life. A major goal of system design is to enable developers to construct systems that operate reliably despite this complexity. One way of achieving this goal is by using formal methods, which are mathematically-based languages, techniques, and tools for specifying and verifying such systems [9].

Only recently have we begun to see a more promising picture for the future of formal methods. For hardware verification, industry is adopting techniques like model checking and theorem proving to complement the more traditional one of simulation. Researchers and practitioners are performing more and more industrial-sized case studies, and thereby gaining the benefits of using formal methods. Actually there are many tools performing hardware verification. Principally the two approaches model checking and theorem proving are getting more interest.

Notable examples about using formal methods for processors specification and verification are described in the literature. The most related to our study is the Motorola CAP [5]. During 1992-1996 Brock of Computational Logic, Inc., working in collaboration with Motorola designers, developed an ACL2 specification of the entire Motorola Complex Arithmetic Processor (CAP), a microprocessor for digital signal processing (DSP). The CAP is one of the most complicated microprocessor yet formalized (with instruction set allowing the simultaneous modification of well over 100 registers in a single instruction). The formal specification tracked the evolving design and included a simpler non-pipelined view that was proved equivalent on a certain class of programs [4].

The verification of the processor ADSP-2100 is very similar in complexity to the Motorola CAP. However, our objective is to specify and later verify the full instruction set of the processor using the theorem prover HOL [6] . The ADSP-2100 family is a collection of programmable single-chip microprocessors that share a common base architecture optimized for digital signal processing (DSP) and other high-speed numeric processing applications. The various family processors differ principally in the type of on-chip peripherals they add to the base architecture. On-chip memory, a timer, serial port(s), and parallel ports are available in different members of the family.

The HOL system is an interactive environment for machine-assisted theorem-proving in higher-order logic [6]. The HOL logic is simple but expressive, incorporating higher-order functions and Milner-style polymorphism [3]. HOL uses the programming language ML [11] to program proof strategies. Theorems are encoded via an abstract type whose only constructors are the primitive inference rules of the logic. The strength of HOL comes from two principles proprieties. First backward (goal-directed) proof is supported, and may be freely mixed with forward proof. Second, adherence to definitional extension guarantees that the consistency of the logic is not compromised [1].

The rest of the paper is organized as follows. In Section 2, we describe the ADSP-2100 processor architecture. In

Proceedings of the 2002 IEEE Canadian Conference
on Electrical & Computer Engineering
0-7802-xxxx-x/02/$10 © 2002 IEEE

Section 3, we present the methodology used in the specification process. This latter will be detailed in Section 4. Finally, in Section 5, we outline the general conclusions and point out to future work.

2. THE ADSP-2100 FAMILY

2.1 Basic Units

The processors of the ADSP-2100 has three separated computational units: Arithmetic and Logic Unit (ALU), Multiplier/Accumulator (MAC) and the Barrel Shifter (Figure 1). These units treat 16 bits fixed point data [3].

The two Address Generators and the Program Sequencer are responsible of the addressing management. The use of two address generators allow the execution of two data reading (fetch) from the external memory. The program sequencer has some embedded units allowing the execution of internal loops which increases the efficiency of the processor [12].

The principal characteristic of this family of processors is that it is based on a modified Harvard architecture [8]. In fact, data is stored in both the data and the program memories. This is very important because it allows the execution of two instructions in the same clock cycle.

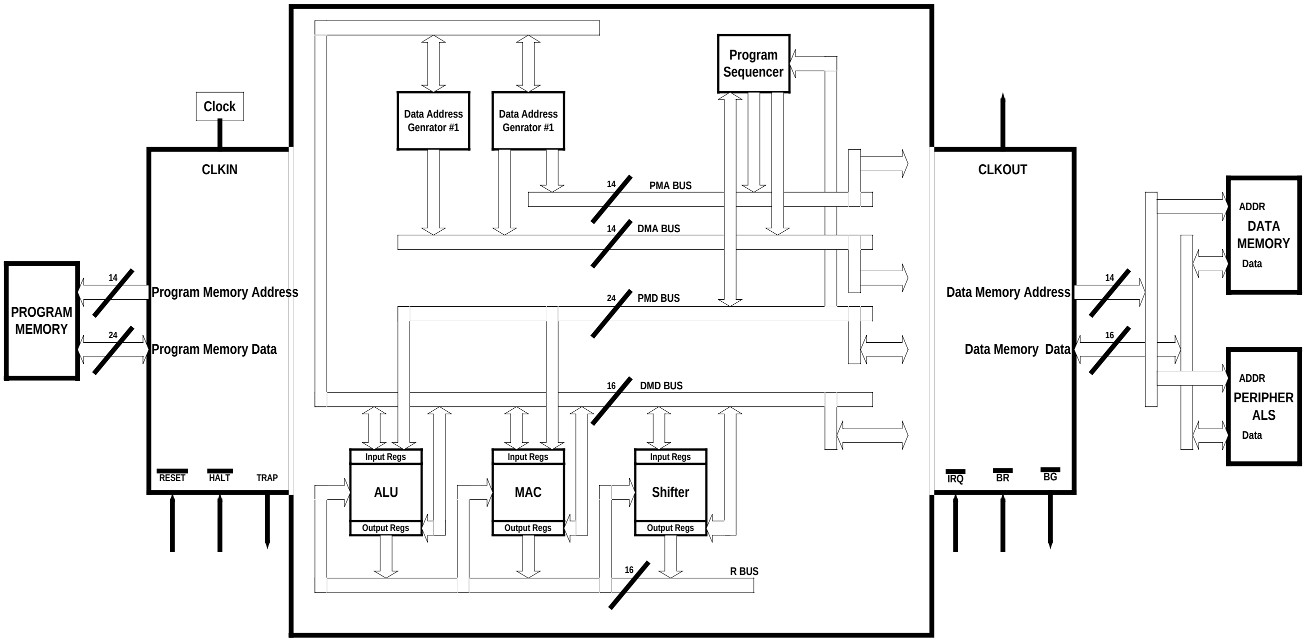


Fig. 1. ADSP 2100 Processor Architecture [2].

2.2 Instruction Set of the ADSP-2100 Family

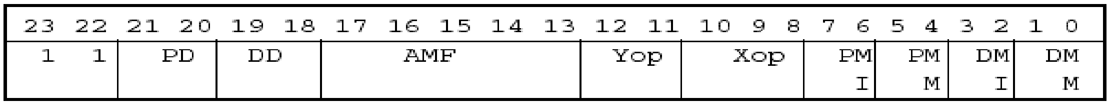
The instruction set of the ADSP-2100 family are characterized by the classification of the instructions and the multiplicity of instructions per command [2]. In fact, the instructions are classified by the concerned unit. In other words, the instructions of the ALU have the same format that is different form that of the program sequencer for example. On the other hand, the majority of the instructions are composed of two parts: the first is a memory access (for reading or writing) and the second is an operand execution (ADD operand for example).

The general format of the instructions is given in Figure 2. It is composed of two principal parts: the identifier of the instruction and its parameters. An example of an instruction is given in Figure 3.

General Format:

Opcode Identifier	Opcode Parameters

Fig. 2. General format of the instructions of the ADSP-2100 family [2].



23 22	21 20	19 18	17 16 15 14 13	12 11	10 9 8	7 6	5 4	3 2	1 0
1 1	PD	DD	AMF	Yop	Xop	PM I	PM M	DM I	DM M

Fig. 3. ALU instruction with data and program memory read [2].

3. GENERAL METHODOLOGY

We propose to define a methodology for the specification and verification of the ADSP-2100 processor. This methodology will satisfy certain characteristics to guarantee its capability to deal with the processor features, to take advantage from the processor architecture and to prepare the proof goal in the simplest way to the HOL system. In fact, the complexity of a DSP processor in terms of number of variables, variable types and the variety of instrucions makes it impossible the use of the direct way [5].

The methodology that we defined includes four principle steps:

(1) Simplification of the processor units specifications.

(2) Construction of the specification of the implementation of the processor.

(3) Writing the specification of the behavior of the processor.

(4) Making the proof of the goal "*Processor Implementation ⇒ Processor Specification*" for every instruction.

In this paper, we will focus on the first three specification steps. The verification step is described elsewhere [7]

4. FORMAL SPECIFICATION

In our case the formal specification concerns two parts: the specification of the hardware and the specification of the behavior. The final goal is to simplify the proofs as much as possible. For this reason, we started by specifying the units of the processor. Then we wrote a functional specification of these units. Our first task was then to make the proof of the following implication: "*hardware representation⇒ functional representation for every unit*."

This will allow us to manipulate simpler representations of the units during the verification phase. In fact, this will give us a high level representation of the units omitting the internal signals and therefore minimizing the number of parameters.

4.1 Components Specification

The first step in the specification of the processor is to specify the basic components that will serve in the construction of the specification of the units of the processor. The basic components that we specified are the multiplexers, registers, buses and memories.

The following specification presents the case of one multiplexer which has two inputs the first of type *Word8* (8 bits) and the second of type *Word16* (16 bits), and one output of type *Word8* (8 bits). The selection between the two inputs is done according to the value of the boolean parameter *input_select* which, when set to true (T) gives the first input to the output and when set to false (F) gives the second input to the output:

$$\begin{aligned}&\text{MUX\_8\_16\_8}(input1, input2, output, inptSelect) = \\ &\quad \forall t.\quad ((inputSelect(t) = T) \wedge (output(t) = input1(t))) \\ &\qquad \vee\ ((inputSelect(t) = F) \wedge (output(t) = input2(t)))\end{aligned}$$

4.2 Iterative Units Specification

For every unit of the processor, we also defined a behavioral specification and a hardware specification of the unit. This is done to simplify the instruction verification task by manipulating a simple representation of the processor. Next, we use the ALU unit as example to describe our approach.

The ALU is the unit responsible for the arithmetic and logic operations. It is composed of a group of input registers, a group of output registers, many multiplexers and one base unit making the arithmetic and logic operations.

In the functional specification of the ALU we tried to represent this unit in a simple and optimized way. Our representation is composed of three parts: the selection of the inputs, the selection of the appropriate function and the storage of the result.

An important remark is that all the types used within the ALU are the same (*Word16*: 16 bits in this case). Therefore, it is possible to use a single abstract type to represent all parameters of this unit. This is very important because it allows us to do generic proofs that are independent from the sizes of the buses or the registers. This way our proofs are true even for systems considering data types either than *Word16*.

4.2.1. Functional Specification of the ALU. In the functional specification of the ALU we tried to represent this unit in a simple and optimized way. Our representation is composed of three parts: the selection of the inputs, the selection of the appropriate function and the storage of the result.

Inputs Identification: The inputs of the base unit of the ALU depend on the multiplexers at the input of the registers *AX* and *AY* and on the two inputs *X* and *Y* (see Figure 4). In our representation we selected the value of input of this unit directly from the values of control of the multiplexers. These commands are in our case: *ALUMUXXinput*, *ALUMUXYinput*, *AXinputselect* and *AYinputselect*.

Selection of the Appropriate Function: The function that will be executed by the ALU comes from the instruction by means of the Program Sequencer. The parameter *FunctionCode* identifies the actual operation. This parameter is read from the **AMF** field in case of the ALU instructions (see Figure 3).

Data Storage: At the end of the operation of the ALU, the result has to be stored either in the register *AF* only or in both registers *AF* and *AR*. This result can be either written in the program or data memories by means of the R bus.

4.2.2. ALU Specification Construction. The ALU is formed by many components provided in a multi-pages hardware description. If we consider that the processor core contains 8 other units more complex than the ALU, we can easily deduce that the problem of verification of the instruction set will be impossible if we will not simplify the units specifications. In fact, we will deal with thousands of parameters and a very long specification.

Table 1 describes the steps that we used to make the proof of the implication between the hardware specification of the ALU and its behavioral specification. In the first step, we considered a sub-system of the ALU containing the base unit of the arithmetic and logic operations and the selection of the first input Y. Then in the second step, we considered the registers AY and AR. By adding to each iteration a new set of components we ended by representing the full ALU unit.

The ALU specification construction is detailed in Figure 4. We start by specifying the basic ALU unit only. Then progressively we add the other components (registers, multiplexers...) one by one. In each iteration we eliminate some internal signals from the ALU specification. For example, in the "iteration 2" the new specification of the ALU will not contain any connection between the register AF and the basic ALU unit. This register will be represented as a parameter of the new specification and not as a component. In the final iteration, we will get the functional specification of the ALU as a single box. Therefore, it is proved that the ALU can be specified as a short yet simple unit that can be easily manipulating while constructing the full processor specification and during the proofs of the full instruction set.

Table 1. Steps of the construction of the ALU specification.

Step	**Proof**		
	Hardware Specification	⇒	*Behavioral Specification*
1	The multiplexer at the output of the register AY. The base unit of the ALU.	⇒	Sub-system 1 of the ALU.
2	Sub-system 1 of the ALU. The multiplexer at the input of the register AY. The register AY. The register AF.	⇒	Sub-system 2 of the ALU.
3	Sub-system 2 of the ALU. The multiplexer at the input of the register AX. The register AX.	⇒	Sub-system 3 of the ALU.
4	Sub-system 3 of the ALU. The multiplexer at the input of the register AR. The register AR.	⇒	Functional Specification of the ALU.

4.3 Processor Specification Construction

After simplifying all the processor units, we were able to construct the full processor specification. However, keeping in mind that the simpler is processor specification, the easier is the instruction set verification, we add another step to the specification process.

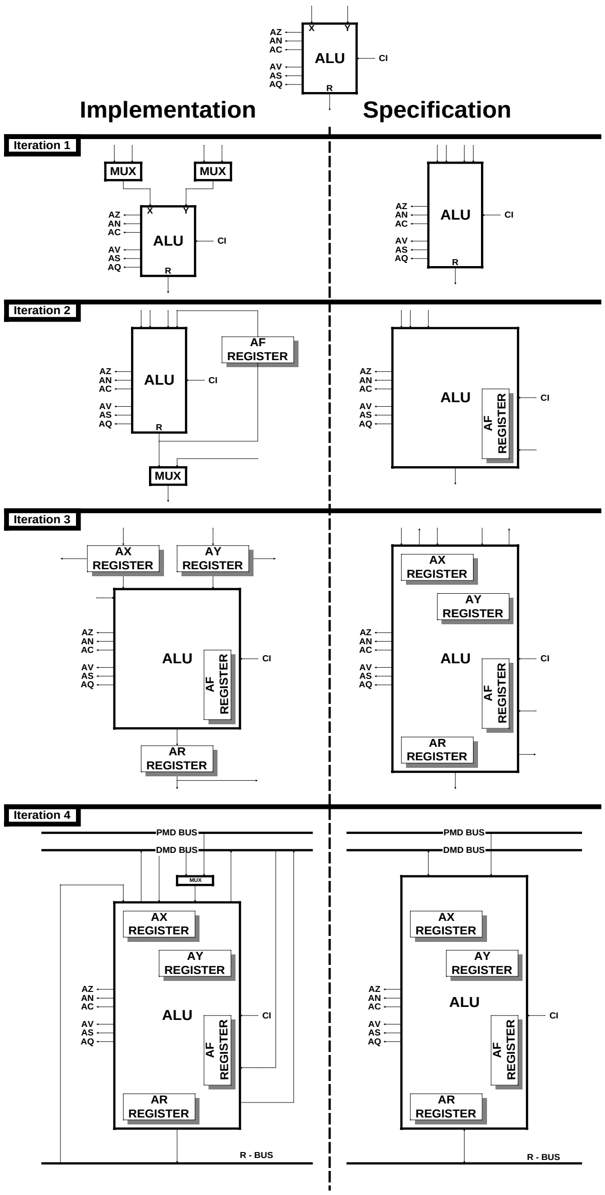


Fig. 4. ALU unit specification.

In place of a unique representation of the processor, we formed an instruction-dependent processor specification (Figure 5). In other words, the processor is described as a set of representations. Each of these representations is related to one of the opcodes. This way, we simplify the processor representation without modifying its behavior since each opcode concerns only a subset of units of the processor.

Global Representation of the Processor: This representation contains a full description of the processor. It contains all the inputs and outputs.

Instruction Specific Representation of the Processor*:* This representation takes into account the knowledge of the way the processor is working. In fact, we only take into account the units related the instruction being executed. For example, if an instruction concerns only the ALU, there is no need to add the specification of the MAC. This way, we are looking to the system differently depending on each instruction.

If we consider the example of the opcode represented in Figure 3, then the representation of the processor is the one given in Figure 6 for the case of the ALU operation with program and data memory read. In comparison to the global representation of the processor given previously in Figure 1, the new representation includes only the ALU, the Program Sequencer and the first Data Address Generator. So the internal size of the processor is reduced by the half since the MAC, the Shifter, the second address generator and the inter-bus transfer units are not included. We have to notice that this reduction of the size of the processor do not affect its behavior since the "ALU operation with data and program memory read" does not concern the other units.

Currently, the construction of the instruction specific representation of the processor is done manually. However, we plan to automate this procedure in future work. In fact, the construction of this specification requires only the definition of the table of correspondence between the instructions and the units that are involved in the instruction.

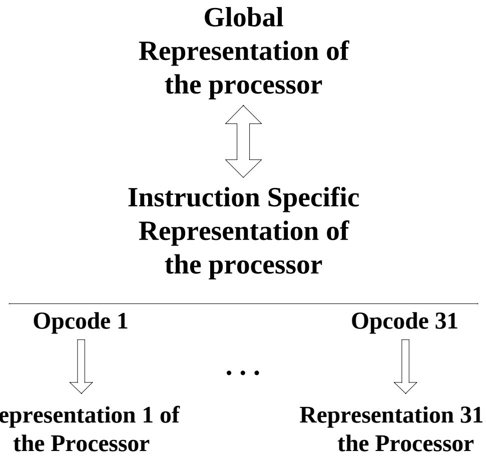

Global Representation of the processor
Instruction Specific Representation of the processor
Opcode 1
Opcode 31
. . .
Representation 1 of the Processor
Representation 31 of the Processor

Fig. 5. Simplification of the processor specification.

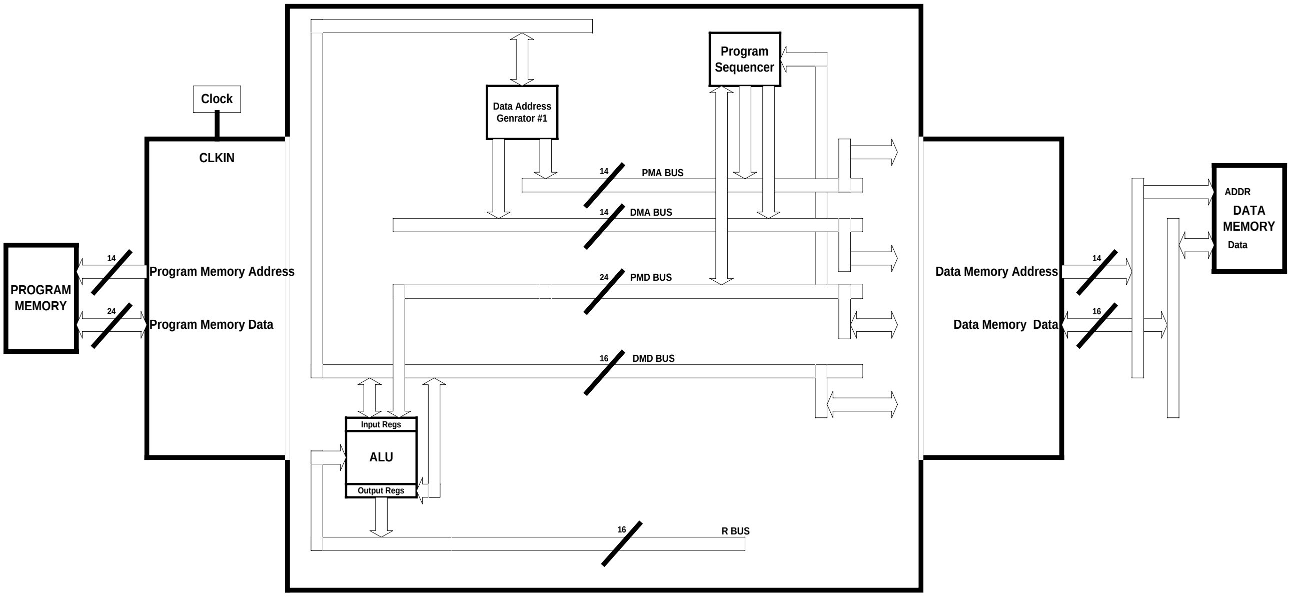


Fig. 6. Instruction specific representation of the processor.

5. CONCLUDING REMARKS

In this paper, we investigated the formal specification of the DSP ADSP-2100 processor using the HOL theorem prover. The main contributions of this work are: (1) the specification of both the behavioral and structural representation of a commercial DSP chip; (2) the specification of any other DSP processor will be a matter of adapting and reusing the specification that we made; and (3) the behavioral and the structural specification of the processor are reduced enough to allow the verification of the full instruction set. For instance, without making a simplification of the units specifications, it will be impossible to make the verification of the ADSP-2100 processor using HOL. In fact, the specification of such a complex system took more than 100 pages.

References

[1] P. Andrews. “*An Introduction to Higher Order Logic: To Truth through Proof*,” Academic Press, New York, 1986.

[2] Applications Engineering Staff of Analog Devices, DSP Division. “*Digital Signal Processing Applications Using the ADSP-2100 Family*,” Prentice Hall, Englewood Cliffs, NJ 07632, 1996.

[3] G. Birtwistle, B. Graham, and S.- K. Chin, “*new_ theory ‘HOL‘; An Introduction to Hardware Verification in Higher Order Logic*,” August 1994.

[4] B. Brock, M. Kaufman, and J.S. Moore, “Heavy inference: Theorems about commercial microprocessors,” *Formal Methods in Computer-Aided Design*, Springer-Verlag, November 1996.

[5] E. M. Clarke and J. M. Wing. “Formal Methods: State of the Art and Future Directions,” *ACM Computing Surveys*, December 1996.

[6] M. Gordon, and T. Melham, “*Introduction to HOL: A theorem Proving Environment for Higher-Order Logic*,” Cambridge University Press: Cambridge, UK, 1993.

[7] A. Habibi, S. Tahar and A. ghazel, “Formal Verification of the ADSP-2100 Processor Using the HOL Theorem Prover,” Technical Report, Department of Electrical and Computer Engineering, Concordia University, March 2002.

[8] R. J. Higgins, “*Digital Signal Processing in VLSI. Englewood Cliffs*,” NJ: Prentice-Hall, 1990.

[9] C. Kern and M. Greenstreet, “Formal Verification in Hardware Design: A Survey”, *ACM Transactions on Design Automation of E. Systems*, Vol. 4, April 1999, pp. 123-193.

[10] T.F. Melham, “*Higher Order Logic and Hardware Verification*,” Cambridge Tacts in Theoretical Computer Science 31, Cambridge University Press, 1993.

[11] R. Milner and. M. Tofte. “*The definition of Standard ML*,” The MIT Press, Cambridge, Massachusetts and London, England, 1991.

[12] W. S. Steven, “*The Scientist and Engineer's Guide to Digital Signal Processing*,” Second Edition. California Technical Publishing, 1999.