Proceedings of the First PhD Symposium on Sustainable Ultrascale Computing Systems (NESUS PhD 2016)
Timisoara, Romania

Jesus Carretero, Javier Garcia Blas
Dana Petcu
(Editors)

February 8-11, 2016

Processor Model for the Instruction Mapping Tool

Roman Mego

Brno University of Technology, Czech Republic
roman.mego@phd.feec.vutbr.cz

Abstract

This paper describes the model designed for the instruction mapping tool, which can be used for generating the low level assembly code for the digital signal processing algorithms. The model is based on the Very Long Instruction Word architecture. The Texas Instrument TMS320C6678 was the pattern and finally was described with the created model. The paper is showing the parameters of the hardware resources and also the instruction set.

Keywords Processor model, Instruction mapping, VLIW

I. Introduction

Several years ago, in applications for digital signal processing applications, the critical code was not written using high level languages, but it was hand optimized in the assembly language. This approach was chosen because of the non-effective results generated by the compilers. This procedure resulted in the long development time and high cost. The other complication is that the final code cannot be used on the different processor architecture. In the case of the migration on the different processor, the code must be rewritten into the different form.

Nowadays, the modern compilers are capable of generating effective code. This statement applies mainly for the scalar processor architectures. It is given by the wide use of the scalar processors in different sectors, from the industrial and medical equipment, to the customer electronics, which leaded to the development of the effective compilers. There are also frameworks, where the architecture can be defined for various architectures such as [1] or [2].

But there are also different architectures, not widely used, where the use of high level languages leads to the ineffective code. These processors are usually the ones that use instruction level parallelism, such as superscalar or Very Long Instruction Word (VLIW). To avoid the problems related with the software creating using assembly language, the new tool for DSP algorithm mapping under development [3].

This paper is dealing with the processor model used in the tool. The next chapters will show the model structure based on the VLIW architecture.

II. Model Description

To cover the majority of possible cases of the processor internal structure, the more complex processor was chosen as the reference. It was the TMS320C6678 [4] which is 8-core digital signal processor based on the C66x CorePac [5] made by Texas Instruments.

Single C66x DSP core contains 8 functional units and 64 general purpose registers. Its simplified structure is shown in figure 1. At first sight, it may seem that the core has quite large amount of the resources for parallel operation, but it has its limitation.

The first is that the functional units are not equal. They are not capable to execute the same instructions. Functional units are marked .L1, .L2, .S1, .S2, .D1, .D2 and .M1, .M2. The .D units are primary used for the loading and storing data into the memory. The .L and .S units are designed for the general arithmetic, logic and branch operations as well. The last, .M units, are able to perform multiply operations with single and double precision floating point values. All of the units

are also able to execute other types of instructions, but not with all data types.

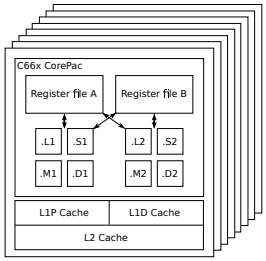

Figure 1: Simplified structure of the TMS320C6678.

The second limitation is caused by the division of the previously mentioned hardware resources into 2 identical data paths. These data paths are marked as Data Path A and Data Path B. Because of this it is not possible to directly access registers from Data Path A with functional unit from Data Path B. It can be done only through the Register File Cross Paths marked 1x and 2x. The single cross path in the C66x is capable to transfer 64-bit operand in the instruction. In addition, this operand can be used in multiple instructions in the same execute packed, which was not allowed in the older C64x core.

The model itself is aimed only on the description of the processor core, not the processor as the entire unit. The main parts of the model are:

- hardware resources of the core;
- instruction set.

II.1 Hardware Resources

The topology of the model is based on the VLIW architecture with the multiple data path.

II.1.1 Data Paths

From the outside view, the data path is the top level element, which contains all basic hardware resources. For this reason, the part of the model with the hardware resources is set of structures describing the data path.

The selected TMS320C6678 has 2 practically identical data paths, so the model in this case can contain only the template of one data path and information about the number of the data paths in the given architecture. But in general, the processor may consist of several different data paths, so every element in the model has its own definition.

Each data path contains the physical and virtual (or logical) resources, what will be explained later in the paper.

II.1.2 Cross Paths

As it was mentioned in the TMS320C6678 description, the data paths work as the separated units. The data cannot be directly moved between the register files and the functional units cannot read the register value. For this purpose, the model is able to define cross paths.

Each cross path is defined by the following parameters:

- source data path with register file;
- maximum width of the transferred data;
- maximum number of operands where the value can be used.

The meaning of the source data path is clean. The target data path is not defined at this point, because the functional units in the TMS32C6678 are not handling the operands in the same way. The .D, .M and .S units can read only the second operand through the cross path and the .L units can access to the different register file for both operands (figure 2). For this reason, the destination of the cross paths is defined individually on the functional units.

The maximum width of transferred data is given by the bus width, which is 64-bit in the selected processor despite the fact, that the register size is 32-bit. There is no need to define this parameter to different value than the multiply of register width, so the model keeps only the number of possible transferred registers.

The requirement of parameter which can tell if it is possible to use the operand transferred by the cross path in the multiple operations is given by the difference between the C66x and C64x cores. In the C64x, it is possible to use the data from the cross path only

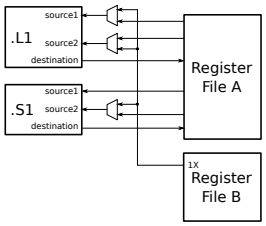

Figure 2: Example of the cross path connection to the functional units [5].

in the one functional unit at once in compare with the C66x where this limitation does not exists.

II.1.3 Functional Units

Each data path includes the set of functional units. The only one parameter, except the name, of the functional unit is the identification of the operand input connection to the cross path. The referenced C66x and also the older C64x are composed of the 2 data paths, so in this case the parameter could be only with the meaning connected or disconnected. But in general, the processor could have more than 2 data paths and therefore it is needed to identify which cross path is connected into the functional unit input.

II.1.4 Registers

The last physical hardware resources in the presented model are the general purpose register files. Each data path has one register file defined by the set of the registers. The registers are identified only by their names. Even the width of the registers is not mentioned in the model. To determine how many and which registers to represent data type, virtual resources are used. They will be described in the next chapter parts.

II.1.5 Register Groups and Data Types

Register groups are only logical definitions for the tool, to determine which registers can be used together as the single value (figure 3). As it was mentioned, the model is not working with the physical width with the registers. Also the registers can handle different number of bits on different architectures, so the decision which group to use as given data type cannot be made. For this reason, the data types supported by the tool are assigned to the created register groups.

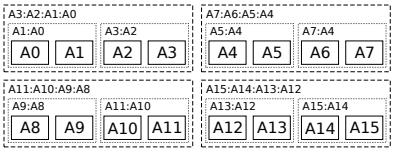

Figure 3: Creating register groups from the physical registers.

III. Instruction Set

The instruction set is next big part in the model describing the processor. It is not divided into other segments as the hardware resources. It is only the list of the instructions that can fit into the operation abstraction of the tool. It includes the arithmetical and logical operations and the data loading and storing instructions.

Each instruction is represented by the following attributes:

- name of the instruction;
- instruction format;
- instruction operation;
- data type of the operands;
- functional units capable to execute instruction;
- number of cycles needed to read the instruction and operands;
- number of cycles needed to write result to registers;
- total number of cycles needed to execute the instruction.

The meaning of the instruction name is clear. Its purpose is only the identification by the user.

The instruction format gives the position of the parameters in the final notation of the generated code.

Some of the instructions are able to process data with different number representation. For example the ABS instruction in the C66x is able to process 32-bit integers and 64-bit integers as well. That is why this parameter is list of the data types.

Functional units are another list acting as the instruction parameter. This list contains the functional units from all data paths. They are not divided into smaller groups.

The last group of parameters defines the timing of the instruction. The full instruction cycle was reduced into 3 stages. During the read stage, the functional unit is fetching instruction and the input value must be prepared in the registers. After this stage, the functional unit can be used for other purpose and the input register can be overwritten. The write stage moves the result of the operation into the destination registers. At this stage, the register must be prepared to receive new data to prevent overwrite the valid values for other operations. The instruction is executed between these stages and the resources can be freely used without limitations. Figure 4 shows the timing of the MPYDP instruction as the example.

Pipeline stage	1	2	3	4	5	6	7	8	9	10
Read	src1_l src2_l	src1_l src2_h	src1_h src2_l	src1_h src2_h						
Write									dst_l	dst_h
Unit in use	.M	.M	.M	.M						

Figure 4: MPYDP instruction pipelining.

IV. Implementation

The processor model is implemented as part of the instruction mapping tool. This tool is written in the C++ language and the model is not specified directly. It is in form of classes and the tool is reading user specified JSON file [6], which contains the structure of the specific architecture.

The simple command line tool to editing the architecture was also created. This editor is helpful during the defining the new architecture, because it keeps the valid format of the files, which could be corrupted by the mistype and also watches over the right connection between the parameters.

V. Conclusion

This paper presented the processor model designed for the instruction mapping tool, which was primary intended for VLIW architectures. The model was implemented as the part of the instruction mapping tool. Its functionality was verified with the mentioned tool on the TMS320C6678 processor. The model is primary aimed on the VLIW architectures, but it should be able to define other architectures such as the scalar or superscalar processors. This was not verified and it will be the part of the future work.

Acknowledgment

Publication of this paper was supported by the COST action IC1305, Network for Sustainable Ultrascale Computing (NESUS).

References

[1] I. Povazan et al., "A Retargetable C Compiler for Embedded Systems," in *Engineering of Computer Based Systems (ECBS-EERC) 2013 3rd Eastern European Regional Conference*, August 2013.

[2] S. Rajagopalan et al., A retargetable VLIW compiler framework for DSPs with instruction-level parallelism, *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 20, issue 11.

[3] R. Mego and T. Fryza, "Tool for algorithms mapping with help of signal-flow graph approach", in *Radioelektronika 2014 24th International Conference*, April 2014.

[4] Texas Instruments, *Multicore fixed and floating-point digital signal processor* [online], Available: http://www.ti.com/lit/ds/symlink/tms320c6678.pdf.

[5] Texas Instruments, *TMS320C66x CorePac user guide* [online], Available: http://www.ti.com/lit/ug/sprugw0c/sprugw0c.pdf.

[6] ECMA International, *ECMA-404 The JSON Data Interchange Format*, 1st Edition, Available: http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf.