

INSTRUCTION MAPPING PROCESS ON THE VLIW ARCHITECTURES

Roman Mego

Doctoral Degree Programme (4), FEEC BUT

E-mail: roman.mego@phd.feec.vutbr.cz

Supervised by: Tomas Fryza

E-mail: fryza@feec.vutbr.cz

Abstract: This paper deals with the process of instruction mapping on the digital signal processors. This process is used by the newly developed tool, which is designed for generating low-level assembly code for very long instruction word processors. The tool is suitable for creating cores of the signal processing algorithms.

Keywords: instruction mapping, low-level, digital signal processing, very long instruction word

1. INTRODUCTION

The advanced real time signal processing applications are realized mostly using the digital signal processors (DSP) instead of the analogue circuits. The complexity of the digital signal processing algorithms is still increasing, which puts higher requirements on the DSP resources. The main aspects influencing the performance of the final application are:

- the architecture suitability,
- the software optimization.

The DSPs from the architecture point of view are highly optimized for the frequent access to the memory, executing instruction in the loops and performing special operations typical for the DSP algorithms, such as multiply and accumulate (MAC) [1].

The software optimization is the second aspect influencing the final performance of the application. Several years ago, critical parts of the software was optimized by the hand in the low-level assembly language. This approach was chosen because the high-level compilers were not generating efficient code. This leads to the longer development time and therefore to higher cost.

Nowadays the situation has changed. The modern compilers are able to optimize the final code which is practically not worth to spend more time on hand optimization. This especially applies for the scalar processors. There are also special architectures, where the results are less effective and there is still worth to do low-level parts of software. They are mainly using the instruction level parallelism. This also includes the very long instruction word (VLIW) architecture [2], where the creation of low-level code is more complex than on the traditional scalar architectures. There are also retargetable compiler frameworks, such as [3], which are more efficient than commercially available compilers, but creating desired model of the processor is not an easy task. For this reason, the tool for efficient instruction mapping is developed.

This paper will introduce the approach used by the mapping tool. This tool allows to implement DSP algorithms on VLIW processors in the efficient way. It uses the signal-flow graph description of the algorithm which is transformed low-level assembly code.

2. REFERENCE PLATFORM

The VLIW architecture, as it was mentioned above, takes the advantage from the instruction parallelism. This means, that single processor core contains multiple functional units able to execute operations in the same time [2]. The difference between VLIW and superscalar processor is the point of decision, which operations will be performed at the same time (figure 1). The superscalar processors contain the dispatch buffer, which dynamically allocates the functional units during the runtime [4]. The instructions on VLIW architectures are allocated statically during the software compilation. The compiler creates the instruction packets, which are directly sent into the functional units. This creates the space for the core simplification, optimization or extension for the new functionality.

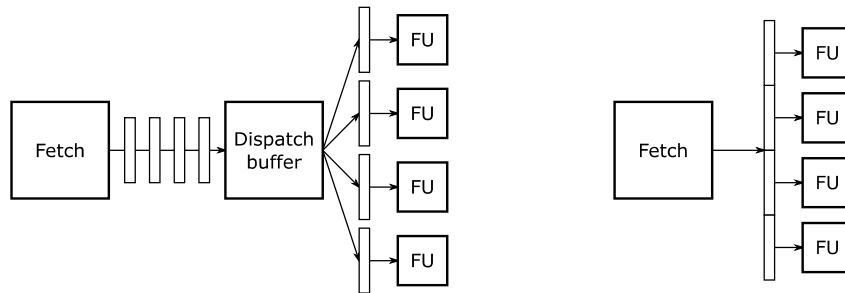


Figure 1: Superscalar (left) and VLIW (right) difference.

The reference processor for the tool design is the TMS320C6678 from the Texas Instruments [5]. This is the 8-core fixed/floating-point DSP. Each core consists of the two identical data paths, where each data path has four functional units and the set of thirty-two 32-bit registers (figure 2). The functional units are not identical in the same data path. Each of them was designed for different purpose. The processor was not chosen, because it is multi-core, but for the diversity in the core structure.

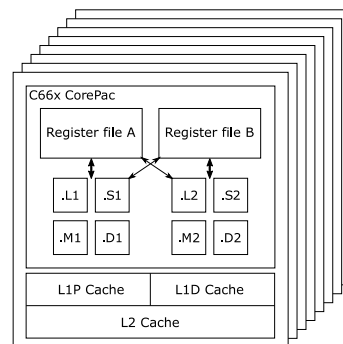


Figure 2: Simplified structure of the C66x core.

3. TOOL IDEA

The tool will be used for optimizing the signal processing operations, not the branch or function call control. The idea is based on the main weakness of the compiled code for this kind of processors. The high-level notation of algorithms is sequential and compiler tries to keep the same order of the instructions. The result is that the final code uses the limiting number of the functional units in the same time.

The tool uses the signal-float graph approach for the algorithm description. This description, as the analogue circuit, can contain multiple parallel signals and operations. The order of the processing will not be as it is in the source code text file, but with the relations of the signals and operations in the algorithm. This should be decided automatically by the tool without the user interaction.

4. INSTRUCTION MAPPING PROCESS

The whole process of the instruction mapping from the reading of the input files to the generating low-level assembly code can be divided into the following steps:

- reading and parsing input files
- finding the relations between the operations and signals
- sorting the operations and signals
- mapping the operations and signals to the functional units and registers
- generating output file with low-level assembly code

4.1. READING AND PARSING INPUT FILES

When the tool is executed, it needs to know the algorithm which will be mapped and also the architecture. The information is stored separately in two text files. The first keeps the information about the target processor, which consists of:

- the list of data paths,
- the list of functional units,
- the list of registers,
- the list of the instructions.

The second file contains the algorithm description, which is independent on the architecture. The main elements are:

- signals,
- operations.

Signals are represented with the data type and its functionality. The functionality divides the signals on three groups, input, output and internal signal. Operations are the mathematical or logical actions performed on signals. It can be recognized as the node in the signal graph with signals on its inputs and output.

4.2. FINDING RELATIONS BETWEEN THE SIGNALS AND OPERATIONS

When the algorithm notation is parsed, there is time to finding the relations. This step can be used to validate the connections. Signals practically represent the variables, which are used in the mathematical operations. The difference in comparison with the ordinary programming languages is that the value to the signal can be assigned only once. This is similar to the hardware description languages (HDL). All internal signals must have assigned its creator (operation) and it must be used as the input at least on one operation. The input and output signals do not need to respect all of these conditions. The example is shown on figure 3, which represents single butterfly of the Fast Fourier Transform (FFT).

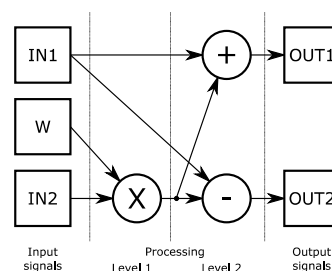


Figure 3: Single FFT butterfly represented by signal-flow diagram.

In this step we can also connect the operations in the algorithm with the suitable instructions supported by the processor. The instruction is chosen on the operation itself and the data type of the input and output signals. The tool now creates internally new structure, which references the architecture independent parsed data and extends it with other architecture dependent information.

4.3. SORTING THE OPERATIONS AND SIGNALS

Sorting is the first step of the mapping itself. It is preparation of the possible execution order, which will be assigned to the functional units in the next steps. This order is based on the relations between the items in the algorithm. The main condition is that the operation must be on higher level than the highest level from the creators of the all input signals. For example, if all of the operation input signals are input signals of the algorithm, this operation is placed on first level. If one signal is the input signal of the algorithm and second signal is created by the operation on second level, the operation is placed on third level. Multiple operations can be placed on the same level without any restriction given by the architecture (figure 3). There can be additional sorting on the individual levels for better results.

4.4. MAPPING OPERATIONS AND SIGNAL

The mapping of the operations and signals is realized with the two 2-dimensional maps (figure 4). First dimension represents time in instruction cycles in both maps. Second dimension represents hardware resources, functional units or registers. During the mapping, the order of the operations is reflected in the process. The operations are mapped as the first. The placing is similar to first-fit method from the first instruction cycle, so in final different levels can be executed simultaneously if there is a free space in the map. Operations are mapped only on one data path to avoid the usage of the cross paths.

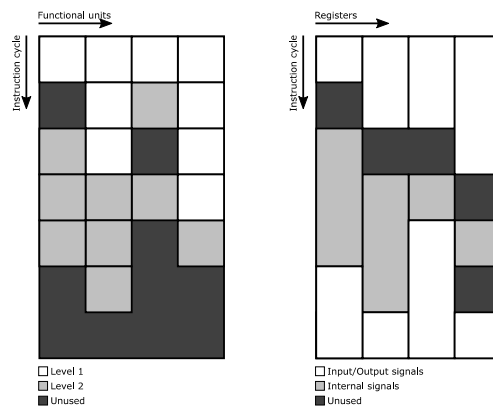


Figure 4: Example of functional unit and register map.

After the operations are mapped to the functional units, the signals must be allocated to the registers as well. In first step, the lifetime of the signal is determined from the position of operations in time. The lifetime of the signal starts at the end of the operation which creates it and ends when it passed into the last operation. This means, that the registers are allocated for the signal only when necessary, not for the whole time of the algorithm. The input and output signals has its lifetime given by the start and end instruction cycle of the final algorithm. After that, the signals are mapped in the similar way like the operations.

4.5. GENERATING OUTPUT FILE

The last step it to generate output file with the low-level assembly file. All information is provided in the referenced structures form the maps. Each instruction in the architecture definition contains the template for human readable notation, where the functional unit and registers will be added. The tool goes through operation map from the first instruction cycle and joins available operations together into the instruction packets.

5. EXAMPLE OF THE GENERATED MAP

The example of the mapped algorithm is showed on figure 5. The first part is the source code of the 4-point Fast Fourier Transform without the signal definitions. The next two part are maps of the functional units and registers. More information about the mapped algorithms can be found in [6].

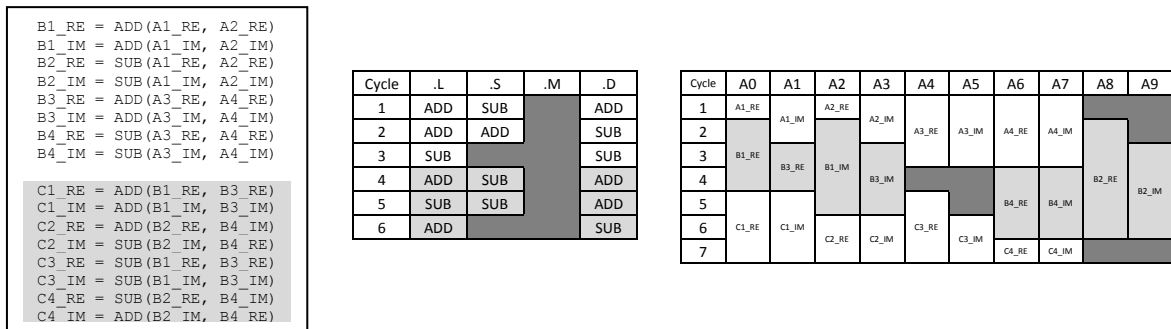


Figure 5: Source code of 4-point FFT and generated map for resource usage.

6. CONCLUSION

This paper presented the way of the instruction mapping using the signal-flow graph approach instead of sequential notation. It means that the desired algorithm is described only with the relations between signals and the tool determines the order of the execution. This approach was adjusted for the VLIW architectures. The tool can be also used for scalar processors, but probably without any benefits. The efficiency of the generated code for TMS320C6678 model is close to the hand-optimized code. The disadvantage of the tool is, that it can be used only for cores of the algorithms, because the limiting factor is the number of the registers.

ACKNOWLEDGEMENT

The research published in this submission was financially supported by the Brno University of Technology Internal Grant Agency under project no. FEKT-S-14-2177 (PEKOS).

REFERENCES

- [1] Smith, Steven W. The scientist and engineer's guide to digital signal processing. 1st ed. San Diego, Calif.: California Technical Pub., 1997. ISBN 0966017633.
- [2] Philips Semiconductors. An Introduction to Very Long Instruction Word Computer Architecture. Publication no. 9397-750-01759.
- [3] Rajagopalan, Subramanian et al. A retargetable VLIW compiler framework for DSPs with instruction-level parallelism. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems. 2002. ISSN 02780070.
- [4] Shen, John Paul. Modern processor design: fundamentals of superscalar processors. Long Grove: Waveland Press, 2013. ISBN 9781478607830.
- [5] Texas Instruments Incorporated. TMS320C6678 Multicore Fixed and Floating-Point Digital Signal Processor. [Online] 2014. [Cited: 14 March 2016]. <<http://www.ti.com/lit/gpn/tms320c6678>>.
- [6] Mego, Roman and Fryza, Tomas. Efficiency of the signal processing algorithms using signal-flow based mapping tool. International Conference Radioelektronika. 2015. ISBN 978-1-4799-8117-5.