Teaching Multi-Core DSP Implementation on EVM C6678 Board

Aleksei Kharin, Sergey Vityazev, and Vladimir Vityazev
Department of Telecommunications and Radio Engineering Foundations
Ryazan State Radio Engineering University, RSREU
Ryazan, Russia
vityazev.s.v@tor.rsreu.ru

***Abstract*—Teaching implementation of digital signal processing systems plays a very important role in recent technical education. The multi-core digital signal processor (DSP) is a new type of architecture widely used now in the industry. A new course on multi-core DSP programming is considered in this paper. The lab experiments are described. The course has been developed for the TMS320C6678 multi-core DSPs. This paper provides educators with a content that cover theoretical and technical skills that are required by industry.**

Keywords—multi-core; DSP; signal processing; programming; real-time processing

I. INTRODUCTION

The adoption of courses on the practical implementation of digital signal processing systems into educational programs on Electrical & Electronic Engineering or Computer Science is widely used in universities. Many such courses have already been developed for different hardware platforms, including general-purpose processors (GPPs) [1-3], field programmable gate arrays (FPGAs) [4, 5], graphical processing units (GPUs) [6] and digital signal processors (DSPs), which are historically the first integrated circuits specially designed to solve digital signal processing tasks in real-time with low-power consumptions. The educational courses on DSP programming are described in [7-11]. In this paper we concentrate on multicore (DSPs).

During the last decades, DSP has evolved dramatically, moving from a single chip with one core to a multi-core heterogeneous System-On-a-Chip (SoC). The processor industry stepped to multi-core architectures in the second decade of the 21st century [12-13]. The multi-core approach tried to solve the problem of higher computations while staying low power, but it produced a lot of new problems, including algorithm parallelization, multi-core programming and parallel threads synchronization. Multi-core gave new challenges to universities. Previous courses had to be updated or new ones had to be developed. It became necessary to give much more information to students within the course. New terms and new principles started to be used. New architectural components such as new instructions, new peripherals, new bus structure and hardware accelerators appeared. On the software side, new programming tools started to be used such as Open Multi-Processing (OpenMP) and Open Computing Language (OpenCL).

In this paper, a new educational course on multi-core DSP programming is considered. The goal of the course is to give students a knowledge base on modern approaches to multi-core DSP software development. The structure of the course is offered. The laboratory experiments are designed and described. The course has been developed for the TMS320C6678 multi-core DSP from Texas Instruments (TI).

The paper is divided into six sections. After the introduction, the laboratory infrastructure including laboratory equipment and software tools is described in Section II. Section III considers the details of lab experiments. The course project which students do to consolidate the knowledge is described in Section IV. Section V deals with assessment aspects. Finally, some conclusions are made in Section VI.

II. LABORATORY INFRASTRUCTURE

A dedicated digital signal processing laboratory supporting the development of real-time software is used for teaching multi-core DSP programming. The laboratory is equipped with EVM C6678 evaluation boards [14]. The diagram of that board is illustrated in Fig. 1. The board is based on TMS320C6678 DSP. This is an 8-core homogenous processor with all cores of the same type. Each core is a powerful 1-GHz DSP with very long instruction word (VLIW) architecture capable of 32 milliard multiply-accumulate operations per second (GMACS) in a fixed-point arithmetic and 16 milliard floating-point operations per second (GFLOPS) in a floating-point arithmetic. So each core is a high performance state-of-the-art device and 8 cores on one chip make it possible to increase its productivity 8 times theoretically. During the labs, students learn how to achieve this maximum level of performance and analyze the reasons which limit the performance in practice.

The board does not contain any analog-to-digital convertor (ADC), so it can not be used for direct analog signal processing. It has a number of digital interfaces instead including Ethernet, PCIe and SRIO, Hyperlink and some others. It also includes 512 Mbytes of DDR3 memory to store data and code. During the labs the following

ISBN 978-0-9928626-7-1 © EURASIP 2017

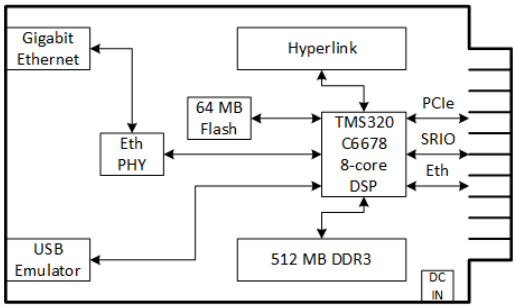

Fig. 1. Block diagram of TMS320C6678 evaluation board

approach is used. A model of a signal is generated and stored into DDR3 onboard memory. This generation is performed on a personal computer or on the board but cannot be accomplished in real-time.After the data are ready in the external memory, real-time signal processing can take place. The results are also stored into DDR3 memory. The results are verified and processing time is measured.

Software in the laboratory includes Code Composer Studio (CCS) integrated development environment (IDE) and software development kit (SDK) from TI that give all the instrumentations required for multi-core DSP programming. The MATLAB environment is an additional specialized software component used in the laboratory for input signal generation and output signal analysis and visualization. The board and student workplace in the laboratory are shown in Fig. 2.

III. LAB EXPERIMENTS

The course contains the lectures and the lab experiments. The required theory is given to the students at the lectures and the labs allow students to consolidate the knowledge with practice. The following lab experiments are offered to the students throughout the course fully reflecting the contents of the course (Fig. 3 and 4).

A. Lab #1. Introduction to CCS

The general task of real-time digital signal processing is discussed at the beginning of the course. Fast repetitive multiply-accumulate (MAC) calculation is stated to be the key to efficient signal processing implementation, and the programming of the processor is considered as a basic way to digital design.

A simple program is written in C-language during the lab #1. The program implements a dot product of two pre-calculated vectors – 128-length floating-point format coefficients and signal samples. The program is written as a CCS-project. It is compiled and loaded onto the DSP on the board. It is run and gives some predefined results.

During the lab students get knowledge of how to work in CCS, how to connect to the real hardware, and how to run their code on DSP. Only one-core is used at this stage. Students implement a basic digital signal processing example. They also measure the processing time with cycle-count registers and make some statements about real-time

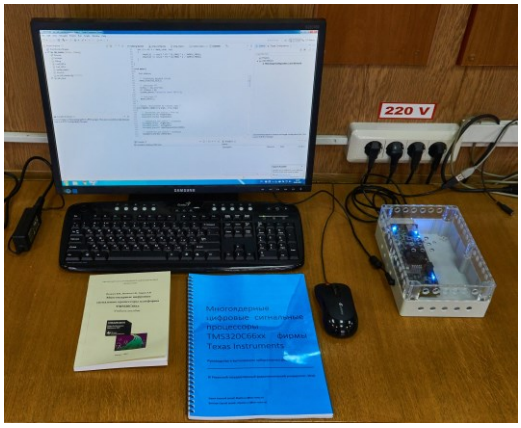

Fig. 2. The board and the workplace in the laboratory

processing. So, the lab #1 is an introduction to the tools and DSP programming in general.

B. Lab #2. Architecture

The architecture of the TMS320C6678 DSP with special attention paid to single core architecture is considered at the lectures and is better understood during lab #2. The computational power of the TMS320C6678 as of any other multi-core processor is achieved not by the number of cores but by the architecture of each of the cores. Moving from single to eight cores theoretically gives eight times the performance gain, but such a gain is not achievable in practice. Single-core software optimization is able to give much better results. One should not think about a multi-core processor as a computational machine capable of solving any computational problem efficiently. A good single core solution should be achieved first and then eight-core implementation can be started. All computational capabilities and limitations are concentrated inside the core. It is very important to explain to students these aspects of working with multi-core. This is the reason the course is started from the single core architecture description and its influence on the overall device performance.

During the lab #2, students modify the program from lab #1 rewriting the dot product function in assembly and calling it from C-project. The assembly code is written for the floating-point arithmetic case. The execution time is measured and compared with the maximum DSP capabilities.

C. Lab #3. Optimization

The course continues with single-core optimization. It is very important to explain to students that multi-core implementation does not make sense if the code for a single core is not optimized. Optimization is considered as a process of software improvement aimed at one of the optimization criteria. Typical criteria are processing time and code size. The processing time optimization criterion is considered throughout the course as it is the most common requirement. Optimization and debugging join together into a common process of software development.

ISBN 978-0-9928626-7-1 © EURASIP 2017

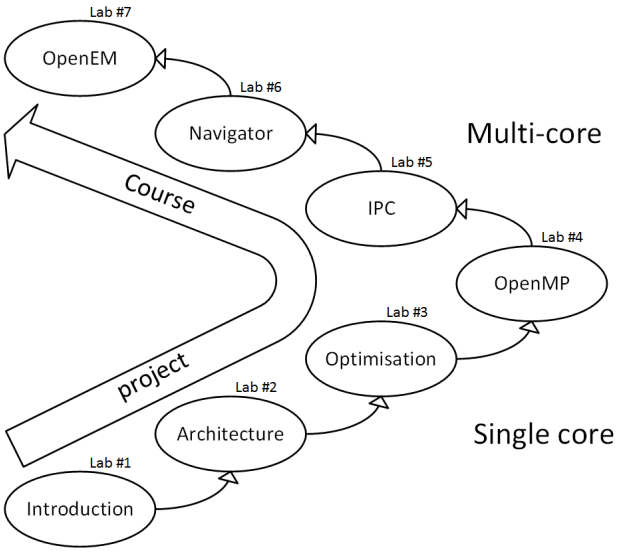

Fig. 3. Course structure

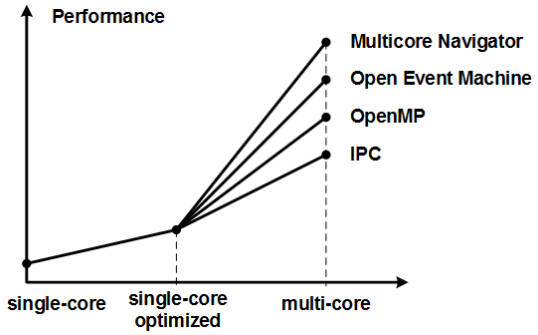

Fig. 4. Course main steps

During the lab, students implement the dot product function with hand-optimized assembly. They learn how to use VLIW DSP architecture more efficiently with parallel instructions and software pipelining technique. Processing time decreases from 2700 processor cycles to 70 cycles, which means about 40 times performance gain with single-core optimization.

Optimization in C is accomplished then for the same dot product function. The efficient code is generated by the compiler with optimization property turned on. Compiler feedback is analyzed. Additionally, C extensions including pragma directives and keywords are used to achieve processing time similar to hand optimized assembler [15].

D. Lab #4. OpenMP

The course moves from single to multi-core implementation at this stage. Introduction to parallel processing is given first during lectures. Basic parallel computing notions are considered including parallel threads, data racing, thread synchronization, shared memory accesses and cache coherency.

First multi-core project is accomplished with the OpenMP technique as the simplest way of software parallelization. The dot product function becomes too simple to be implemented on several cores, so students move to a more complicated task of finite impulse response (FIR) filtering. A frame of input samples is generated in advance and loaded into the shared memory. FIR coefficients are also known and preloaded into the memory. Simple sinusoidal signals are used to test the program. The length of a frame can be about several thousands and the length of impulse response is 128. FIR-filtering is performed based on the dot product function written and optimized within previous labs. Execution times of single- and multi-core implementations are measured and compared.

It is very important for the students at this first stage of multi-core programming to realize that multi-core overheads always take place and this should be in mind during parallelization strategy selection. To study this, students perform parallelization at different levels. In the case of the FIR-filter, they can choose a different frame length. The greater the length, the more computations are performed independently, resulting in a higher processing time to parallelization overhead ratio.

E. Lab #5. IPC

The same task of FIR-filtering is performed then with another method of parallel software design called Inter Processor Communication (IPC). In a broad sense, IPC is a common name for any communication between cores. But talking about TI's processors, the user should think of IPC as a software package of modules developed by Texas Instruments to enable inter processor communications for a variety of TI's processors (not only DSPs).

Students develop two projects: one for the master core and another one for all slave cores. The master core makes data ready and performs general configuration. Then the master core sends messages to the slave cores. These messages bring information about which data should be processed by slave cores: a pointer to the beginning of data chunks inside input buffer, the length of chunks and a pointer to a part of the output buffer, slave cores have to calculate. This is done with MessageQ module APIs. Then the master core waits for the response from all slave cores. Slave cores send messages to the master core with information about their readiness for processing. The master core notifies slave cores that they may start processing. This is done with Notify module APIs. All cores make their part of calculations. They are synchronized again after they finish the processing.

Students verify the result of multi-core filtering and measure the time for signal processing. This time can be compared to the single core processing time and to the OpenMP parallel processing time.

F. Lab #6. Multicore Navigator

Many modern multi-core architectures use special hardware to facilitate fast data movements and job distribution inside the chip [16, 17]. The multicore Navigator achieves this in the TMS320C6678 DSP. It manages fast and

ISBN 978-0-9928626-7-1 © EURASIP 2017

flexible data movements inside the multicore system and coordinates the work of the cores. Students study data movement with the Navigator during lab #6. Lab #7 considers the second function of the Navigator – job distribution with the Open Event Machine (OpenEM).

The task is still the same. FIR-filtering is performed and parallelized into 8 cores of C6678 DSP. The parallelization strategy is similar to the IPC case but special hardware (the Navigator) is used. Again we do not move any data that should be processed. The input signal is located in the shared memory and the output signal is written to the shared memory too. The multicore Navigator is used to send processing information from the master core to the slave cores and to perform synchronization.

There are two possible ways to program the multicore Navigator: to do it through registers directly or to use TI's low level drivers. It is better to start with registers to understand how to work with the Navigator. But to make it easier to work with registers, we implemented some kind of API that is low enough to show how to program the multicore Navigator through registers but high enough not to bother students with writing to registers themselves.

When the laboratory is completed, students can compare the resulting processing time with that previously received in the OpenMP and IPC labs. These results show that, working at a low level with hardware modules, the best performance is achieved.

G. Lab #7. OpenEM

OpenEM is an open standard for systems with dynamic tasks distribution. It allows the dispatch of dynamically created tasks between cores in multicore systems. Texas Instruments implement OpenEM for their multicore DSPs using capabilities of the multicore Navigator hardware. Actually, OpenEM is implemented as a firmware for PDSP (Packed Data Structure Processor) which is a part of the multicore Navigator. OpenEM utilizes the multicore Navigator resources which is the reason we talk about using OpenEM after the multicore Navigator despite being easier to use and to understand.

After the concept of OpenEM is given and all the necessary details are discussed at the lecture, students go through the laboratory practice. The task for the laboratory is still the same – perform signal frame filtering with computation distributed across 8 cores. The input signal frame consists of 8128 samples and is loaded into the internal shared memory. The output buffer has a length of 8000 samples and is also located in the shared memory. A 128-coefficient filter impulse response is located in the local memory of each core. The master core configures the system, including OpenEM. The master core creates OpenEM events. There are two event queues. One queue includes processing events. There are 8 of them, so each event can be processed on a separate core. Processing events includes information about what to process (a pointer to a part of the input buffer, buffer length, a pointer to a part of the output buffer). All cores process the events. It means they do filtering. The second event queue is processed then. The events in the second queue just signal the cores that they may stop. The master core outputs the results and time of processing.

IV. COURSE PROJECT

The course project is accomplished by students to consolidate their knowledge and to apply it to practical tasks. Students accomplish the project individually. The theme of the project depends on the program students are studying. We will consider here an example of a project which is offered to the students studying for communications.

The task of signal demodulation is solved by students during the course project. A teacher gives students signal records in a raw data format. Each signal is a phrase recorded with the MATLAB. It has a duration of several seconds and is recorded at a sampling frequency of 70-100 kHz. The spectrum of speech is moved to a carrier frequency which is about 20 kHz. Additionally, some narrowband noise is added to the signal. The spectrums of the speech and the noise do not overlapped. Students have to restore the phrase. They demodulate the signal multiplying it by a harmonic and do FIR filtering. Both procedures should be implemented on a single core first. The software should be optimized. Multi-core implementation should be implemented then with any two different parallelization techniques, for example, OpenMP and IPC. No strict requirements to the final software are defined. Students are free to choose the limits for the optimization and to select parallelization schemes.

V. COURSE ASSESSMENT

Much information is given to the students during the course and the materials are difficult enough. That is why formative assessment is absolutely necessary during the course allowing a check to ensure that the students understand the materials. The formative assessment is built upon the lab experiments. A teacher and his assistant check if students complete the labs successfully. Moreover, each lab is defended by the students in the oral form. The teacher (or the assistant) have to confirm that the students understand the lab. The formative assessment does not influence the final grades and is used just to control the intermediate results of learning.

The summative assessment includes a presentation of the course project and a final exam. Each student prepares a report on the course project in a printed form. Retrieved signal records are also prepared by the students. The teacher reviews the reports and checks if the signal is retrieved correctly. Then, the course project is defended in an oral form. The originality of the report, the correctness of software and the understanding of the materials by the students are taken into account during project grading.

Finally, the students pass an exam. The exam is passed in an oral form. This gives the final grades to the students.

VI. CONCLUSION

An educational course on multi-core DSP programming is considered in this paper. The structure of the course is

ISBN 978-0-9928626-7-1 © EURASIP 2017

offered and the results are described. To stay relevant and to become more efficient, the course has to be modified regularly. More advanced courses can include heterogeneous multi-core programming and operating systems.

REFERENCES

[1] J. O. Hamblen, "Using a Low-Cost SoC Computer and a Commercial RTOS in an Embedded Systems Design Course," IEEE Trans. on Education, vol. 51, issue 3, 2008, pp. 356-363.

[2] A. J. Kornecki, J. Zalewski, and D. Eyassu, "Learning real-time programming concepts through VxWorks lab experiments," Thirteenth Conference on Software Engineering Education and Training, 2000, pp. 294-301.

[3] M. A. Wickert, "Using the ARM Cortex-M4 and the CMSIS-DSP library for teaching real-time DSP," IEEE Signal Processing and Signal Processing Education Workshop (SP/SPE), 2015, pp. 283-288.

[4] N. Kehtarnavaz, and S. Mahotra, "FPGA implementation made easy for applied digital signal processing courses," IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), 2011, pp. 2892-2895.

[5] C. Haba, "Using FPGA development boards for multi-course laboratory support," IEEE Global Engineering Education Conference (EDUCON), 2014, pp. 794-797.

[6] A. Asaduzzaman, R. Asmatulu, and M. Rahman, "Teaching Parallel Programming for Time-Efficient Computer Applications," International Journal of Computer Applications, vol. 90, No 7, March 2014 pp. 18-25.

[7] S. Gannot, and V. Avrin, "A Simulink and Texas instruments C6713 based digital signal processing laboratory," European Signal Processing Conference, 2006, pp. 1–4.

[8] C. E. Wick, and G. E. Piper, "Using the ADSP-21061 SHARC EZ-KIT in undergraduate DSP oriented courses," Frontiers in Education Conference, vol. 2, 1998, pp. 691-694.

[9] D. S. Reay, "OMAP-L138 low cost development kit (LCDK) for hands-on DSP teaching," European DSP Education and Research Conference (EDERC), 2012, pp. 164-168.

[10] L. G. Huettel, "A DSP Hardware-Based Laboratory for Signals and Systems," 2006 IEEE 12th Digital Signal Processing Workshop & 4th IEEE Signal Processing Education Workshop, 2006, pp. 456–459.

[11] D. Reay, "Digital Signal Processing and Applications with the OMAP-L138 eXperimenter", Hoboken, New Jersey, John Wiley & Sons, Inc., p. 340, 2012.

[12] G. Blake, R.G. Dreslinski, T. Mudge, "A survey of multicore processors," Signal Processing Magazine, vol. 26, no. 6, pp. 26-37, Nov. 2009.

[13] L.J. Karam, I. AlKamal, A. Gatherer, G.A. Frantz, "Trends in multicore DSP platforms," Signal Processing Magazine, vol. 26, no. 6, pp. 38-49, 2009.

[14] TMDXEVM6678L EVM Technical Reference Manual, SPRUH58, Version 2.0, Texas Instruments, p. 88, 2011.

[15] T. Hahn, J. Humphreys, A. Fritsch, D. Greenstreet, "Demystifying digital signal processing (DSP) programming: The ease in realizing implementations with TI DSPs," Dallas, Texas, Texas Instruments Inc., p. 10, 2015.

[16] S. K. Moore, "Breaking the Multicore Bottleneck," IEEE Spectrum, November 2016, p. 16.

[17] E. Biscondi, T. Flanagan, F. Fruth, Z. Lin, F. Moerman, "Maximizing Multicore Efficiency with Navigator Runtime," Texas Instruments, 2012

ISBN 978-0-9928626-7-1 © EURASIP 2017