Computing and Informatics, Vol. 32, 2013, 509–526

EDUCATIONAL TOOLS FOR OBJECT-ORIENTED DSP INTERACTIVE DSL FRAMEWORK

Anita Sabo, Bojan Kuljić, Tibor Szakáll

Subotica Tech
Marka Oreškovića 16
24000 Subotica, Serbia
e-mail: {saboanita, bojan.kuljic, szakall.tibor}@gmail.com

Communicated by János Fodor

Abstract. This paper presents DSP blocks which were developed to be used as basic elements for realization of the DSP algorithms. For this purpose the description DSL (Domain Specific Language) language was used. The goal of this paper is to define and present a high level language that allows description and development of signal processing algorithms. With the usage of a domain specific language, one can create a compact and easy to understand definition of algorithms. In the paper the authors present the advantages granted by DSL for DSP applications. The created definitions are hardware independent and they can be executed and functionally verified. Efficient code can be generated for various targets without porting. The design of the presented DSL allows code generation for multi-core targets in case of computing-intensive algorithms, code generation for multiple streams and threads. To validate the results these blocks were made available for use to students as an easy method for the introduction of the DSP algorithms in sound and image processing. The main purpose was for the students to gain some basic insight into elementary techniques needed for design, implementation and merging of hardware and software components used in testing of the algorithms for digital signal processing in real time. Through the work with the students it was concluded that the developed DSP blocks presented very good assistance in educational process and therefore this paper was elaborated on that idea. Since real hardware systems were used in this case noise was introduced in the system which does not exist in simulation software and therefore this option produced much larger capabilities for development of the robust algorithms.

Keywords: Educational tool, simulation of DSP filters, interactive framework, functional programming, DSL language, code reuse

1 INTRODUCTION

The task of programming concurrent systems is substantially more difficult than the task of programming sequential systems with respect to both correctness and efficiency. The tendency in development of embedded, DSP systems and processors is shifting to multi core and multiprocessor setups as well. The problem of easy concurrency and algorithm development is important for embedded and DSP systems as well. Code reuse is supported by merging, re-grouping, and splitting of algorithms and groups of algorithms. DSP software development environment allows the application of high-level elements in the target hardware [1]. This development environment uses high-level elements to construct the algorithms. Using a suitable simulator program during the education process may help students to learn and develop power electronic circuits and improve the education quality of power electronics course [2, 3, 4]. The general problem with the algorithm development through specialized software lies in the absence of the basic construction elements. The lack of these elements results in redundancy in software environment because the same implementation steps appear in different constructive elements. There is no unique method for description of the components which makes it very hard to introduce new elements into design without collision with existing elements. With the unification of the basic DSP blocks it was possible to introduce software environment for easy algorithm design. It was possible to identify high-level elements capable of describing any DSP component and therefore the algorithm. This leads to grouping of those elements inside the frame of descriptive language. The application of such a structure allows the implementation of the realized DSP blocks independently of software development environment. The descriptive language consists of basic DSP elements through which complex algorithms were developed. The descriptive language used was the declaration DSL (Domain Specific Language) language which allowed for DSP programs to be designed in the form of graphical diagrams. The purpose was to define specific descriptive language (DSL + UML) in the targeted area (DSP). The language that has been designed with regard to the above statements is used in different fields of signal processing and therefore not limited to only one area, e.g. of sound processing. The language does not contain descriptive operations; algorithms are displayed through data flows and transformations. Because of this feature algorithms created in this language simplify the verification and parallel processing which was a great aid in digital signal processing development. High-level elements annotation enhances program analysis and optimization. Thanks to the logic (semantics) of the language the developed algorithms were easy and simple to realize in the DSP software. The purpose of this research was to uniquely determine the basic operations/functions in the applied DSP algorithms, in order to create a specific language for the DSP area and also to evaluate their advantages and disadvantages. Beside the above-mentioned goals it was also important to evaluate the application of the optimized algorithm, parallelization and feature verification of the DSP specific programming language. As a consequence it has been possible to explore high-level algorithms which are based on the elements of the DSP specific language. Domain specific languages are widely used today, but in the field of digital signal processing there are very few representatives. In the field of digital signal processing there are domain specific languages but they have been developed for use in one specific area, for example, sound processing. There is a need for a modern domain specific language which could generally be used in the field of the DSP development and which could make the work of the developers easier regardless of the area.

2 CONCURRENT PROGRAMMING

Concurrent computing is the concurrent (simultaneous) execution of multiple interacting computational tasks. These tasks may be implemented as separate programs, or as a set of processes or threads created by a single program. The tasks may also be executing on a single processor, several processors in close proximity, or distributed across a network. Concurrent computing is related to parallel computing, but focuses more on the interactions between tasks. Correct sequencing of the interactions or communications between different tasks, and the coordination of access to resources that are shared between tasks, are key concerns during the design of concurrent computing systems. In some concurrent computing systems communication between the concurrent components is hidden from the programmer, while in others it must be handled explicitly. Explicit communication can be divided into two classes:

Shared memory communication. Concurrent components communicate by altering the contents of shared memory location. This style of concurrent programming usually requires the application of some form of locking (e.g. mutexes (meaning mutual exclusion), semaphores, or monitors) to coordinate between threads. Shared memory communication can be achieved with the use of Software Transactional Memory (STM) [5, 6, 7]. Software Transactional Memory (STM) is an abstraction for concurrent communication mechanism analogous to database transactions for controlling access to shared memory. The main benefits of STM are composability and modularity. That is, by using STM one can write concurrent abstractions that can be easily composed with any other abstraction built using STM, without exposing the details of how the abstraction ensures safety.

Message Passing Communication. Concurrent components communicate by exchanging messages. The exchange of messages may be carried out asynchronously (sometimes referred to as "send and pray"), or one may use a rendezvous style in which the sender blocks until the message is received. Message-passing concurrency tends to be far easier to reason about than shared-memory concurrency, and is typically considered to be a more robust, although slower, form of concurrent programming. The most basic feature of concurrent programming is illustrated in Figure 1. The numbered nodes present instructions that need to be performed and as seen in the figure certain nodes must be executed simultaneously. Since most of the time intermediate results from the node operations are part of the same calculus this presents great challenge for practical systems. A wide variety of mathematical theories for understanding and analyzing message-passing systems are available, including the Actor model [8]. In computer science, the Actor model is a mathematical model of concurrent computation that treats "actors" as the universal primitives of concurrent digital computation: in response to a message that it receives, an actor can make local decisions, create more actors, send more messages, and determine how to respond to the next message received.

Such approach offers many advantages, namely increased application throughput – the number of tasks done in certain time period will increase; high responsiveness for input/output – input/output intensive applications mostly wait for input or output operations to complete; concurrent programming allows the time that would be spent waiting to be used for another task. It can be stated that there are more appropriate program structures – some problems and problem domains are well-suited to representation as concurrent tasks or processes.

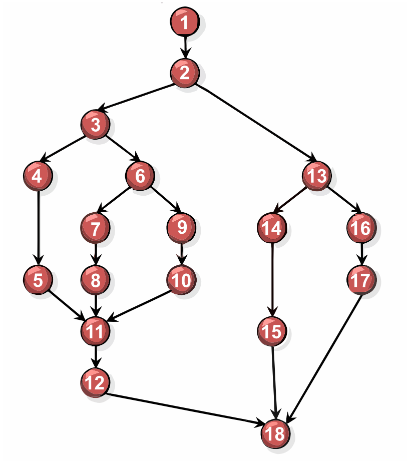

Fig. 1. The data flow of a software

3 COMMUNICATION

In case of distributed systems the performance of parallelization largely depends on the performance of the communication between the peers of the system. Two peers communicate by sending data to each other, therefore the performance of the peers depends on the processing of the data sent and received. The communication data contains the application data as well as the transfer layer data. It is important for the transfer layer to operate with small overhead and provide fast processing. Embedded systems have specific requirements. It is important that the communication meets these requirements.

The design of the presented method is focused around the possibility to support and execute high level optimizations and abstractions on the whole program. The graph-based software layout of the method provides the possibility to execute graph algorithms on the software architecture itself. The graph algorithms operate on the software's logical graph not the execution graph. This provides the possibility for higher level optimizations (super optimization). The architecture is designed to be easily modelable with a domain specific language. This domain specific language eases the development of the software, but its primary purpose is to provide information for higher level optimizations. It can be viewed as the logical description, documentation of the software. Based on the description language it is possible to generate the low level execution of the software, i.e. that it is not necessary to work at a low level during the development of the software. The development is concentrated around the logic of the application. It focuses on what is to be achieved instead of the small steps that need to be taken in order to get there.

4 EMBEDDED SYSTEMS AND CONCURRENT PROGRAMMING

The architecture of modern embedded systems is based on multi-core or multi-processor setups. This makes concurrent computing an important problem in the case of these systems as well. The existing algorithms and solutions for concurrency were not designed for embedded systems with resource constraints. In the case of real-time embedded systems it is necessary to meet time and resource constraints. It is important to create algorithms which prioritize these requirements. Also, it is vital to take human factor into consideration and simplify the development of concurrent applications as much as possible and help the transition from the sequential world to the parallel world. It is also important to have the possibility to trace and verify the created concurrent applications. The traditional methods used for parallel programming are not suitable for embedded systems because of the possibility of deadlocks. Deadlocks pose a serious problem for embedded systems [9], because they can cause huge losses. The methods shown in section II (Actor model and STM), which do not have deadlocks, have increased memory and processing requirements this also means that achieving real-time execution becomes harder due to the use of garbage collection. Using these methods and taking into account the requirements of embedded systems one can create a method which is easier to use than low-level threading and the resource requirements are negligible. In the development of concurrent software the primary affecting factor is not the method used for parallelization, but the possibility to parallelize the algorithms and the software itself. To create an efficient method for parallel programming, it is important to ease the process of parallelizing software and algorithms. To achieve this, the used method must force the user to a correct, concurrent approach of developing software. This has its drawbacks as well, since the user has to follow the rules set by the method.

The presented method has a steep learning curve, due to its requirements toward its usage (software architecture, algorithm implementations, data structures, resource management). On the other hand, these strict rules provide advantages to the users as well, both in correctness of the application and the speed of development. The created applications can be checked by verification algorithms and the integration of parts created by other users is provided by the method itself. The requirements of the method provide a solid base for the users. In the case of sequential applications the development, optimization and management is easier than in the case of concurrent applications. Imperative applications have a state when executed. This state can be viewed as the context of the application. The results produced by imperative applications are context-dependent. Imperative applications can produce different results for the same input because of different contexts. Sequential applications execute one action at a given moment with a given context. In the case of concurrent applications, at a given moment, one or more actions are executed within one or more contexts, where the contexts may affect each other. Concurrent applications can be decomposed into sequential applications which communicate with each other through their input, but their contexts are independent. This is the simplest and cleanest form of concurrent programming.

5 CONCURRENT PROGRAMMING FOR EMBEDDED SYSTEMS

Embedded systems are designed to execute specific tasks in a specific field. The tasks can range from processing to peripheral control. In the case of peripheral control, concurrent execution is not as important, in most cases the use of event-driven asynchronous execution or collective IO is a better solution [10]. In the case of data- and signal processing systems the parallelization of processing tasks and algorithms is important. It provides a significant advantage in scaling and increasing processing capabilities of the system. The importance of peripheral and resource management is present in data processing systems as well. The processing of the data and peripheral management needs to be synchronized. If we fail to synchronize the data acquisition with data processing the processing will be blocked until the necessary data are acquired; this means that the available resources are not being used effectively. The idea of the presented method is to separate the execution, data management and resource handling parts of the application. The presented method emphasizes data processing and is made up of separate modules. Every module has a specific task and can only communicate with one other module. These modules include peripheral/resource management module, data management module and the execution module. The execution module is a light weight thread, it does not have its own stack or heap. This is a requirement due to the resource constrains of embedded systems. If required, the stack or heap can be added into the components of the execution thread with to the possibility of extending the components of the execution thread with user-defined data structures. The main advantage of light weight threads is that they have small resource requirements and fast task switching capabilities [11, 12]. The execution module interacts with the data manager module which converts raw data to a specific data type and provides input for the execution module. The connection between the data manager and the execution module is based on the Actor model [8] which can be optimally implemented in this case, due to the restrictions put on the execution module which can only read and create new data (types) and cannot modify it. The execution module can be monolithic or modular. Modular composition is required for complex threads were processing is coupled with actions (IO). The execution threads can be built up from two kinds of components, processing and execution/action ones. The component used in the execution module is a type which for a given input type 'a' creates a given type 'b'.

This operation will always give the same result for the same input. The processing component is referentially transparent, i.e. it does not support destructive actions [13]. The type variables 'a' and 'b' can have the same types.

The action component is similar to the processing component, it is usable in cases where destructive actions must be supported. These components request the execution of specific actions which are received and executed by a transactional unit. The design of the transactional mechanism is based on transactions, just as in software transactional memory. The threads in the execution module are not connected to each other. It is possible to achieve interaction between the threads. One or more execution threads can be joined with the use of the reduce component. The reduce component iterates through the values of the given threads, merging them into one component or value. The merging algorithm is specified by the user, as well as the order of the merging. The joining of the threads follows the MapReduce model, where the map functions correspond to the threads and the reduce function corresponds to the merging algorithm provided by the user [14]. The method introduced in this paper is usable for concurrent programming in real-time embedded systems as well. The complexities of the algorithms used in the method are linear in the worst case. The priority of threads can be specified, i.e. the order of execution can be predetermined. It is possible to calculate the amount of time required to execute a specific action. This way the created systems can be deterministic.

Threads can be separated into two parts. The two parts create a client server architecture, where the server is the data manager and the client is the actions/steps of the thread. The job of the server (producer) is to provide the client (consumer) with data. The server part sends the data to the client part. The server part protects the system from possible collisions due to concurrent access or request to resources. The client part has a simple design – it is made up of processing steps and actions. The job of the asynchronous resource manager is to provide safe access to resources for the server part of the threads. The resource manager does not check the integrity of data, its only job is to provide the execution threads server part with raw data. Parallelization of software is not trivial in most cases [15]. The method presented in the paper takes this fact into consideration. It is important that the parallelizable and sequential parts of the software can be easily synchronizable. The presented view of software (as seen in Figure 2) is easily implementable into the model of the presented method. Based on the data flow of the software, it is possible to implement it into the model of the presented method for concurrency.

6 DSP DSL

To apply the presented method for digital signal processing applications at a higher level, a domain specific language needs to be designed on top of the software framework. This will make it possible to efficiently apply concurrency to digital signal processing applications. The implemented software framework for the presented method supports the usage of custom defined DSLs. The first step is to identify the basic building blocks for DSP applications and present it in a form which makes it possible to easily apply function composition on them. This makes it easier to combine them together. The basic operations in DSP applications are basic mathematical operations, convolution, transformations, and FFT. From these operations, we can implement the DSP logic of an application with composition. By connecting these components together it is possible to support operations such as digital filters and so on. The presented method provides the composition and abstraction features, the user only needs to define the basic operations.

7 DSP BLOCKS

The most basic DSP elements are low pass filters. In the next segment a short features overview of the FIR and IIR filters is given. FIR filters have the following characteristics:

- stable, because the transfer function has no poles,
- the phase characteristic is linear,
- linear amplitude characteristics,
- it has no feedback loop,
- impulse response is finite.

As opposed to that IIR filters are as follows:

- impulse response is infinite,
- linear characteristics,
- time invariant,
- transfer function has poles and zeroes,
- the most used forms for IIR filter are discrete, cascade and parallel form.

The IIR filter design has been realized by the translation of the already known analog transfer functions into discrete transfer functions.

8 ALGORITHM DEVELOPMENT

For algorithm development high-level tools were used such as Matlab, Simulink and C++ (cf. Figure 2). DSP algorithm simulation has been performed on personal computer for performance analysis. The advantages of the development on PC:

- high-level programming tools allow short development time and application of the C language on multiple DSP platforms,
- easy monitoring and modification of the program written in a higher level programming language by integrated development tools,
- input-output operations were easily obtained through files on the disc which lead to simplified system analysis,
- computer simulation allowed to use the of data formats and floating point operations which greatly simplified the development,
- Matlab and Simulink software make it easier to develop fixed point DSP blocks.

9 SOFTWARE DEVELOPMENT

Quality DSP software has four characteristics:

- reliability,
- maintenance,
- ability to expand,
- performance.

The program is reliable if it crashes rarely or never. Maintenance allows easy correction inside the program and the best maintenance allows the program to be corrected by programmers beside the manufacturer [16]. In order to create portability to other hardware platforms it is necessary to realize the function for further improvement of the program. The ability to expand allows new features and corrections to be introduced into the program. If a program, after years of exploitation in different platforms, still works without errors, it can be considered to be reliable.

A good DSP program often contains more simple functions which serve one purpose and can easily be used in other programs. It is wise to avoid exotic programming tricks because this often has a great impact on program reliability. In DSP applications hardware and software development is performed simultaneously, as shown in Figure 3. The DSP programmer must be capable of solving problems on hardware and software level [17]. Since the price of the hardware has decreased dramatically over the past few years, the greatest part in the price of the DSP development is in software development. The software application life cycle involves the realization of the software project:

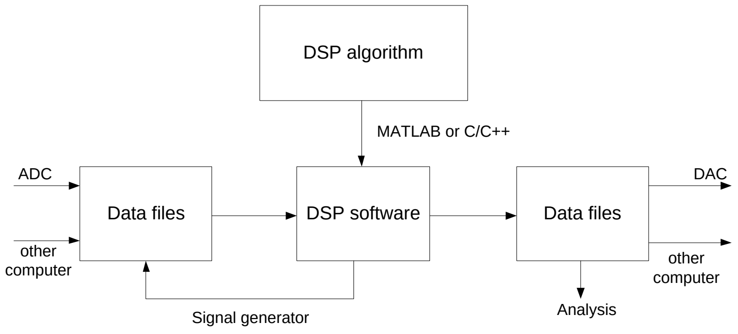

Fig. 2. Software development on personal computer

- project definition,
- detailed specifications,
- coding and modular testing,
- integration,
- maintenance,
- system testing.

Software maintenance represents a significant part of the development. This involves development of the functions and correction of the errors spotted during software exploitation. In programming it is vital to apply a structurally well-documented approach to programming right from the beginning. During coding it is important to create a basic specification for tasks in signal processing. This specification must contain the basic algorithm and task description, memory requirements, size limitations, etc. With well-tested specification it was possible to locate errors before coding has even started. DSP program coding is an interactive process and therefore it needs the ability of systematic testing during the time of coding through integrated development tools. Modular or partial development was responsible for this process. Every module can be tested separately which enhances the probability that the complete system will be error-free during system integration. When it comes to software development, there are two types of DSP tools:

- assembly language,
- C/C++ language.

Assembly language represents the mnemonic code which was used directly by the processor [18]. One of the positive features of the assembly language is that it offers complete control over the coding process to the programmer and therefore it is possible to develop most efficient programs. The downside is that it takes a long time and it is very complex. Often the ideal solution is to combine assembly and C language blocks. The basic program and management is written in C language, while

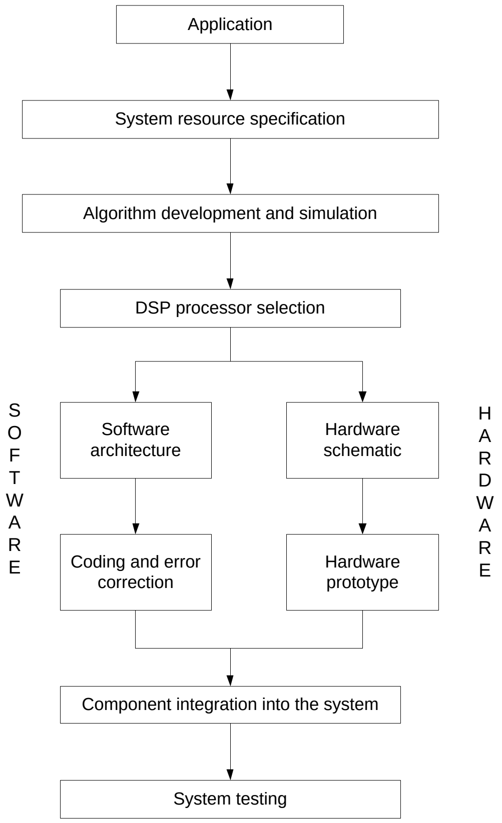
Application
System resource specification
Algorithm development and simulation
DSP processor selection
SOFTWARE
Software architecture
Hardware schematic
HARDWARE
Coding and error correction
Hardware prototype
Component integration into the system
System testing

Fig. 3. Simplified diagram of a DSP system

the critical portions are written in assembly language. In this combined surrounding assembly functions are called from C program. Inside one file it is possible to develop "hand optimized" function that could be inserted into other programs.

10 DSP SOFTWARE DEVELOPMENT

The general task of the DSP software was data processing, synchronization of data flows, peripheral control and communication between units that constituted the system. In software development and addition of new, as well as enhancements of old algorithms was obtained by the modular expansion feature. Programs are formed through elements that are divided into the framework and add-ons [19]. The framework can be expanded by user written units. The logic behind software and hardware must be independent so the user can concentrate on the software part; this is established by the framework. Functionality is obtained by add-ons. Add-ons are linked to the framework through the use of components. The framework incorporates only basic types which aid algorithms, e.g. they are used to develop additional units. Additional units can be divided into two major groups:

- data processing units,
- control flow units.

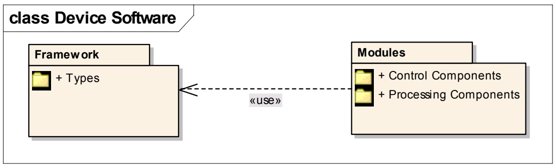

Fig. 4. Link between framework and module

The framework architecture is object-oriented and it was implemented in ANSI C programming language. Figure 4 shows dependence between different parts of the system. The central unit was designed for control purposes. The generator feeds the central unit with the data and then that data is routed to the appropriate unit for data processing. Communication between the units is asynchronous and event driven. The control unit is connected to every unit and subunit in the system and has full control over the functions of every unit (cf. Figure 5). The subunit can communicate with the central unit in the form of messages. The input message in basic mode of operation instantly enters the processing block but it is possible to store the data for later processing. Every message has a separate handler which can be substituted even while the program is running. The message content can be expanded. The control unit presents a pipeline for data processing with the help of the component manipulator. The control component incorporates all components and their descriptions, e.g. identification and type. The special unit is responsible for processing and pipeline construction. The specific parts of hardware are separated from the framework core, e.g. power source management. The framework uses data structures for power source management and for communication between units. Data structures use the framework manipulator to perform memory allocation.

11 EXPERIMENTAL RESULTS

The purpose of the interactive development system in real time is a comparison between simulations results with every stage of development of the elements used in the targeted processor. This method can be used in the interactive development for testing and verification in a step by step way. The development system is designed for the application in the area of digital signal processing but it can be expanded to other fields [20]. The applications used for digital signal processing are resource demanding because the calculations must be performed with high precision. As

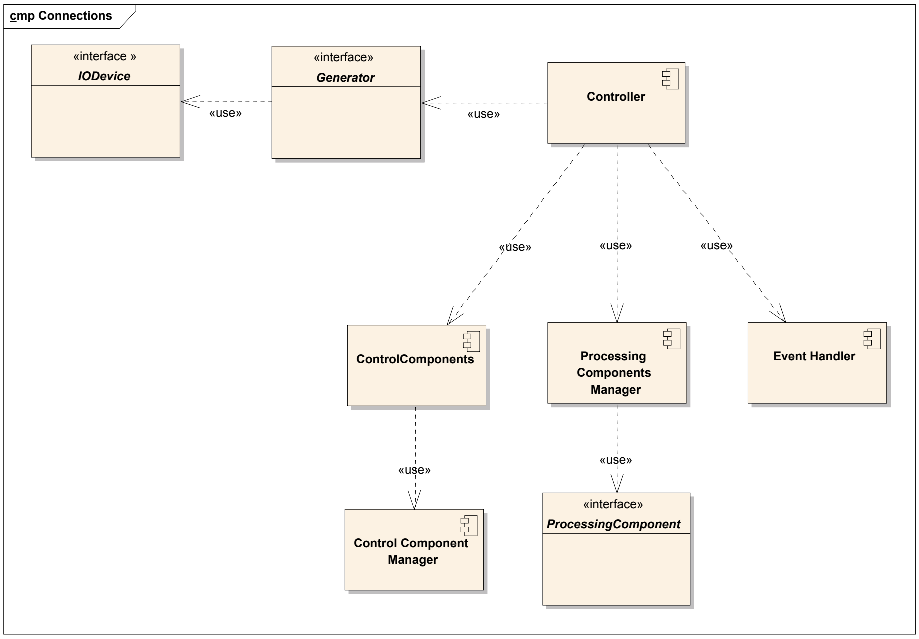

Fig. 5. Framework block diagram

an example, FIR filter was used. The design of the filter was performed in Matlab. The obtained results were valid and within the targeted boundaries of the given task. The development system was used to verify that the filter will behave in the same manner in the hardware processor as it does in the simulation. This system compares the characteristics of the given filter between the implementation in the targeted processor and the Matlab simulation (cf. Figure 6). The results show that there was a difference between the two instances because the targeted processor stored filter coefficients differently and also lacked support for floating point operations [21]. This problem was easily spotted during the early phase of the development by proposed software. The processor used for this experiment was TMS320C5510 located on DSK development platform TMS120VC5510 from Texas Instruments.

The modeling which was used in this development system is very effective for the research and development of the algorithms because the developed UML diagram is easy to implement in the software as the pipeline for data processing. Any modification performed on the UML diagram was instantly transferred into the pipeline for data processing. This produced a great advantage as the research and testing of the algorithm are on a high level and results are obtained on real hardware.

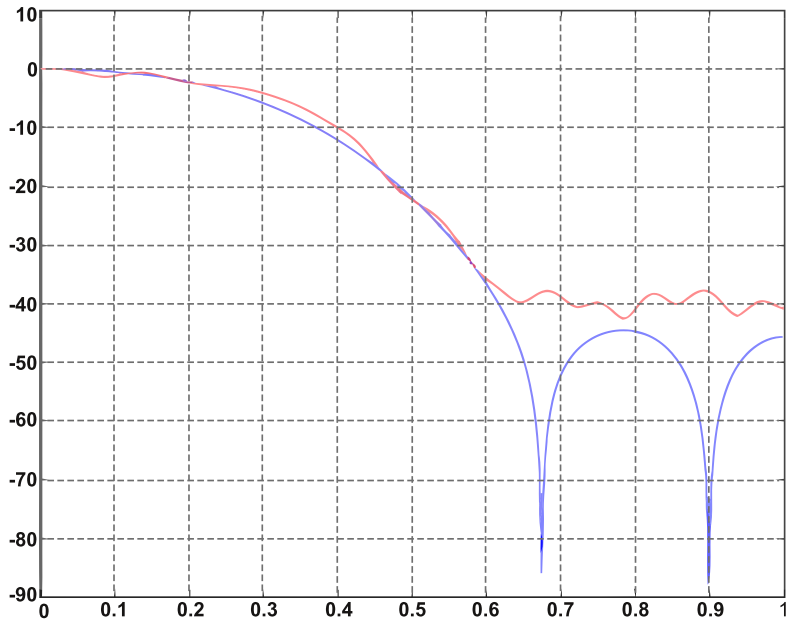

Fig. 6. Difference between filter in Matlab (blue) and on DSP processor (red)

12 APPLICATION IN EDUCATION

Because of its interactive and graphical nature, this software environment was the ideal candidate for application in an educational course in digital signal processing [22, 23]. Before using these tools the students had 2 hours of theoretical lectures followed by 2 hours of introduction in computer laboratory. Using this software environment student can do the following:

- understand the principles of FIR and IIR filters,
- correlate filter parameters with an actual circuit,
- make realistic experiments in laboratory,
- design a filter with the desired characteristics,
- improve knowledge in the least amount of time.

The results have been compared before and after using the software. Every student was given an evaluation sheet; the answers are shown in Table 1. The data presented in the table was gathered over two years, 2008 and 2009. Total number of the students participating in the experiment is 149. Through the analysis of the evaluation sheets it was determined that the new software tools have improved the students' scores. The visual and interactive tools have not only improved the students' learning rate but also had impact on raising the students' interest in the subject.

Question	Plenty/ Very well	Sufficient/ Well	Some	A little
How much prior knowledge of the filter design did you have before attending this lab?	2	9	59	79
Did you understand the mathematical models of the filters before attending this lab?	8	43	69	29
Did the tool motivate you to work in the lab?	38	51	37	23
How much experience did you have before attending this lab?	51	35	52	11
Did the experiments in the lab with the tool fit with your previous knowledge?	15	80	39	15
Is the tool easy to use and user friendly?	68	55	24	2
How much do you feel you have benefited from the tool?	80	48	14	7
How do you judge your own work with the tool at this time?	94	36	14	5
Is this lab session proper demonstration of the theoretical session?	62	59	24	4

Table 1. Evaluation sheet

13 CONCLUSION

Concurrent programming is complex and hard to achieve. In most cases the parallelization of software is not a straightforward and easy task. The realized concurrent programs usually have safety and performance issues. For embedded systems the existing parallelization algorithms and solutions are not optimal due to resource requirements and safety issues. The goal is to realize such a solution for concurrent programming, which is optimal for embedded systems, and helps and simplifies the development of concurrent programs. The key to successful development of parallel programs is in the realization of tools which take into consideration the human factors and aspects of parallel development. The model presented in this paper builds on the advantages of existing parallelization algorithms with human factor as its primary deciding factor. At this moment the development systems very poorly satisfy today's strict and terms in the field of digital signal processing. In addition, tasks in digital processing are getting more complex every year. Because of this it is necessary to develop new methods which would concentrate on fast development on real hardware. In this paper one possible solution is presented in the form of an interactive system with elements developed in a high-level programming language. The method of interactive development in real time speeded up the development process because it allowed algorithm testing and correcting in the early stages of the development. Integration into already available software tools (Matlab) simplifies the adaptation of this method. Furthermore, it was shown that this approach not only influenced real hardware development but also improved the learning process with the students. This way the students benefited from having a tool that can be used both for simulation in the laboratory and as an aid in real hardware development and testing at their job after graduation.

REFERENCES

[1] Daniel, W. H.: Circuit Simulation as an Aid in Teaching the Principles of Power Electronics. IEEE Trans. Educ., Vol. 36, 1993, pp. 10–16.

[2] Maravić Čisar, S.—Pinter, R.—Radosav, D.—Cisar, P.: Software Visualization: The Educational Tool to Enhance Student Learning. Proceedings of 33rd International Convention (MIPRO 2010), Computers in Education, Vol. IV, May 24–28, 2010, Opatija (Croatia) 2010, ISSN 1847-3938, ISBN 978-953-233-054-0, pp. 234–238.

[3] Szedmina, L.: Could You Check This, Please? Experiences in a Bilingual Environment. Acta Polytechnica, Vol. 7, 2010, No. 2, ISSN 1785-8860, pp. 155–162.

[4] Mester, G.: Would we Realise the Aims of the Lisbon Strategy 2000 in Higher Education of Europe? Technical University of Applied Sciences, TH Wildau, Berlin 2009.

[5] Harris, T.—Marlow, S.—Peyton Jones, S.—Herlihy, M.: Composable Memory Transactions. Proceedings of the 10th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming 2005, pp. 48–60.

[6] Discolo, A.—Harris, T.—Marlow, S.—Peyton Jones, S.—Singh, S.: Lock-Free Data Structures using STMs in Haskell. Functional and Logic Programming 2006, pp. 65–80.

[7] Harris, T.—Peyton Jones, S.: Transactional Memory with Data Invariants. ACM SIGPLAN Workshop on Transactional Computing 2006.

[8] Baran, P.: On Distributed Communications Networks. IEEE Transactions on Communications Systems, Vol. 12, 1964, No. 1, pp. 1–9.

[9] Sanchez, C.: Deadlock Avoidance for Distributed Real-Time and Embedded Systems. Dissertation, Department of Computer Science of Stanford University 2007.

[10] Yorozu, Y.—Hirano, M.—Oka, K.—Tagawa, Y.: MTIO. A Multi-Threaded Parallel I/O System. Proceedings of 11th International Parallel Processing Symposium 1997, pp. 368–373.

[11] Narlikar, G. J.—Blelloch, G. E.: Space-Efficient Scheduling of Nested Parallelism. ACM Transactions on Programming Languages and Systems 1999, pp. 138–173.

[12] Narlikar, G. J.—Blelloch, G. E.: Space-Efficient Implementation of Nested Parallelism. Proceedings of the Sixth ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming 1997.

[13] Bondavalli, A.—Simoncini, L.: Functional Paradigm for Designing Dependable Large-Scale Parallel Computing Systems. Proceedings of International Symposium on Autonomous Decentralized Systems (ISADS 93) 1993, pp. 108–114.

[14] Dean, J.—Ghemawat, S.: Simplified Data Processing on Large Clusters. Sixth Symposium on Operating System Design and Implementation (OSDI 04) 2004.

[15] Amdahl, G.: Validity of the Single Processor Approach to Achieving Large-Scale Computing Capabilities. AFIPS Conference Proceedings 1967, pp. 483–485.

[16] Rodgers, P. D.: Improvements in Multiprocessor System Design. ACM SIGARCH Computer Architecture News, Vol. 13, 1985, No. 3, pp. 225–231.

[17] Kuo, S. M.—Lee, B. H.—Tian, W.: Real-Time Digital Signal Processing Implementations and Applications. Second Edition, Wiley 2006, ISBN10: 0470014954.

[18] Xiong, J.—Johnson, J.—Johnson, R.—Padua, D.: SPL: A Language and Compiler for DSP Algorithms. Conference on Programming Language Design and Implementation 2001, pp. 298–308.

[19] Bhattacharyya, S. S.: Compiling Dataflow Programs for Digital Signal Processing. Ph. D. thesis, Department of Electrical Engineering and Computer Sciences, University of California at Berkeley 1994.

[20] Buck, I.—Foley, T.—Horn, D.—Sugerman, J.—Hanrahan, P.: Book for GPUs: Stream Computing on Graphics Hardware. ACM Transactions on Graphics, Proceedings of SIGGRAPH 2004, August 2004.

[21] Chassaing, R.: Digital Signal Processing and Applications with the C6713 and C6416 DSK. Wiley Interscience 2004, ISBN 0-471-69007-4.

[22] Elmas, C.—Sönmez, Y.: An Educational Tool for Power Electronics Circuits. Computer Applications in Engineering Education, Vol. 18, 2010, No. 1, pp. 157–165.

[23] Samuelis, L.—Szabó, C.: Automatic the Measurement of the Complexity of Students Assignments. Knowledge Technologies and Applications, Networking Centre of High Quality Research on Knowledge Technologies, Košice – Budapest, SzTAKI, 2007, pp. 116–125.

Anita Sabo graduated in 2005 from Technical University of Novi Sad. She received her Magister degree in micro-computing electronics and her Ph. D. degree in microcomputing electronics from Technical University of Novi Sad in 2008 and 2012, respectively. Currently she is a lecturer and member of the Department of Informatics at the Subotica Tech-College of Applied Sciences.

Bojan Kuljić received his B. Sc. degree in electronics and telecommunications from Subotica Tech-College of Applied Sciences in 2004. He is currently on M. Sc. studies at Technical University of Novi Sad and has been working at Subotica Tech since November 2011.

Tibor Szakáll received his B. Sc. degree from Subotica Tech-College of Applied Sciences in 1992. In 2004 he graduated from Technical Faculty "Mihajlo Pupin" in Zrenjanin. In 2012 he received his Magister degree from the same faculty. Since February 1992 he has been working at Subotica Tech-College of Applied Sciences; his topics of interest include electronics, telecommunications and informatics.