Rapid Prototyping of flexible Embedded Systems on multi-DSP Architectures

Bernhard Rinner, Martin Schmid and Reinhold Weiss
Institut für Technische Informatik
Technische Universität Graz
Inffeldgasse 16/1, 8010 Graz, Austria
[rinner, schmid, rweiss]@iti.tu-graz.ac.at

Abstract

The functionality of a typical embedded system is specified once at design time and cannot be altered later during the whole mission period. There are, however, a number of important application domains that ask for both flexibility and availability. In such a flexible embedded system the functionality can be modified while the application is running.

This paper presents a rapid prototyping environment for flexible embedded systems on multi-DSP architectures. This prototyping environment automatically maps and schedules an application onto a multi-DSP architecture and introduces a special, lightweight reconfiguration environment onto the target platform. A running multi-DSP application can, therefore, be modified by reconfiguring software tasks. By using our prototyping environment the modified application can be tested, simulated and emulated prior to the implementation on the target.

keywords: embedded system; multi-DSP architectures; task reconfiguration; testability; rapid prototyping

1 Introduction

Digital signal processing (DSP) functionality is increasingly embedded into more and more applications. Often multi-DSP architectures are used to keep pace with the ever increasing performance requirements. The functionality of such a typical embedded systems is specified once at design time and cannot be altered later during the whole mission period. There are, however, a number of important application domains that ask for both *flexibility* and *availability*, i.e., systems operating in remote and hostile environments, systems with extraordinary long mission periods and embedded systems that must not be shut down during update periods. Examples for such embedded systems include satellite receivers with update functionality, high-level controllers for technical processes and communication systems.

In a flexible embedded system, a potential modification of the functionality must already be taken into account at the initial design and implementation. Two aspects are especially relevant in this context.

Support for design and implementation Even with a fixed functionality the development of an embedded system is often supported by tools for design automation and prototyping [2]. Support for design and implementation is almost mandatory for flexible embedded systems, i.e., the modified application must be thoroughly simulated, tested and emulated before it is downloaded to the target system.

Mechanism for reconfiguration In order to modify a running application on an embedded system a special download mechanism is necessary. This *reconfiguration environment* may not interfere with the DSP code execution and must be a lightweight and predictable component since resources on an embedded system are always rare.

This paper focuses on increasing the flexibility of embedded multi-DSP systems. The key goals of this research are (i) to support the design and implementation of flexible embedded systems by a prototyping environment, (ii) to integrate a mechanism for the modification of the functionality on a running application, and (iii) to improve the testability of the embedded system. This work is based on PEPSY – a **p**rototyping **e**nvironment for multi-DS**P** **sy**stems [6, 7]. Given a specification, i.e., an application model, a hardware model and mapping constraints, PEPSY automatically maps and schedules the DSP application onto the multi-DSP system and synthesizes the complete code for each processor.

PEPSY has been extended by a reconfiguration mechanism that allows to add, delete or modify software tasks. This reconfiguration is completely integrated into PEPSY, thus all of PEPSY's main features such as optimization, performance prediction and code generation, can be used when modifying the functionality. Via the reconfiguration environment, the communication between software tasks can now be inspected. All of these actions are performed while

Proceedings of the Design,Automation and Test in Europe Conference and Exhibition (DATE'03)
1530-1591/03 $17.00 © 2003 IEEE

the application is running on the target system enabling *functional* as well as *performance* testing at early stages in the development process.

The rest of this paper is organized as follows. Section 2 presents the design flow for flexible embedded systems using PEPSY. Section 3 describes the reconfiguration mechanism implemented on the target system and the host. Section 4 presents experimental results. Section 5 discusses related and future work.

2 Automated Design of Multi-DSP Systems

Figure 1 depicts the overall design flow for the automated design and implementation of multi-DSP applications. This design flow is integrated into the prototyping environment PEPSY [8]. During the initial design, PEPSY automates the parallelization of data-flow oriented applications onto heterogeneous multi-processor systems, i.e., it computes an optimized multi-processor implementation given a specification, resource constraints and an objective function.

The specification of the design problem is based on two models. The application model describes the overall application by an extended data-flow graph G_A [1], i.e., the nodes of this graph represent (functional) tasks of the application and the arcs represent data dependencies between the tasks. The hardware model describes the multi-processor system onto which the application is mapped. Each processing element is represented by a node of this graph G_H. Physical point-to-point connections are described by the arcs. Restrictions on the mapping of tasks onto processing elements may be specified by mapping constraints m.

The optimizer computes an optimized multi-processor implementation I, i.e., it approximates an optimal mapping and scheduling for all tasks given the specification subject to an objective function and resource constraints. The implementation further includes code for the reconfiguration environment. The mapping and scheduling generated by the optimizer consist of a task list for each processor. This task list includes the application tasks as well as the sender and receiver tasks introduced for inter-processor communication. For each task, start and end times are predicted by the optimizer using a communication model for buffered data transfer [6]. This communication model is the basis of PEPSY's performance prediction. Additional parameters such as the task's memory requirement and deadline as well as the processor's memory capacity may be specified and used to express more complex design goals than the standard multi-processor scheduling problem [8].

The final step in this prototyping environment is automatic code generation and synthesis. The goal of this step is to generate the multi-processor application, i.e., to generate, compile, link and download the complete code.

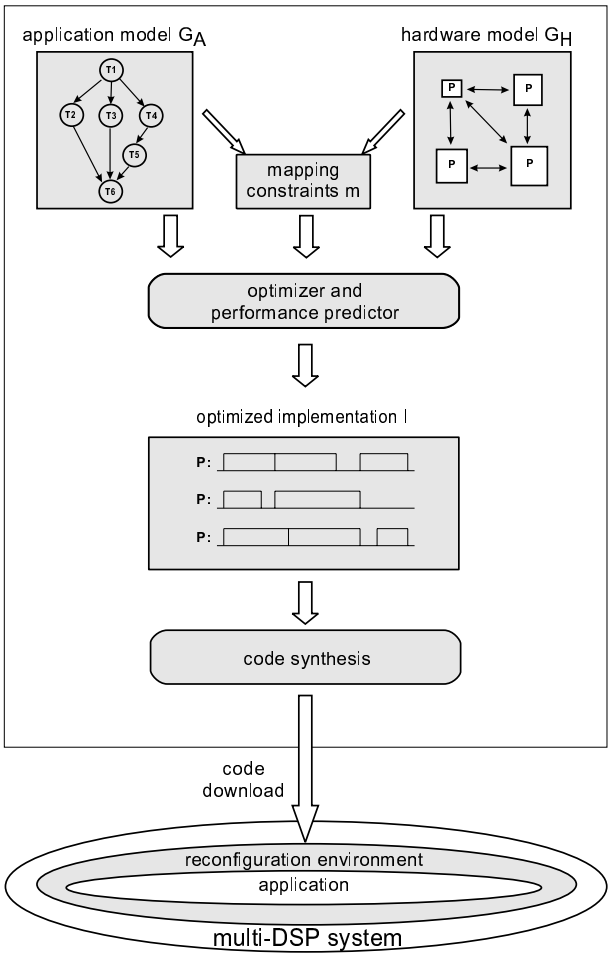

Figure 1. Design flow of our PEPSY prototyping environment.

3 Reconfiguring Embedded Applications

In order to extent PEPSY to the reconfiguration of flexible embedded systems, two modifications are necessary. First, a *reconfiguration environment* must be introduced on the target system. Second, a dedicated interface must be integrated on the host system. This *host tool* controls the data transfer between host and target and communicates with PEPSY (Figure 2).

3.1 Reconfiguration Environment

The reconfiguration environment is a software component on the target platform and provides the basic functionalities (i) to read communication data from the target system to debug the application, (ii) to load data onto the target hardware, and (iii) to load code to modify a running application on the target system at the task level. These modifica-

Proceedings of the Design,Automation and Test in Europe Conference and Exhibition (DATE'03)
1530-1591/03 $17.00 © 2003 IEEE

tions must be performed without any interference with the DSP code execution in order to guarantee the availability of the prototype hardware.

In the context of PEPSY's design flow, the reconfiguration environment has to fulfill further strong requirements:

1. The environment's functionality must be an easy-to-add and lightweight component.
2. The environment must be scalable, i.e., the functionality must be independent of the number of DSPs on the target system.
3. The environment's execution time must be small and must be included into PEPSY's performance prediction.
4. The data transfer between host and target system during reconfiguration must be secured in order to detect transmission faults.
5. Starvation and deadlock situations during reconfiguration must be avoided.

These requirements are fulfilled by introducing a dedicated kernel task on each processor at the initial design. This kernel task implements the functionality of the reconfiguration environment and is executed at the beginning of each loop cycle prior to any application tasks. During its execution the kernel performs the necessary operations and terminates strictly after a predetermined time. To guarantee a small lightweight kernel, only the necessary functionality for communication and data up- and download is implemented and only a small portion of data can be transferred between host and DSP within a single loop iteration. The memory management of all DSPs is handled by the host.

The kernel task has been implemented on Texas Instruments TMS320C40 DSPs. It has been entirely written in assembler to provide a small footprint and fast execution. The transmission protocol implemented provides the functionality to (i) request the status of the kernel task, (ii) to initialize the download of code, (iii) to download a data packet to memory, (iv) to start a task, and (v) to send memory contents to the host.

A memory protection mechanism has been implemented to protect the kernel code and the auxiliary memory locations from being overwritten by the process of data download onto the target system.

Communication between the kernel tasks and the host is realized in a master/slave structure. All (reconfiguration) communication is initiated by the host, and the corresponding data is sent to the kernel task mapped on the DSP directly connected to the host. Data for kernel tasks on all other DSPs is routed through a tree-structured network. DSP kernel tasks receive commands or transmit them

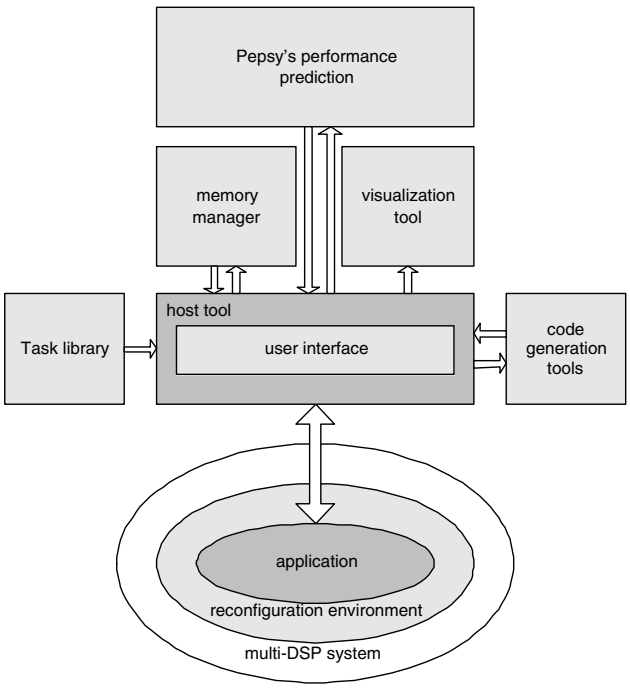

Figure 2. Integration of the reconfiguration environment and the host tool into PEPSY.

to the next DSP in the tree structure. The advantage of this structure is that starvation or deadlock situations caused by blocking communications can be avoided. If the host initiates a data upload from a DSP, it first sends the command to the dedicated DSP and waits until the data upload has completed. No other communication is initiated between the request and completion of the upload.

The transmitted data is partitioned into packets of size not larger than the size of the communication buffers (16 words for the TMS320C40). By transferring only a single packet within a single kernel task, communication can't get blocked. The transfer of a single packet is secured by a CRC. In case of a transmission failure, a retransmission is initiated at the next kernel tasks call. The tree structure is also scalable. Additional DSPs can be simply added to the leafs of the tree.

3.2 Host Tool

The host tool is able to display and to use all features of the kernel task in order to extend the system to the desired level of functionality. This tool can directly communicate with the kernel of an individual DSP; commands are then executed by the kernel. Afterwards a return packet is sent back to the host.

The host tool introduces an access layer between basic

Proceedings of the Design,Automation and Test in Europe Conference and Exhibition (DATE'03)
1530-1591/03 $17.00 © 2003 IEEE

protocol commands and PEPSY (see Figure 2). It allows to directly download data and code of tasks stored in the common object file format (COFF). The required memory management of the DSPs is handled separately at the host. Before downloading a task, the memory manager determines free locations for code and data sections. A linker command file is automatically generated and the sections are linked to these memory areas.

For the purpose of observation and debugging, the tool loads communication data from the target hardware. To address communication data, the user has to specify a communication buffer. First, the buffer's address is obtained from the memory manager. Subsequently, the host sends a command to the DSP containing the buffer address to upload a portion of data starting at the buffer address. The maximum size of buffer data to be inspected is 16, which corresponds to the packet size for the data transfer. Finally, the data can be visualized online or stored for later analysis.

In the case of reconfiguration, a task is transferred to a DSP and the executive's loop is altered. At the beginning, the user defines the position of the newly-inserted task in the schedule of the application and the accessed communication buffers. PEPSY's performance prediction then determines the communication and computation delays for this altered application. The memory manager determines free memory space for the task's code and data memory demands. Task information is retrieved from a task library file. A specially generated linker command file addresses the free memory space on the DSP. Afterwards, the code generation tools produce a file in the common object file format (COFF), which can be directly loaded onto the DSP's memory by the host tool. Subsequently, an altered executive loop is generated and loaded onto the DSP addressing the newly-inserted task in the favored schedule. Finally, the kernel task switches from the old to the altered executive loop.

We have measured the maximum kernel execution time as about 5 μs during a data transfer. The transfer of a single word to any DSP requires at most 0.25 μs.

4 Experimental Results

We demonstrate the modification of the functionality of a flexible embedded system using PEPSY. A streaming filter example on a multi-DSP architecture serves as example.

The target system for this experiment consists of four TMS320C40 DSPs from Texas Instruments. As shown in Figure 3 each processor of this multi-DSP system is connected to each other. Communication to the host is handled via DSP #1. This DSP is further connected to a data acquisition board with a digital/analog converter (DAC) and an analog/digital converter (ADC). The sampling rate of this data acquisition board can be set up to 100 kHz. The cycle

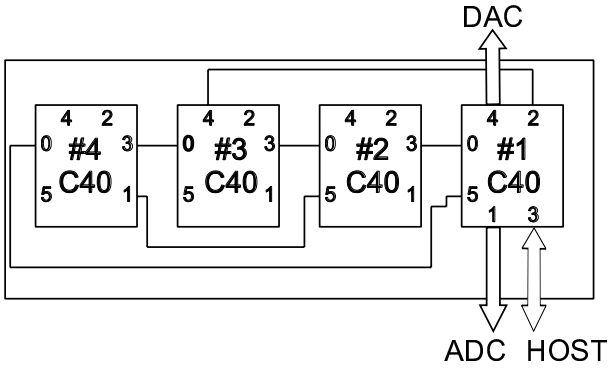

Figure 3. The multi-DSP system consisting of 4 TMS320C40 processors used as target platform for the experimental evaluation.

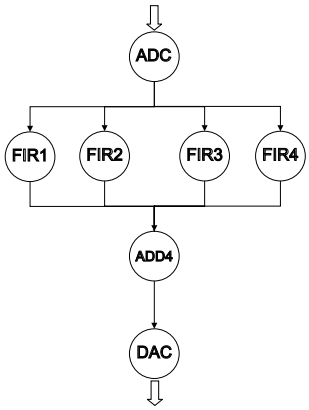

Figure 4. Task graph of the application example.

period of each DSP is 40 ns.

Figure 4 depicts the task graph of the filter example. At each iteration through this graph a sample data word is read from the data acquisition board by the ADC task. In the initial design this word is transmitted to a parallel filter bank consisting of four FIR filters. The individual filter outputs are summarized in task ADD4; the overall result is then delivered to the data acquisition board by the DAC task. The maximum execution time for a single iteration is given by 125 μs determined by the sampling rate of the communication system of 8 kHz.

The hardware model G_H which is derived from Figure 3 and the application model G_A which is derived from Figure 4 are the starting point for the optimization step in PEPSY.

The result of the optimization step is shown in Figure 5. In this gantt diagram the (static) task schedule is shown

Proceedings of the Design,Automation and Test in Europe Conference and Exhibition (DATE'03)
1530-1591/03 $17.00 © 2003 IEEE

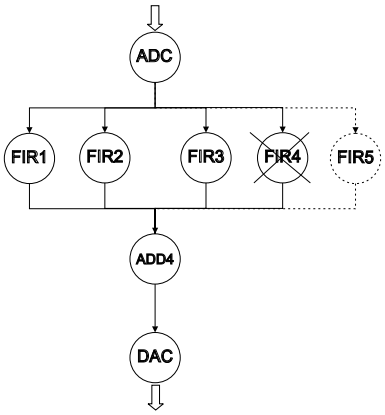

Figure 6. Modified task graph for redesign.

for each DSP. Note that PEPSY automatically inserts all required communication tasks denoted by R and S, respectively. The kernel task K is introduced at the beginning of each schedule. The kernel tasks are part of the reconfiguration environment. PEPSY's performance estimation results in an overall execution time of 82.2 μs. Thus, the real-time constraint on the overall execution time for a single iteration (125 μs) is not violated.

PEPSY finally synthesizes the complete code for the executive function as well as compiles, links and downloads the application onto the target system. The measured overall execution time is 85 μs which is very close to the estimated time.

Figure 6 depicts a modification of the functionality in the embedded multi-DSP application. A new filter task FIR5 is added to the filter bank which replaces filter task FIR4. The number of execution cycles for task FIR5 is 380.

The schedule derived by PEPSY's performance prediction is depicted in Figure 7. The delays are similar to the ones from the initial design. PEPSY's prediction of the overall execution time is 82,2 μs. Since FIR5 terminates before the communication with DSP #4 is started, the replacement of FIR4 with FIR5 does not result in an increase of the overall execution time. PEPSY automatically generates the executive function and synthesizes the executive and the task FIR5. The memory map for this tasks is managed by the memory manager.

The task reconfiguration is controlled by the host tool and consists of the following steps:[1]

1. The code and data for task FIR5 is loaded into a free memory area of DSP #1.

2. The modified code of the executive function is loaded onto DSP #1.

3. The kernel task of the reconfiguration environment starts the altered executive function.

The overall execution time for the modified filter application on the multi-DSP system has been measured as 85 μs which is the same small deviation from the estimation as in the initial example.

5 Discussion

In this paper we have presented our improved prototyping system for the design and implementation of flexible embedded systems on multi-DSP architectures. The primary goal of this research is to test, to predict the performance and to emulate a modified multi-DSP application prior to the implementation on the target. The prototyping environment PEPSY is now able (i) to observe communication data between tasks, (ii) to modify communication data between tasks and (iii) to reconfigure tasks while the overall application is running at the target platform.

There are related prototyping systems for digital signal processing applications known in the literature. These prototyping systems lack, however, in the ability to support functional modifications. Madisetti [5] presented the state of the art and identified future challenges in prototyping large DSP systems in the mid nineties. Fresse et al. [3] have developed a prototyping environment for a multi-DSP system targeted for image processing applications. Their environment requires a functional description given by dataflow graph and generates a parallel implementation onto a heterogeneous target platform. The GRAPE-II [4] environment uses synchronous and cyclo-static data flow graphs as application models. Grape-II maps and schedules the application tasks onto a heterogeneous target system comprised of digital signal processors and dedicated hardware (FPGA).

Some operating systems in the area of DSP also support some mechanisms for testing the overall application, e.g., Virtuoso [9]. A problem with operating systems is often that they offer only very limited access to a running multi-DSP application and introduce a (large) memory and performance overhead into the application. On the other hand, the overhead introduced by PEPSY's reprogramming environment is very small compared to operating system (kernels).

Future work of this research is focused on (i) improving the reprogramming environment, (ii) porting the extended PEPSY framework to different target platforms, (iii) applying our improved prototyping framework on various applications and (iv) extend the design and reprogramming process of PEPSY to heterogeneous systems with DSPs and FPGAs.

[1] Note that in this example reconfiguration is only required on DSP #1.

Proceedings of the Design,Automation and Test in Europe Conference and Exhibition (DATE'03)
1530-1591/03 $17.00 © 2003 **IEEE**

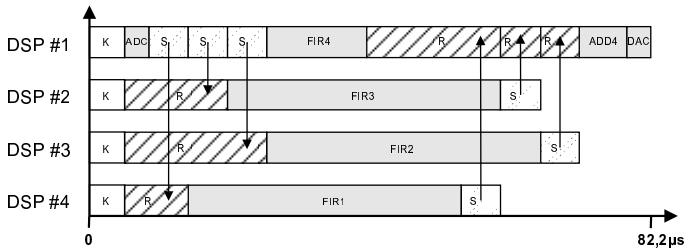

Figure 5. Gantt diagram of the optimized implementation of the initial design including the estimated computation and communication times.

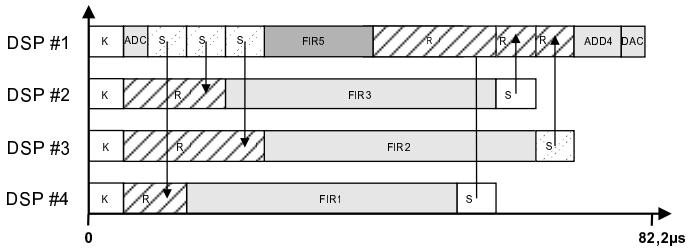

Figure 7. Gantt diagram depicting the schedule and predicted execution time of the redesign. Changes from the initial schedule are colored dark gray.

References

[1] S. S. Bhattacharyya, P. K. Murthy, and E. A. Lee. Synthesis of embedded software from synchronous dataflow specifications. *Journal of VLSI Signal Processing Systems*, 21(2), 1999.

[2] M. Eisenring, L. Thiele, and E. Zitzler. Conflicting criteria in embedded system design. *IEEE Design & Test of Computers*, 17(2):51–59, 2000.

[3] V. Fresse, M. Assouil, and O. Deforges. Rapid prototyping of image processing applications onto a multiprocessor architecture. In *International Conference on Acoustics, Speech, and Signal Processing ICASSP2000*, May 2000.

[4] R. Lauwereins, M. Engels, M. Ade, and J. Peperstraete. Grape-II:A System-Level Prototyping Environment for DSP Applications. *IEEE Computer*, 28(2):35–43, February 1995.

[5] V. K. Madisetti. Rapid Digital System Prototyping: Current Practice, Future Challenges. *IEEE Design & Test of Computers*, 13(3):12–22, 1996.

[6] C. Mathis, B. Rinner, M. Schmid, R. Schneider, and R. Weiss. A New Approach to Model Communication for Mapping and Scheduling DSP-Applications. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing ICASSP 2000*, pages 3354–3357, Istanbul, Turkey, June 2000. IEEE.

[7] B. Rinner, B. Ruprechter, and M. Schmid. Rapid Prototyping of Multi-DSP Systems Based on Accurate Performance Estimation. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing ICASSP 2001*, Salt Lake City, U.S.A., May 2001. IEEE.

[8] B. Rinner, M. Schmid, and R. Weiss. Automated Design and Implementation of Parallel Signal Processing Applications based on Performance Estimation. Technical Report TR 01/05, Institute for Technical Informatics, Graz University of Technology, 2001.

[9] E. Verhulst. Virtuoso: A virtual single processor programming system for distributed real-time applications. *Microprocessing and Microprogramming*, 40:103–115, 1994.

Proceedings of the Design,Automation and Test in Europe Conference and Exhibition (DATE'03)
1530-1591/03 $17.00 © 2003 **IEEE**