A Unified Codesign Run-time Environment for the UltraSONIC Reconfigurable Computer

Theerayod Wiangtong[1], Peter Y. K. Cheung[1], and Wayne Luk[2]

[1] Department of Electrical & Electronic Engineering,
Imperial College, London, UK
{tw1, p.cheung}@imperial.ac.uk
[2] Department of Computing,Imperial College, London, UK
wl@imperial.ac.uk

Abstract. This paper presents a codesign environment for the UltraSONIC reconfigurable computing platform which is designed specifically for real-time video applications. A codesign environment with automatic partitioning and scheduling between a host microprocessor and a number of reconfigurable processors is described. A unified runtime environment for both hardware and software tasks under the control of a task manager is proposed. The practicality of our system is demonstrated with an FFT application.

1 Introduction

Reconfigurable hardware has received increasing attention from the research community in the last decade. FPGA-based designs become popular because of their reconfigurable capability and short design-time which the old design style, like ASICs, cannot offer. Instead of using FPGAs simply as ASICs replacements, combining reconfigurable hardware with conventional microprocessors in a codesign system provides an even more flexible and powerful approach for implementing computation intensive applications, and this type of codesign system is our attention in this paper.

The major concerns in the design process for such codesign system are the synchronization and the integration of the hardware and the software design [1]. Examples are the partitioning between hardware and software, the scheduling of tasks and the communication between hardware and software tasks in order to achieve the shortest overall runtime. Decision on partitioning plays a major role in the overall system performance and cost. Scheduling is important to complete all the tasks in a real-time environment. These decisions are based heavily on design experience and are difficult to automate. Many design tools leave this burden to the designer by providing semi-auto interactive design environments. Fully automatic design approach for codesign system is generally considered to be impossible at present. In addition, traditional implementation employs a hardware model that is very different from that used in software. These distinctive views of hardware and software tasks can cause problem in the

design process. For example, swapping tasks between hardware and software can result in a totally new structure in the control circuit.

This paper reports a new method of constructing and handling tasks in a codesign system. We structure both hardware and software tasks in an interchangeable way without sacrificing the benefit of concurrency found in conventional hardware implementations. At the same time, the hardware task model exploits the advantages of modularity, scalability, cohesion and structured approach offered by software tasks. We further present a codesign flow containing the partitioning and scheduling algorithms to automate the decision process of where and when tasks are implemented and run. Our codesign system using this unified hardware/software task model is applied to a reconfigurable computer system known as UltraSONIC [14]. Finally we demonstrate the practicality of our system by implementing an FFT computational engine. The novel contributions of this paper are: 1) a unified way of structuring and modelling hardware and software tasks in a codesign system; 2) proposing a codesign flow for a system with mixed programmable hardware and microprocessor resource; 3) propose a run-time task manager design to exploit the unified model of tasks; 4) the demonstration of implementing a real application by using our model in the UltraSONIC reconfigurable platform.

The rest of this paper is organized as follows. Section 2 presents some of work related to codesign development systems. In section 3, we explain how the hardware and software tasks are modelled in our codesign system. Section 4 describes the codesign flow and environments of the UltraSONIC system used as the realistic target. A case study of implementing the FFT algorithm on the UltraSONIC system is discussed in section 5. Section 6 concludes this paper.

2 Related Work

There are many approaches to hardware/software codesign. Most focus on some particular stages in the design process such as system specification and modelling, partitioning and scheduling, compilation, system co-verification, cosimulation, and code generator for hardware and software interfacing. For example, Ptolemy [2] concentrates on hardware-software co-simulation and system modelling. Chinook [5] focuses on the synthesis of hardware and software interface. MUSIC [6], a multi-language design tools between SDL and MATLAB, is applied to mechatronic system. For embedded systems, CASTLE [7] and COSYMA [8] design environments are developed.

Codesign system specifically targeted for reconfigurable systems are PAM-Blox [4] and DEFACTO [3]. PAM-Blox is a design environment focusing on hardware synthesis to support PCI Pamette board that consists of five XC4000 Xilinx FPGAs. This design framework does not provide a complete codesign process. DEFACTO is an end-to-end design environment for developing applications mapped to configurable platform consists of FPGAs and a general-purpose microprocessor. DEFACTO concentrates on raising the level of abstraction to a higher level, and develop the parallelizing compiler technique to achieve opti-

mizations on loop transformations and memory accesses. Although both PAM-Blox and DEFACTO are developed specifically for reconfigurable platforms, they take no account of the existence of tasks in microprocessor. Consequentially, neither system is suitable for hardware/software codesign.

In this work, tasks can interchangeably be implemented in software or reconfigurable hardware resources. We suggest a novel way for task modelling and task management which will be described in details in the next section. We also present a novel idea of building infrastructure for dataflow-based applications implementing on this type of codesign system consisting of a single software resource (in the form of a microprocessor) and multiple reconfigurable hardware. Our approach is inherently modular and is suitable for implementing runtime reconfigurable designs.

3 System Architecture

Fig. 1 shows the hardware/software system model adopted in this work. We assume the use of a single processor (software) resource SW capable of multitasking, and a number of concurrent hardware processing elements PE0 to PEn, which are implemented on FPGAs. We employ a loosely coupled model with each processing element (PE) having its own single local memory. All system constraints such as shared resource conflicts, reconfiguration times (of the FPGAs) and communication times are all taken into account.

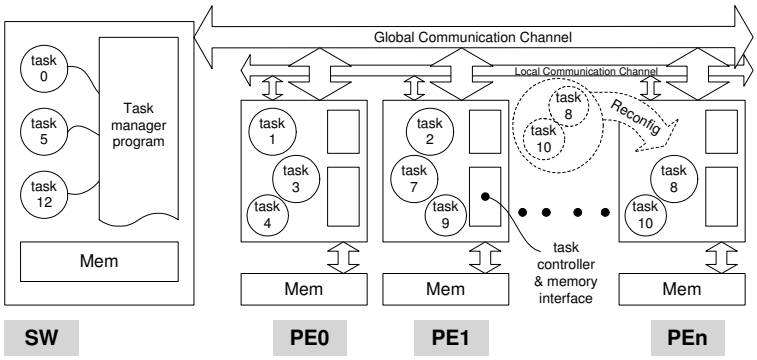


Fig. 1. Codesign System Architecture

The assumptions used in our model are: 1) tasks implemented in each hardware PE are coarse grain tasks which may consist of one or more functional tasks (blocks, loops). 2) Each PE has one local memory, only one task can access the local memory at any given time. Therefore multiple tasks residing in a given PE must execute sequentially; however, tasks residing across different PEs can execute concurrently. 3) Tasks for a PE may be dynamically swapped in and out

through dynamic reconfiguration. 4) A global communication channel is available for the processor and the PEs to communicate with each other. 5) Local communication channels are available for neighboring PEs to communicate with each other in a pipeline ring.

Because of the reconfigurable capability of the hardware, we can build the hardware tasks very much like software tasks. In this way, the management of task scheduling, task swapping and task allocation can be done in a unified manner, no matter whether the task in question is implemented in hardware or in software. Concurrency is not affected as long as we map concurrent tasks onto separate PEs. Although conceptually different PEs are separate from each other, multiple PEs may be implemented on a single FPGA device.

3.1 Hardware Task Model

UltraSONIC is a reconfigurable computer system designed specifically for real-time video processing applications. In such an application domain, it is reasonable to assume that applications are dominated by dataflow behavior with few control flow constructs[12]. Algorithms can be broken down into tasks in coarse (or functional) granularity and are represented as a directed acyclic graph (DAG). Nodes in the graph represent tasks and edges represent data dependency between tasks. Each task is characterized by its execution time, resource occupied, and its communication cost with other tasks.

The tasks we implement on our system are assumed to conform the following restrictions:

- Tasks in the DAG are processed (once for each task) from top to bottom according to their precedence levels and priorities.
- Communication between tasks is always through local single-port memory.
- Task execution is done in three consecutive steps: read input data, process the data, and write the results. This is done repeatedly until input data stored in memory are all processed. Thus the communication time between memory and task while executing is considered to be a part of the task execution time [11].
- Exactly one task in a given PE is active at any one time. This is a direct consequence of the single port memory restriction. However, multiple tasks may run concurrently provided that they are mapped to different PEs. This is an improvement over the model proposed by others in [9].
- A task starts executing as soon as all the necessary incoming data from its sources have arrived. It starts writing outgoing data to destinations immediately after processing is completed [10].

4 The Design Environment

Fig. 2 depicts the codesign environment in our system. The system to be implemented is assumed to be described in some suitable high level language, which is

then mapped to a DAG. Tasks are represented by nodes and communications by edges. The nodes are then partitioned into hardware or software tasks, and are scheduled to execute in order to obtain the minimum makespan (total processing time). The partitioning and scheduling software used is adapted from tabu search and list scheduling algorithms reported earlier by the authors [13]. The algorithms are, however, modified to be compatible with this architectural model. A two-phase clustering algorithm is used as a pre-processing step to modify the granularity of the tasks in the DAG in order to improve the critical path delay, enable more task parallelism and provide results the achieve 13%-17% shorter makespan [15].

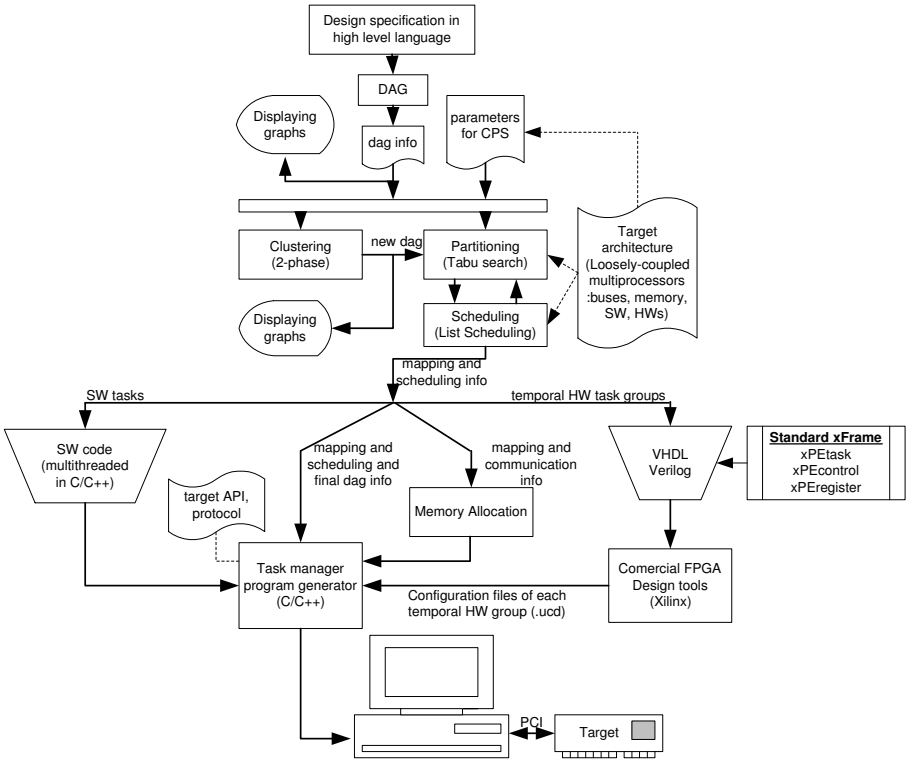


Fig. 2. Codesign environment

This group of algorithms, containing clustering, partitioning and scheduling, is collectively called the *CPS* algorithm for short. The *CPS* algorithm reads textual input files including DAG information and control information for clustering, partitioning and scheduling process. During this input stage, the user can optionally specify the type of tasks as *software-only* task, *hardware-only* task, or *dummy* task. A software-only task is a task that the user intentionally im-

plements in software without exception, and similarly for a hardware-only task. Dummy tasks are either source or sink for inputting and outputting data respectively, and are not involved in any computation. In our system, we assume that input and output data are initially provided and written to the microprocessor memory. Unspecified tasks are then free to be partitioned, scheduled and bound to either hardware or software resources.

The result of the partitioning and scheduling process are the physical and temporal binding for each task. Note that in case of hardware tasks, they may be divided into many temporal groups that can either be statically mapped to the hardware resource, or dynamically configured during runtime.

We currently assume that software tasks are manually written in C/C++, while hardware tasks are designed manually in a HDL (such as Verilog) using a library-based approach. Once all the hardware tasks for a given PE are available, they are wrapped in a standard frame with a pre-designed circuit (xPEtask, xPEregister and xPEcontrol) which is task independent. Commercially available synthesis and place-and-route tools are then used to produce the final configuration files for each hardware. Each task in this implementation method requires some hardware overhead to implement the task frame wrapper circuit. Therefore our system favours partitioning algorithms that generate coarse grain tasks.

The results from the partitioning and scheduling process, the memory allocator, the task control protocol, the API functions, the configuration files of hardware tasks, are used to automatically generate the codes for a *task manager program* that controls all operations in this system, such as dynamic configuration, task execution and data transfer. The resulting task manager is inherently multi-threaded to ensure that tasks are run concurrently.

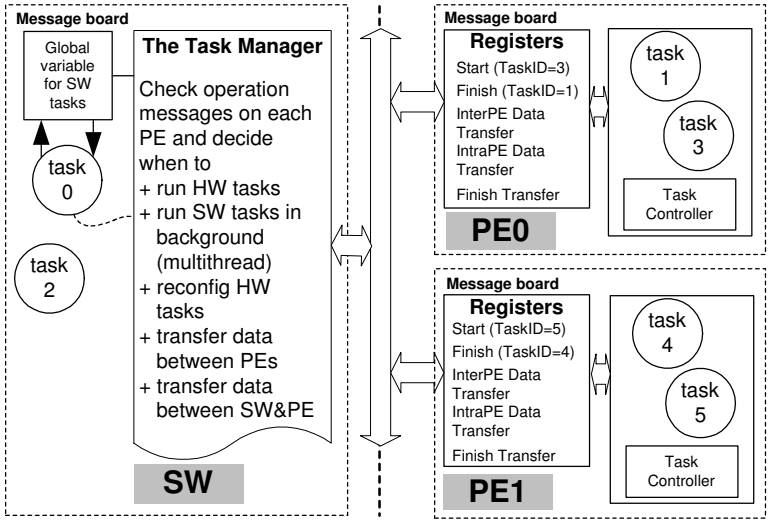


Fig. 3. The Task Manager control view

4.1 The Task Manager Program

The center of our implementation is the task manager (TM) program running on the microprocessor (software) resource to manage both hardware and software tasks. This program controls the sequencing of all the tasks, the transfer of data and the synchronization between them, and the dynamic reconfiguration of the FPGA in the PEs when required. Fig. 3 shows the conceptual control view of the TM and its operation. The TM communicates with a local task controller on each PE in order to assert control. A message board is used in each PE to receive commands from the TM or to flag finishing status to the TM.

In order to properly synchronize the execution of the tasks and the communication between tasks, our task manager employs a message-passing, event-triggered protocol when running a process. However, unlike a reactive codesign system [16], we do not use external real-time events as triggers. Instead, we use the termination of each task or data transfer as event-triggers, and signaling of such events is done through dedicated registers. For example, in Fig. 3, messages indicating execution completion from tasks 1 are posted to registers inside PE0. The task manager program polls these registers, finds the message, then proceeds to the next scheduled task, in this case task 3. By using this method, tasks on each PE is run independently because the program operates asynchronously at the system level.

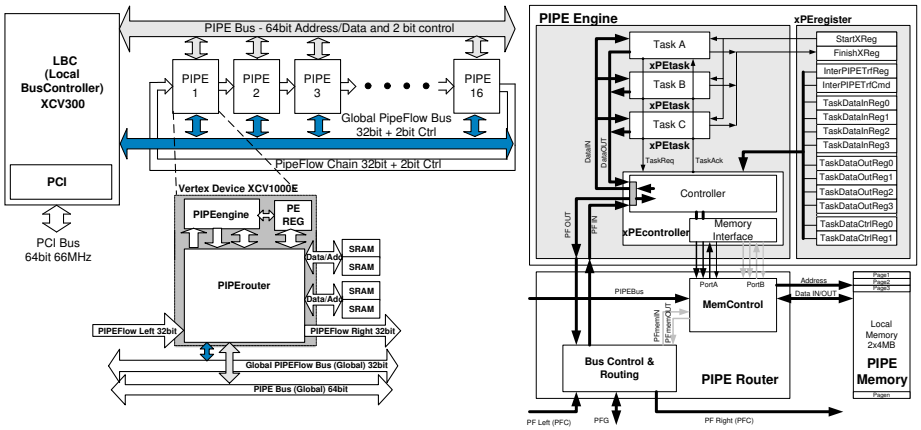
4.2 The UltraSONIC Reconfigurable Platform

The codesign described above is targetted for the UltraSONIC System [14]. UltraSONIC (see Fig. 4(a)) is a reconfigurable computing system designed to cope with the computational power and the high data throughput demanded by real-time video applications. The system consists of Plug-In Processing Elements (PIPEs) interconnected by local and global buses. The architecture exploits the spatial and the temporal parallelism in video processing algorithms. It also facilitates design reuse and supports the software plug-in methodology.

Fig. 4(b) shows how our codesign model is implemented in the UltraSONIC PIPE. The xPEcontrol implements the message passing protocol to control the operation of all the hardware tasks (xPEtask) resident in this PIPE. The message board is implemented in xPEregister. The total hardware overhead of using the Task Manager is modest. It consumes around 10% of the reconfigurable resource on each PIPE (which is implemented on Xilinx's XCV1000E).

5 An Example: FFT Implementation

In order to demonstrate the working of our system, we chose to implement the well known FFT algorithm. Although we can implement the algorithm for an arbitrary data length, we use an 8-point FFT implementation to illustrate the results of our codesign system. The DAG of an 8-point FFT can be straightforwardly extracted as shown in Fig. 5(a). Nodes 0, 1, 2 are used for arranging



(a) The UltraSONIC reconfigurable platform (b) The hardware design structure in each PIPE

Fig. 4. The hardware implementation in UltraSONIC architecture

inputs data and are implemented as software-only tasks. Tasks 3 to 14 are butterfly computation nodes. The number shown inside the parenthesis (the second number) on each edge is the number of data values needed for each iteration of the task. Each task is executed repeatedly until all data (shown as the first number on the edge) are processed. Initially in this DAG, the software execution times are obtained by profiling tasks on PC, while the hardware times and areas are obtained by using Xilinx development tools.

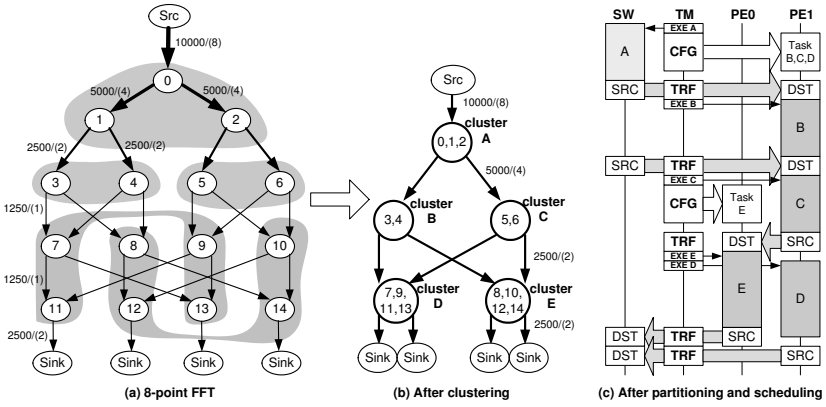


Fig. 5. The DAG of 8-point FFT algorithm

This DAG information is supplied to the CPS algorithm in our design environment (see Fig. 2) to perform automatic clustering, partitioning and schedul-

ing. Parameters such as reconfiguration time, bus speed, FPGA size, are all based on the UltraSONIC system. The clustering algorithm produces a new DAG that contains tasks in higher granularity as shown in Fig. 5(b). These new tasks are then iteratively partitioned and scheduled. The computational part of each of the hardware tasks are designed manually in Verilog and the software tasks are written in C. Our tools then combine the results from the partitioning and scheduling algorithms, wrap the hardware task designs automatically with the standard task frame, and generate the task manager program.

Fig. 5(c) depicts the run-time profile of this implementation. Each column represents activities on the available resources which are software tasks (SW), and two hardware processing elements (PE0 and PE1). TM is the task manager program which is also running on the software resource. The execution of this algorithm proceeds from top to bottom. It shows all the runtime activities including configuration (CFG), transferring data (TRF), events that trigger task executions (EXE), the source of data (SRC), receiving data (DST) and the executions of the tasks (A to E).

The FFT algorithm for different data window sizes are also tested on the UltraSONIC system and are shown to work correctly. The method, although requires some manual design steps, is very quick. Implementing the FFT algorithm only took a few hours from specification to completion.

6 Conclusions

This paper presents a semi-automatic codesign environment for a system consisting of single software and multiple reconfigurable hardware. It proposes the use of a task manager to combine the runtime support for hardware and software in order to improve modularity and scalability of the design. Partitioning and scheduling are done automatically. Codes for software tasks are run in software concurrently (using multi-threaded programming) with the task manager program which is based on message-passing and event-triggered protocol. Implementation of the FFT algorithm on UltraSONIC demonstrates the practicality of our approach.

Future work includes testing our codesign system with more complex applications, tools to map behavioral or structural descriptions to DAG automatically and to improve the task management environment so that external asynchronous real-time events can also be handled.

7 Acknowledgement

The authors would like to acknowledge the continuing support of John Stone, Simon Haynes, Henry Epsom and the rest of the UltraSONIC team at Sony Broadcast Professional Research Laboratory in UK.

References

1. Ernst, R., "Codesign of embedded systems: status and trends", *IEEE Design & Test of Computers*, 1998.
2. Manikutty, G.; Hanson, H., "Hardware/Software Partitioning of Synchronous Dataflow Graphs in the ACS domain of Ptolemy", University of Texas, Literature Survey, Final Report May 12 1999.
3. Hall, M.; Diniz, P.; Bondalapati, K.; Ziegler, H.; et al., "DEFACTO:A Design Environment for Adaptive Computing Technology", *Proceedings of the 6th Reconfigurable Architectures Workshop*, 1999.
4. Mencer, O.; Morf, M.; Flynn, M.J., "PAM-Blox: high performance FPGA design for adaptive computing", *FPGAs for Custom Computing Machines*, 1998.
5. Chou, P.H.; Ortega, R.B.; Borriello, G., "The Chinook hardware/software co-synthesis system", *System Synthesis*, 1995.
6. Coste, P.; Hessel, F.; Le Marrec, P.; Sugar, Z.; et al., "Multilanguage design of heterogeneous systems", *Hardware/Software Codesign*, 1999.
7. Wilberg, J.; Kuth, A.; Camposano, R.; Rosenstiel, W.; et al., "Design Exploration in CASTLE", *Workshop on High Level Synthesis Algorithms Tools and Design (HILES)*, 1995.
8. Ernst, R., "Hardware/Software Co-Design of Embedded Systems", *Asia Pacific Conference on Computer Hardware Description Languages*, 1997.
9. Srinivasan, V.; Govindarajan, S.; Vemuri, R., "Fine-grained and coarse-grained behavioral partitioning with effective utilization of memory and design space exploration for multi-FPGA architectures", *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 9, pp. 140 -158, 2001.
10. Hou, J.; Wolf, W., "Process partitioning for distributed embedded systems", *Hardware/Software Co-Design*, 1996.
11. Pop, T.; Eles, P.; Peng, Z., "Holistic scheduling and analysis of mixed time/event-triggered distributed embedded systems", *Hardware/Softwarw Codesign*, 2002.
12. Chatha, K.S.; Vemuri, R., "Hardware-software partitioning and pipelined scheduling of transformative applications", *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 10, pp. 193-208, 2002.
13. Wiangtong, T.; Cheung, P.Y.K.; Luk, W., "Comparing Three Heuristic Search Methods for Functional Partitioning in HW-SW Codesign", *International Journal on Design Automation for Embedded Systems*, vol. 6, pp. 425-449, July 2002.
14. Haynes, S.D.; others, a., "UltraSONIC: A Reconfigurable Architecture for Video Image Processing", *Field-Programmable Logic and Applications (FPL)*, 2002.
15. Wiangtong, T.; Cheung, P.Y.K.; Luk, W., "Cluster-Driven Hardware/Software Partitioning and Scheduling Approach For a Reconfigurable Computer System", submit to *Field-Programmable Logic and Applications (FPL)*, 2003.
16. De Micheli, G., "Computer-aided hardware-software codesign", *IEEE Micro*, Vol 14, pp. 10-16, 1994.