Preemptive HW/SW-Threading by combining ESL methodology and coarse grained reconfiguration

Marko Rößler, Ulrich Heinkel
Professur Schaltkreis- und Systementwurf
Chemnitz University of Technology
09126 Chemnitz, Germany
email: marko.roessler, ulrich.heinkel@etit.tu-chemnitz.de

I. Extended Abstract

Modern systems fulfill calculation tasks across the hardware-software boundary. Tasks are divided into coarse parallel subtasks that run on distributed resources. These resources are classified into a software (SW) and a hardware (HW) domain. The software domain usually contains processors for general purpose or digital signal calculations. Dedicated co-processors such as encryption or video en-/decoding units belong to the hardware domain. Nowadays, a decision in which domain a certain subtask will be executed in a system is usually taken during system level design. This is done on the basis of certain assumptions about the system requirements that might not hold at runtime. The HW/SW partitioning is static and cannot adapt to dynamically changing system requirements at runtime. Our contribution to tackle this, is to combine a ESL based HW/SW codesign methodology with a coarse grained reconfigurable System on Chip architecture. We propose this as Preemptive HW/SW-Threading.

There have been multiple research activities in both of these fields. The concept of preemptive multitasking on reconfigurable architectures has been proven in [1] and [2]. Nevertheless, most of the work are case studies that miss to provide programming models and work flows. In [3], Anderson et al. summarize multiple years of work. They describe a programming interface for HW/SW threads based on reinterpreted calls to the Pthread programming interface in combination with handy compilation process. Our work can be seen as an extension to this.

Figure 1 outlines the proposed methodology. Starting point of the work flow is the refinement of an algorithmic C description of a system application into parallel executable subtasks (threads). At this stage, we inject fragments of management code. This code allows to start and stop threads, synchronize threads with mutexes, parameter value to pass and the load/store of a thread context. The management code will be included in either implementation of a subtask and provides a common functionality to the HW and SW representation of a thread. For the software domain a common GCC is used for compilation. ESL tools like CatapultC or CoDeveloper from ImpulseC are used to generate synthesisable HDL code for the hardware domain.

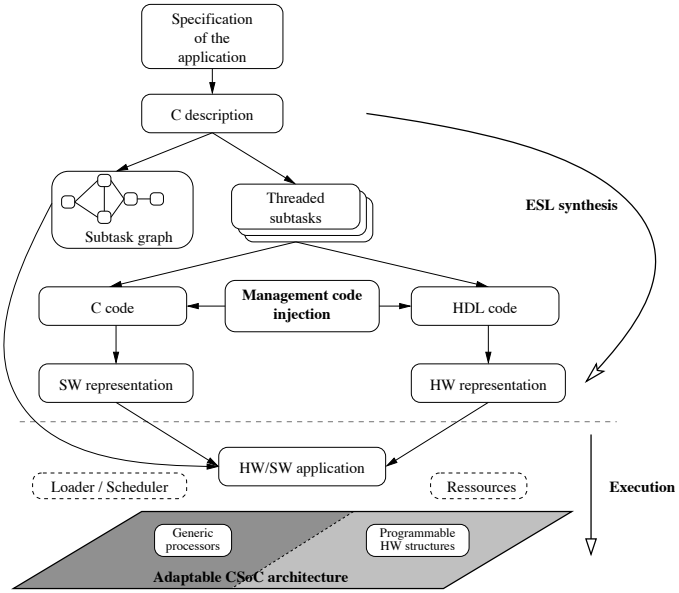

Fig. 1. Design flow for an adaptable CSoC using HW/SW-Threading

The runtime management is carried out by a central scheduler and a loader running in the software domain. The scheduler plans the execution and distribution of system applications and subtasks depending on available resources and external requirements. The loader is in charge of dynamic reconfiguration for the hardware domain.

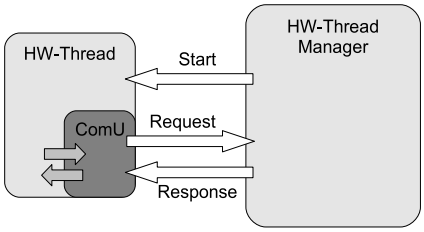

Fig. 2. Communication between HW-Thread and the central management

The concept and the overall toolflow was first introduced in [4]. A MJPEG coding chain was investigated as an usecase example of the flow in [5]. A library called **libcodev** has been developed to embed the SW representation of a subtask into the native thread environment of a minimalistic Linux operating system.

We finished the adaption of the Posix-Thread model to the High-Level C synthesis methodology of the CoDeveloper tool in order to provide a unified programming framework for HW/SW threads. This leads to an automated approach of transfering pthread library calls into the HW representation of a subtask during synthesis. Such calls are for instance pthread_create(), pthread_join() or pthread_exit(). In addition the handling of mutexes for the synchronisation of access to shared ressources has been integrated. Therefore, threads can lock and unlock a shared memory location by calling pthread_mutex_lock(), pthread_mutex_unlock() and the like.

In Figure 2, the principle of adding a small communication unit to the various HW threads is stated. This unit translates the calls into a protocol between threads and the central management unit. The central management holds the status of the overall system within two tables. The first table contains entries of all threads including their type, status, executing domain, priority and a list of executed subthreads. The second table stores the mutexes of the system with status and a queue of threads waiting for access. The central management implements various scheduling strategies for assigning mutexes to waiting threads and a simple procedure to decide in which domain a new thread will be executed.

The communication in the system takes place over a central bus structure. The system architecture is shown in Figure 3. All HW threads, the CPU which is executing SW threads and the management unit and the system SDRAM are connected to the bus structure.

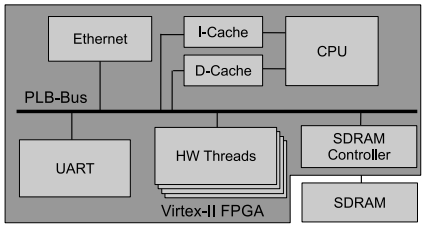

Fig. 3. Architecture of the CSoC for HW/SW threading

Current work is the integration of preemption. Interrupting a task or thread at any time without its prior notice we understand as hard preemption. In this case the scheduler is able to interrupt suddenly after an assigned time frame has expired. This has been shown to work for the hardware domain in [2]. Simmler et al. achieve this by stopping the system clock signal and getting hold of the internal state of all registers and flip flops by reading the SRAM information of the reconfigurable logic. This approach has some drawbacks. On one hand the hardware must not be preempted during RAM or E/A access. That would require additional synchronization between the scheduler and the HW thread. On the other hand a context switch between SW and HW domain is hardly possible because of the different style of their context. In the SW domain, this is a CPU register set, stack and instruction pointer whereas in HW domain this would be a bitstream.

The approach we are investigating will be a soft preemption that makes use of breakpoints. These will be inserted at the C level. A preemption unit automatically inserted aside of the communication unit. It will act cooperatively on suspend requests from the thread management and extract the context of the subtask in a generic way. A thread will later be resumed in either the HW or SW domain by restoring the context and jumping to the previous breakpoint.

For the moment the system is statically mapped onto a XILINX Virtex II Pro FPGA. Therefore the scheduler can only use HW thread ressources as they have been assigned to the system during the design phase. The next and final steps will integrate partial dynamic reconfiguration into the system in order to achieve more flexibility. This will be realized in a croase grained manner as entire IP blocks on the bus system become reconfigurable.

REFERENCES

[1] J.-Y. Mignolet, V. Nollet, P. Coene, D. Verkest, S. Vernalde, and R. Lauwereins, "Infrastructure for design and management of relocatable tasks in a heterogeneous reconfigurable system-on-chip," in *DATE '03: Proceedings of the conference on Design, Automation and Test in Europe*. Washington, DC, USA: IEEE Computer Society, 2003, p. 10986.

[2] L. Levinson, R. Maenner, M. Sesler, and H. Simmler, "Preemptive multitasking on fpgas," in *FCCM '00, Proceedings of the 2000 IEEE Symposium on Field Programmable Custom Computing Machines*. Washington, DC, USA: IEEE Computer Society, 2000, p. 49.

[3] E. Anderson, J. Agron, W. Peck, J. Stevens, F. Baijot, E. Komp, R. Sass, and D. Andrews, "Enabling a uniform programming model across the software/hardware boundary," in *FCCM '06: Proceedings of the 14th Annual IEEE Symposium on Field-Programmable Custom Computing Machines*. Washington, DC, USA: IEEE Computer Society, 2006, pp. 89–98.

[4] Marko Rössler and Ulrich Heinkel, "Concept for Dynamic Distribution of Tasks in Hardware/Software Systems," in *Dresdner Arbeitstagung Schaltungs- und Systementwurf DASS'06*, Dresden, Germany, Mai 2006, pp. 97–101.

[5] Enrico Billich and Marko Rössler and Ulrich Heinkel, "Discovering the optimal HW/SW-Partitioning of a MJPEG codec using the High-Level ESL-Tool CoDeveloper," in *Dresdner Arbeitstagung Schaltungs- und Systementwurf DASS'08*, Dresden, Germany, Mai 2008.