Infrastructure for Design and Management of Relocatable Tasks in a Heterogeneous Reconfigurable System-on-Chip

J-Y. Mignolet, V. Nollet, P. Coene, D.Verkest[‡]*, S. Vernalde, R. Lauwereins[‡]
IMEC vzw, Kapeldreef 75, 3001 Leuven, BELGIUM
{mignolet, nollet, coene}@imec.be

Abstract

The ability to (re)schedule a task either in hardware or software will be an important asset in a reconfigurable systems-on-chip. To support this feature we have developed an infrastructure that, combined with a suitable design environment permits the implementation and management of hardware/software relocatable tasks. This paper presents the general scope of our research, and details the communication scheme, the design environment and the hardware/software context switching issues. The infrastructure proved its feasibility by allowing us to design a relocatable video decoder. When implemented on an embedded platform, the decoder performs at 23 frames/s (320x240 pixels, 16 bits per pixel) in reconfigurable hardware and 6 frames/s in software.

1. Introduction

Today, emerging run-time reconfigurable hardware solutions are offering new perspectives on the use of hardware accelerators. Indeed, a piece of reconfigurable hardware can now be used to run different tasks in a sequential way. By using an adequate operating system, software-like tasks can be created, deleted and pre-empted in hardware as it is done in software.

A platform composed of a set of these reconfigurable hardware blocks and of instruction-set processors (ISP) can be used to combine two important assets: flexibility (of software) and performance (of hardware). An operating system can manage the different tasks of an application and spawn them in hardware or in software, depending on their computational requirements and on the quality of service that the user expects from these applications.

Design methodology for applications that can be relocated from hardware to software and vice-versa is a challenging research topic related to these platforms. The application should be developed in a way that ensures an equivalent behavior for its hardware and software implementations to allow run-time relocation. Furthermore, equivalence of states between hardware and software should be studied to efficiently enable heterogeneous context switches.

In the scope of our research on a general-purpose programmable platform based on reconfigurable hardware [1], we have developed an infrastructure for the design and management of relocatable tasks. The combination of a uniform communication scheme and OCAPI-xl [8, 9], a C++ library for unified hardware/software system design, allowed us to develop a relocatable video decoder. This was demonstrated on a platform composed of a commercial FPGA and a general purpose ISP. It is the first time to our knowledge that full hardware/software multitasking is addressed, in such a way that the operating system is able to spawn and relocate a task either in hardware or software.

The remainder of this paper is organized as follows. Section 2 puts the problem into perspective by positioning it in our general research activity. Section 3 describes the communication scheme we developed on the platform and its impact on the task management. Section 4 presents the object oriented design environment we used to design the application. Section 5 discusses the heterogeneous context switching issues. Section 6 gives an overview of implementation results on a specific case study. Finally some conclusions are drawn in Section 7. Related work [3, 5, 6, 7, 10, 12] will be discussed throughout the paper.

2. Hardware/software multitasking on a reconfigurable computing platform

The problem of designing and managing relocatable tasks fits into the more general research topic of hard-

* also Professor at Vrije Universiteit Brussel
‡ also Professor at Katholieke Universiteit Leuven

Part of this research has been funded by the European Commission through the IST-AMDREL project (Contract No IST-2001-34379).

ware/software multitasking on a reconfigurable computing platform for networked portable multimedia appliances. The aim is to increase the computation power of current multimedia portable devices (such as personal digital assistants or mobile phones) while keeping their flexibility. Performance should be coupled with low power consumption, since portable devices are battery-operated. Flexibility is required because different applications will run on the device, with different architecture requirements. Moreover, it enables upgrading and downloading of new applications. Reconfigurable hardware meets these two requirements and is therefore a valid solution to this problem.

Our research activity addresses different parts of the problem, as shown in Figure1. A complete description is presented in [1].

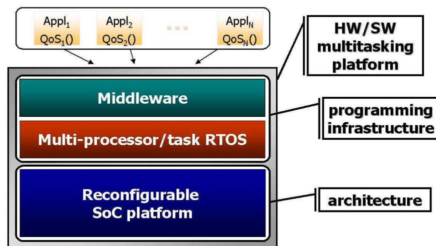

Figure 1. Our research activity

The bottom part represents the platform activity, which consists in defining suitable architectures for reconfigurable computing platforms. The selection of the correct granularity for the reconfigurable hardware blocks and the development of the interconnection network that will handle the communication between the different parts of the system are two of the challenges for this activity.

The interconnection network plays an important role in our infrastructure, since it supports the communication of the system. Networks-on-chip provide a solution for handling communication in complex systems-on-chip (SoC). We are studying packet-switched interconnection networks for reconfigurable platforms [2]. To assist this research, we develop "soft" interconnection networks on commercial reconfigurable hardware. They are qualified soft because they are implemented using the reconfigurable fabric, while future platforms will use fixed networks implemented using standard ASIC technology. This soft interconnection network divides the reconfigurable hardware in tiles of equal size. Every tile can run one task at a given moment.

The middle part of Figure 1 represents the operating system for reconfigurable systems (OS4RS) we have developed to manage the tasks over the different resources. In order to handle hardware tasks, we have developed extensions as a complement to the traditional operating system.

The OS4RS provides multiple functions. First of all, it implements a hardware abstraction layer (HAL), which provides a clean interface to the reconfigurable logic. Secondly, the OS4RS is responsible for scheduling tasks, both on the ISP and on the reconfigurable logic. This implies that the OS4RS abstracts the total computational pool, containing the ISP and the reconfigurable tiles, in such a way that the application designer should not be aware on which computing resource the application will run. A critical part of the functionality is the uniform communication framework, which allows tasks to send/receive messages, regardless of their execution location.

The upper part of Figure 1 represents the middleware layer. This layer takes the application as input and decides on the partitioning of the tasks. This decision is driven by quality-of-service considerations.

The application should be designed in such a way that it can be executed on the platform. In a first approach, we use a uniform HW/SW design environment to design the application. Although it ensures a common behavior for both HW and SW version of the task, it still requires both versions of the task to be present in memory. In future work, we will look at unified code that can be interpreted by the middleware layer and spawned either in HW or SW. This approach will not only be platform independent similar to JAVA, it will also reduce the memory footprint, since the software and the hardware code will be integrated.

3. Uniform communication scheme

Relocating a task from hardware to software should not affect the way other tasks are communicating with the relocated task. By providing a uniform communication scheme for hardware and software tasks, the OS4RS we developed hides this complexity.

In our approach, inter-task communication is based on message passing. Messages are transferred from one task to another in a common format for both hardware and software tasks. Both the operating system and the hardware architecture should therefore support this kind of communication.

Every task is assigned a logical address. Whenever the OS4RS schedules a task in hardware, an address translation table is updated. This address translation table allows the operating system to translate a logical address into a physical address and vice versa. The assigned physical address is based on the location of the task in the interconnection network (ICN).

The OS4RS provides a message passing API, which uses these logical/physical addresses to route the messages. In our communication scheme, three subtypes of

message passing between tasks can be distinguished (Figure 2).

Messages between two tasks, both scheduled on the ISP (P_1 and P_2), are routed solely based on their logical address and do not pass the HAL.

Communication between an ISP task and a FPGA task (P_3 and P_C) does pass through the hardware abstraction layer. In this case, a translation between the logical address and the physical address is performed by the communication API. The task's physical address allows the HAL to determine on which tile of the ICN the sending or receiving task is executing.

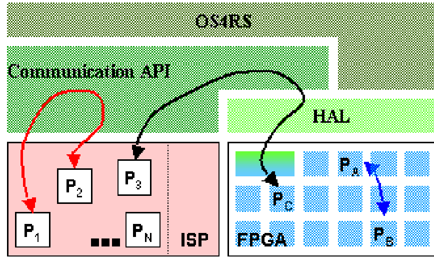

Figure 2: Message passing between tasks.

On the hardware side, the packet-switched interconnection network is providing the necessary support for message passing. Messages between tasks, both scheduled in hardware, are routed inside the interconnection network without passing through the HAL. Nevertheless, since the operating system controls the task placement, it also controls the way the messages are routed inside the ICN, by adjusting the hardware task routing tables.

The packet-switched interconnection network, which supports the hardware communication in our infrastructure, solves some operating system issues related to hardware management such as task placement, location independence, routing, and inter-task communication. Diessel and Wigley previously listed these issues in [3].

Task placement is the problem of positioning a task somewhere in the reconfigurable hardware fabric. At design time, task placement is realized by using place and route tools from the reconfigurable hardware vendor. This usually generates an irregular task footprint. At run-time, the management software is responsible for arranging all the tasks inside the reconfigurable fabric. When using irregular task shapes, the management software needs to run a complex fitting algorithm (e.g. [6, 7]). Executing this placement algorithm considerably increases run-time overhead. In our infrastructure, the designer constrains the place and route tool to fit the task in the shape of a tile. Run-time task placement is therefore greatly facilitated, since every tile has the same size and same shape. The OS4RS is aware of the tile usage at any moment. As a consequence, it can spawn a new task without placement overhead by replacing the tile content through partial reconfiguration of the FPGA.

Location independence consists of being able to place any task in any free location. This is an FPGA-dependent problem, which requires a relocatable bitstream for every task. Currently, our approach is to have a partial bitstream for every tile. A better alternative is to manipulate a single bitstream at run-time (Jbits [4] could be used in the case of Xilinx devices).

The run-time routing problem can be described as providing connectivity between the newly placed task and the rest of the system. In our case, a communication infrastructure is implemented at design-time inside the interconnection network. This infrastructure provides the new task with a fixed communication interface, based on routing tables. Once again, the OS4RS should not run any complex algorithm. Its only action is updating the routing tables every time a new task is inserted/removed from the reconfigurable hardware.

The issue of inter-task communication is handled by the OS4RS, as described earlier this section.

Our architecture makes a trade-off between area and run-time overhead. As every tile is identical in size and shape, the area fragmentation (as defined by Wigley and Kearney in [5]) is indeed higher than in a system where the logic blocks can have different sizes and shapes. However, the OS4RS will only need a very small execution time to spawn a task on the reconfigurable hardware, since the allocation algorithm is limited to the check of tile availability.

4. Unified design of hardware and software with OCAPI-xl

A challenging step in the design of relocatable tasks is to provide a common behavior for the HW and the SW implementation of a task. One possibility to achieve this is to use a unified representation that can be refined to both hardware and software.

OCAPI-xl [8, 9] provides this ability. OCAPI-xl is a C++ library that allows unified hardware/software system design. Through the use of the set of objects from OCAPI-xl, a designer can represent the application as communicating threads. The objects contain timing information, allowing cycle-true simulation of the system. Once the system is designed, automatic code generation for both hardware and software is available. This ensures a uniform behavior for both implementations in our heterogeneous reconfigurable system.

Through the use of the FLI (Foreign Language Interface) feature of OCAPI-xl, an interface can be designed that represents the communication with the other tasks. This interface provides functions like *send_message* and *receive_message* that will afterwards be expanded to the corresponding hardware or software implementation code.

This ensures a communication scheme that is common to both implementations.

5. Heterogeneous context switch issues

It is possible for the programmer to know at design time on which of the heterogeneous processors the tasks preferably should run (as described by Lilja in [11]). However, our architecture does not guarantee run-time availability of hardware tiles. Furthermore, the switch latency of hardware tasks (in the range of 20ms on a FPGA) severely limits the number of time-based context switches. We therefore prefer spatial multitasking in hardware, in contrast to the time-based multitasking presented in [10, 12]. Since the number of tiles is limited, the OS4RS is forced to decide at run-time on the allocation of resources, in order to achieve maximum performance. Consequently, it should be possible for the OS4RS to pre-empt and relocate tasks from the reconfigurable logic to the ISP and vice versa.

The ISP registers and the task memory completely describe the state of any task running on the ISP. Consequently, the state of a preempted task can be fully saved by pushing all the ISP registers on the task stack. Whenever the task gets rescheduled at the ISP, simply popping the register values from its stack and initializing the registers with these values restores its state.

This approach is not usable for a hardware task, since it depicts its state in a completely different way: state information is held in several registers, latches and internal memory, in a way that is very specific for a given task implementation. There is no simple, universal state representation, as for tasks executing on the ISP. Nevertheless, the operating system will need a way to extract and restore the state of a task executing in hardware, since this is a key issue when enabling heterogeneous context switches.

A way to extract and restore state when dealing with tasks executing on the reconfigurable logic, is described in [10, 12]. State extraction is achieved by getting all status information bits out of the read back bitstream. This way, manipulation of the configuration bitstream allows re-initializing the hardware task. Adopting this methodology to enable heterogeneous context switches would require a translation layer in the operating system, allowing it to translate an ISP type state into FPGA state bits and vice versa. Furthermore, with this technique, the exact position of all the configuration bits in the bitstream must be known. It is clear that this kind of approach does not produce a universally applicable solution for storing/restoring task state.

We propose to use a high level abstraction of the task state information. This way the OS4RS is able to dynamically reschedule a task from the ISP to the reconfigurable logic and vice versa. This technique a based on an idea presented in [12]. Figure 3a represents a relocatable task, containing several states. This task contains 2 switch-point states, at which the operating system can relocate the task. The entire switch process is described in detail by Figure 4. In order to relocate a task, the operating system can signal that task at any time (1). Whenever the signaled task reaches a switch-point, it goes into the interrupted state (2) (Figure 3b). In this interrupted state all the relevant state information of the switch-point is transferred to the OS4RS (3). Consequently, the OS4RS will re-initiate the task on the second heterogeneous processor using the received state information (4). The task resumes on the second processor, by continuing to execute in the corresponding switch-point (5). Note that the task described in Figure 3 contains multiple switch-points, which makes it possible that the state information that needs to be transferred to the OS4RS can be different for each switch-point. Furthermore, the unified design of both the ISP and FPGA version of a task, as described in section 4, ensures that the position of the switch-points and the state information are identical.

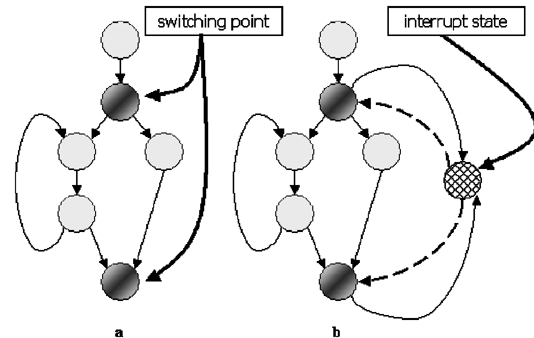

Figure 3: Relocatable task

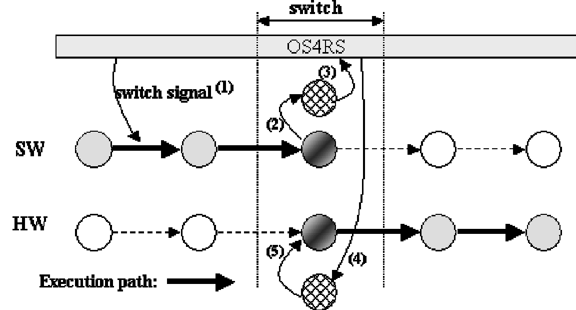

Figure 4: Task switching: from software to hardware.

The relocatable video decoder, described in section 6, illustrates that the developed operating system is able to dynamically reschedule a task from the ISP to the reconfigurable logic and vice versa. At this point in time, this simplified application contains only one switchable state, which contains no state information.

The insertion of these "low overhead" switch-points will also be strongly architecture dependent: in case of a shared memory between the ISP and the reconfigurable

logic, transferring state can be as simple as passing a pointer, while in case of distributed memory, data will have to be copied.

On a long term, the design tool should be able to create these switch-points automatically. One of the inputs of the design tool will be the target architecture. The OS4RS will then use these switch-points to perform the context switches in a way hidden from the designer.

6. Relocatable video decoder

As an illustration of our infrastructure a relocatable video decoder is presented. First the platform on which the decoder was implemented is described. Then the decoder implementation is detailed. Finally performance and implementation results are presented.

6.1 The T-ReCS Gecko demonstrator

Based on the concepts presented in Section 2, we have developed a first reconfigurable computing platform for HW/SW multitasking. The Gecko demonstrator (Figure 5) is a platform composed of a Compaq iPAQ 3760 and a Xilinx Virtex 2 FPGA. The iPAQ is a personal digital assistant (PDA) that features a StrongARM SA-1110 ISP and an expansion bus that allows connection of an external device. The FPGA is a XC2V6000 containing 6000k system gates.

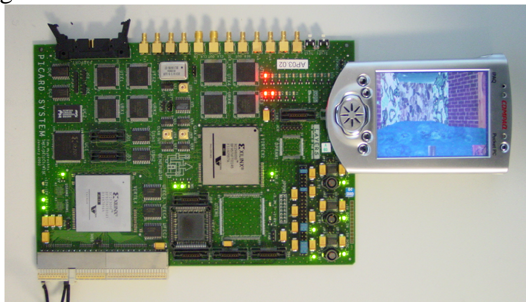

Figure 5. The T-ReCS Gecko demonstrator

The FPGA is mounted on a generic prototyping board connected to the iPAQ via the expansion bus. On the FPGA, we developed a soft packet-switched interconnection network composed of two application tiles and one interface tile.

6.2 The video decoder

Our Gecko platform is showcasing a video decoder that can be executed in hardware or in software and that can be rescheduled at run-time.

The video decoder is a motion JPEG frame decoder. A send thread passes the coded frames one by one to the decoder thread. This thread decodes the frames and sends them, one macroblock at a time, to a receive thread that reconstructs the images and displays them. The send thread and the receive thread run in software on the iPAQ, while the decoder thread can be scheduled in HW or in SW (Figure 6).

The switch point has been inserted at the end of the frame because, at this point, no state information has to be transferred from HW to SW or vice-versa.

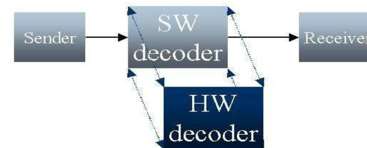

Figure 6. Relocatable decoder

6.3 Results

Two implementations of the JPEG decoder have been designed. The first one is quality factor and run-length encoding specific (referred as *specific* hereafter), meaning that the quantization tables and the Huffman tables are fixed, while the second one can accept any of these tables (referred as *general* hereafter). Both implementations target the 4:2:0 sampling ratio. The results of the implementation of the decoders in hardware are 9570 LUTs for the specific implementation and 15901 LUTs for the general one. (These results are given by the report file from the Synplicity® Synplify Pro™ advanced FPGA synthesis tool, targeting the Virtex2 XC2V6000 device, speed grade -4, and for a required clock frequency of 40 MHz).

The frame rate of the decoder is 6 frames per second (fps) for the software implementation and 23 fps for the hardware. These results are the same for both general and specific implementation. The clock runs at 40 MHz, which is the maximum frequency that can be used for this application on the FPGA. When achieving 6 fps in software, the CPU load is about 95%. Moving the task to hardware reduces the computational load of the CPU, but increases the load generated by the communication. Indeed, the communication between the send thread and the decoder on the one side, and between the decoder and the receive thread on the other side, is heavily loading the processor.

The communication between the iPAQ and the FPGA is performed using BlockRAM internal DPRAMs of the Xilinx Virtex FPGA. While the DPRAM can be accessed at about 20 MHz, the CPU memory access clock runs at 103 MHz. Since the CPU is using a synchronous RAM scheme to access these DPRAMs, wait-states have to be inserted. During these wait-states, the CPU is prevented from doing anything else, which increases the CPU load. Therefore, the hardware performance is mainly limited by the speed of the CPU-FPGA interface. This results in the fact that for a performance of 23 fps in hardware, the CPU is also at

95% load.

Although the OS4RS overhead for relocating the decoder from software to hardware is only about 100 µs, the total latency is about 108 ms. The low OS4RS overhead can be explained by the absence of a complex task placement algorithm. Most of the relocation latency is caused by the actual partial reconfiguration through the slow CPU-FPGA interface. In theory, the total software to hardware relocation latency can be reduced to about 11ms, when performing the partial reconfiguration at full speed. When relocating a task from hardware to software, the total relocation latency is equal to the OS4RS overhead, since in this case no partial reconfiguration is required.

Regarding power dissipation, the demo setup cannot show relevant results. Indeed, the present platform uses an FPGA as reconfigurable hardware. Traditionally, FPGAs are used for prototyping and are not meant to be power efficient. The final platform we are targeting will be composed of new, low-power fine- and coarse-grain reconfigurable hardware that will improve the total power dissipation of the platform. Power efficiency will be provided by the ability of spawning highly parallel, computation intensive tasks on this kind of hardware.

7. Conclusions

This paper describes a novel infrastructure for the design and management of relocatable tasks in a reconfigurable SoC. The infrastructure consists of a unified HW/SW communication scheme and a common HW/SW behavior. The uniform communication is ensured by a common message-passing scheme inside the operating system and a packet switched interconnection network. The common behavior is guaranteed by use of a design environment for unified HW/SW system design. The design methodology has been applied to a video decoder implemented on an embedded platform composed of an instruction-set processor and a network-on-FPGA. The video decoder is relocatable and can perform 6 fps in software and 23 fps in hardware. Future work includes automated switch-point placement and implementation in order to have a low context switch overhead when heterogeneously rescheduling tasks.

Acknowledgements

We would like to thank Kamesh Rao of Xilinx for carefully reviewing and commenting this paper.

References

[1] J-Y. Mignolet, S. Vernalde, D. Verkest, R. Lauwereins, “Enabling hardware-software multitasking on a reconfigurable computing platform for networked portable multimedia appliances”, Proceedings of the International Conference on Engineering Reconfigurable Systems and Architecture 2002, pages 116-122, Las Vegas, June 2002.

[2] T. Marescaux, A. Bartic, D. Verkest, S. Vernalde and R. Lauwereins, “Interconnection Networks Enable Fine-Grain Dynamic Multi-Tasking on FPGAs”, FPL’2002, pages 795-805, Montpellier France.

[3] O. Diessel, G. Wigley, “Opportunities for Operating Systems Research in Reconfigurable Computing”, Technical report ACRC-99-018, Advanced Computing Research Centre, School of Computer and Information Science, University of South Australia, August, 1999

[4] S. Guccione, D. Levi, P. Sundararajan, “ JBits: A Java-based Interface for Reconfigurable Computing”, 2nd Annual Military and Aerospace Applications of Programmable Devices and Technologies Conference (MAPLD).

[5] G. Wigley, D. Kearney, “The Management of Applications for Reconfigurable Computing using an Operating System”, In Proc. Seventh Asia-Pacific Computer Systems Architecture Conference, January 2002, ACS Press.

[6] J. Burns, A. Donlin, J. Hogg, S. Singh, M. de Wit, “A Dynamic Reconfiguration Run-Time System”, Proceedings of the 5th IEEE Symposium on FPGA-Based Custom Computing Machines (FCCM '97), Napa Valley, CA, April 1997.

[7] H. Walder, M. Platzner, “Non-preemptive Multitasking on FPGAs: Task Placement and Footprint Transform”, Proceedings of the International Conference on Engineering Reconfigurable Systems and Architecture 2002, pages 24-30, Las Vegas, June 2002

[8] www.imec.be/ocapi

[9] G. Vanmeerbeeck, P. Schaumont, S. Vernalde, M. Engels, I. Bolsens, “Hardware/Software Partitioning of embedded system in OCAPI-xl”, CODES’01, Copenhagen, Denmark, April 2001.

[10] H. Simmler, L. Levinson, R. Männer, “Multitasking on FPGA Coprocessors”, Proc. 10th Int’l Conf. Field Programmable Logic and Applications, pages 121-130, Villach, Austria, August 2000.

[11] D. Lilja, “Partitioning Tasks Between a Pair of Interconnected Heterogeneous Processors: A Case Study”, Concurrency: Practice and Experience, Vol. 7, No. 3, May 1995, pp. 209-223

[12] L. Levinson, R. Männer, M.Sesler, H. Simmler, “Preemptive Multitasking on FPGAs”, Proceedings of the 2000 IEEE Symposium on Field Programmable Custom Computing Machines.

[13] F.Vermeulen, L. Nachtergaele, F Catthoor, D. Verkest, H. De Man,”Flexible Hardware Acceleration for Multimedia Oriented Microprocessors”, (accepted) IEEE Transactions on Very Large Scale Integration Systems.