


A Platform for Mixed HW/SW Algorithm Specifications for the Exploration of SW and

View metadata, citation and similar papers at core.ac.uk

brought to you by  CORE

provided by Infoscience - École polytechnique fédérale de Lausanne

Christophe Lucarz and Marco Mattavelli

Ecole Polytechnique Fédérale de Lausanne, Switzerland
{christophe.lucarz,marco.mattavelli}@epfl.ch

Abstract. The increasing complexity in particular of video and multimedia processing has led to the need of developing the algorithms specification using software implementations that become in practice generic reference implementations. Mapping directly such software models in platforms made of processors and dedicated HW elements becomes harder and harder for the complexity of the models and for the large choice of possible partitioning options. This paper describes a new platform aiming at supporting the mapping of software specifications into mixed SW and HW implementations. The platform is supported by profiling capabilities specifically conceived to study data transfers between SW and HW modules. Such optimization capabilities can be used to achieve different objectives such as optimization of memory architectures or low power designs by the minimization of data transfers.

1 Introduction

Due to the ever increasing complexity of integrated processing systems, software verification models are necessary to test performance and specify accurately a system behaviour. A reference software, in both cases of the processing specified by a standard body or for any other "custom" algorithm, is the starting point of every real implementation. This is typical case for MPEG video coding where the "reference software" is now the real specification. There is an intrinsic difference between real implementations that can be made of HW and SW components working in parallel using specific mapping of data on different logical or physical memories and the "reference software" that is usually based on a sequential model with a monolithic memory architecture, such difference is generically called "architectural gap". In the process of transforming the reference SW into a real implementation the possibility of exploring different architectural solutions for specific modules and study the resulting data exchanges for defining optimal memory architectures is a very attractive approach. This system exploration option is particularly important for complex systems where conformance and validation of the new module designs need to be performed at each stage of the design otherwise incurring into the risk of

developing solutions not respecting their original specification or not providing adequate performances.

This paper presents an integrated platform and the associated profiling capabilities that supports mixed SW and HW specifications and enables the hardware designer to seamlessly transform a Reference Software into software plus hardware modules mapped on an FPGA.

The paper is organized as follows: section 2 presents a brief state of the art on integrated HW/SW platforms. Section 3 provides a general view of the platform introducing the innovative elements. Section 4 describes the details of the platform that enables HW/SW support. Section 5 presents the capabilities of the profiling tool and explain how it can be used to study and optimize data transfers satisfying different criteria.

2 State of the Art of Platforms Supporting Mixed HW/SW Implementations

Many platforms have been designed to support mixed SW/HW implementations, but all of them suffer from the fact that there is no easy procedure capable to seamlessly plug hardware modules described in HDL to a pure software algorithm. Either the memory management is a burdensome task or the call of the hardware module is done by an embedded processor on the platform.

Environments which support HW/SW implementations are generally based on a platform containing an embedded processor and some dedicated hardware logic like FPGA as described in the work of Andreas Koch [2]. The control program lies in the embedded processor. However, data on the host are available easily thanks to virtual serial ports. But the plugging of hardware modules inside the reference software running on the host remains the most difficult task.

The work of Martyn Edwards and Benjamin Fozard [3] is interesting in the way a FPGA-based algorithm can be activated from the host processor. This platform is based on the Celoxica RC1000-PP board and communicates with the host by using the PCI bus. The control program is on the host processor, sends control information to the FPGA and transfers data in small shared memory which is part of the hardware platform. In this case, the designer/programmer must do explicitly the data transfer between the host and the local memory. Many other works about coprocessors have been reported in literature. Some examples are given in [4] [5]. However the problem of seamless plug-in of HDL modules is still there, the data transfers in the charge of the designer with can be a very burdensome task when dealing with complex data-dominated video or multimedia algorithms.

In some works on coprocessors, data transfers can be generated automatically by the host like for instance in [6]. But data are copied in the local memory at a fixed place. Thus, the HDL module must be aware of the physical addresses

of the data in the local memory. The management of the addresses can be a burdensome task when dealing with complex algorithms.

The Virtual Socket concept and associated platform has been presented in [10] [9] [7] and has been developed to support the mixed specification of MPEG-4 Part2 and Part 10 (AVC/H.264) in terms of reference SW including the plug-in of HDL modules. The platform is constituted by a standard PC where the SW is executed and by a PCMCIA card that contains a FPGA and a local memory. The data transfers between the host memory and the local memory on the FPGA must be explicitly specified by the designer/programmer.

Specifying explicitly the data transfers would not constitute a serious burden when dealing with simple deterministic algorithms for which the data required by the HDL module are known exactly. Unfortunately for very complex design cases where design trade-offs are much more convenient (and often are the only viable solutions) than worst case designs data transfers cannot be explicitly specified in advance by the designer.

Our work is based on the Virtual Socket platform to which we add the virtual memory capability to allow automatic data transfers from the host to the local memory. The goal of our platform implementation is to provide a "direct map" of any SW portion to a corresponding HDL specification without the need of specifying any data transfer explicitly. In other words, to extend the concept of Virtual Socket for plugging HDL modules to SW partition with the concept of virtual memory. HDL modules and software algorithm share an unified virtual memory space. Having a shared memory - enforced by a cache-coherence protocol - between the CPU running the SW sections and the platform supporting HW avoids the need of specifying explicitly all the data transfers. The clear advantage is that the data transfer needs of the developed HDL module can be directly profiled so as to explore different memory architecture solutions. Another advantage of such direct map is that conformance with the original SW specification is guaranteed at any stage and the generation of test vectors is naturally provided by the way the HDL module is plugged to the SW section.

3 Description of the Virtual Socket Platform

The Virtual Socket platform is composed of a PC and a PCMCIA card that includes a FPGA and a local memory. The Virtual Socket handles the communications between the host (the PC environment) and the HDL modules (in the FPGA inside the PCMCIA).

Given that the HDL modules are implemented on the FPGA, in principle they would only have access to the local memory (see figure 1). This was the case of the first implementation of the Virtual Socket platform, with the consequence that all the data transfers from the host to the local memory had to be specifically specified in advance by the designer/programmer himself. Such operation beside being error prone or be implemented transferring more data than necessary it is not straightforward and may become difficult to be handled when the volume of data is comparable with the size of the (small) local

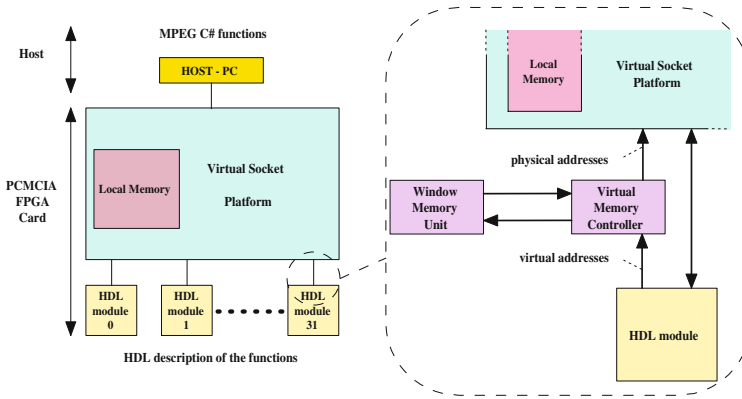


Fig. 1. The Virtual Socket platform overview

memory. Therefore, an extension has been conceived and implemented so as to handle these data transfers automatically. The Virtual Memory Extension (VME) is implemented by two components: the hardware extension to the Virtual Socket platform (Window Manager Unit) and a Virtual Manager Window (VMW) library on the host PC. The cache-coherence protocol is implemented in the Window Manager unit (WMU) using a TLB (Translation Lookaside Buffer) and is handled by the software support (VMW). The HDL module is designed simply generating virtual addresses relative to the user virtual memory space (on the host) to request data and execute the processing tasks.

The processing of the data on the platform using the virtual memory feature proceed as follows. The algorithm starts the execution on the PC and associated host memory. The Virtual Socket environment allows the HDL module to have a seamless direct access to the host memory thanks to the Virtual Memory Extension and allows the HDL module to be started easily from the software algorithm thanks to the VMW Library. Figure 2 shows what are the relations between the host memory, the reference software algorithm, the hardware function call and the HDL module.

Given an algorithm described in a mixed HW/SW form (1): some parts are executed in software with the host processor (5), some other parts are executed by hardware HDL modules (4) on the Virtual Socket platform hardware. To deal with mixed HW/SW algorithms, it is very convenient if the HDL and C functions have access to the same user memory space (6) which is part of the host hardware and where are stored the data to process. The main memory space is trivially available for the parts of the algorithm executed in software, which is much less evident for the parts executed in hardware.

The section of C code the programmer intends to execute in hardware is replaced by the hardware call function (2). This latter is based on the Virtual Manager Window Library. The programmer sets the parameters to give to the HDL module. The `Start_module()` function drives the Virtual Socket platform

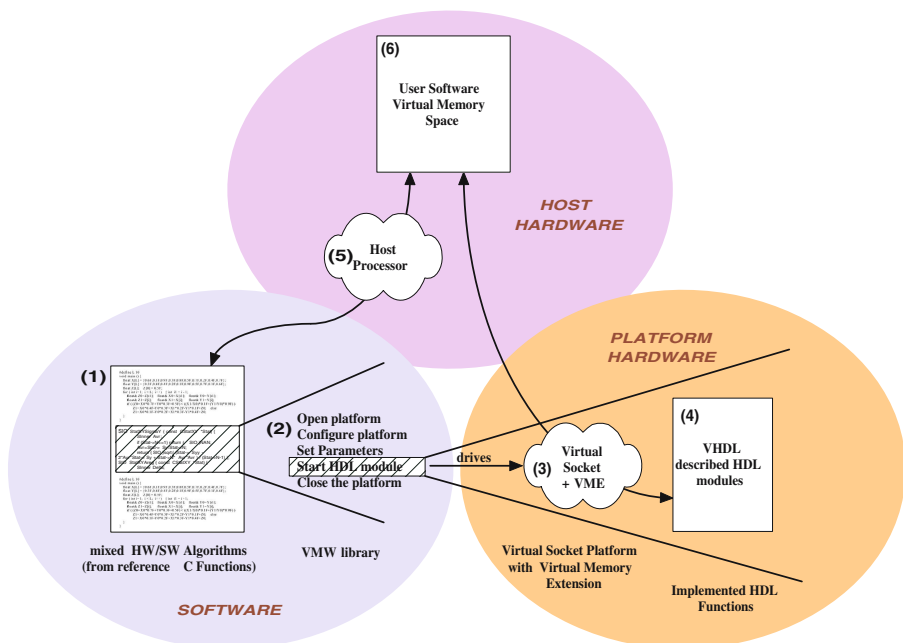


Fig. 2. Interactions between the C function, the HDL module and the shared memory space

and the VME (3) to activate the HDL module (4). The VMW library manages all the data transfers between the main memory (6) and the local memory of the platform (3) because as the HDL module is in a FPGA, it has access only this local memory. Thanks to the VME, the HDL module has access to the host memory without intervention of the programmer. Data are sent to the HDL module and results are updated in the main memory automatically thanks to the software library support. When the HDL module finishes its work, the hardware call function is terminated by closing the platform and the reference software algorithm can be continued on the host PC.

4 Details on HW Implementation and SW Support

The following section describes in more details how the Virtual Socket platform supporting the Virtual Memory Extension is implemented. The first part explains how virtual memory accesses are possible from the HDL modules. Then, the Virtual Memory Window library, i.e. the software support is described in details to show how virtual memory accesses are handled. The final part explains how HDL modules can be integrated in the platform using a well-defined protocol.

4.1 The Heart of Simplicity: HDL Modules Virtual Memory Accesses

The HDL modules are implemented on the FPGA, so that they have access only to the local memory of the Virtual Socket platform. With the implementation of the Virtual Memory Extension, the HDL modules have a direct access to the software virtual memory space located on the host PC.

The right part of figure 1 shows in details how the connections between a HDL module, the Virtual Socket platform and the Virtual Memory Extension are implemented (in the hardware of the PCMCIA card). The virtual addresses generated by the HDL modules are handled by the Virtual Memory Controller (VMC) and the Window Memory Unit (WMU). The WMU is a component taken from the work of Vuletić and al.[8]. The WMU translates virtual addresses into physical addresses. The VMC is in charge of intercepting precise signals at right time from the interface between the HDL module and the platform in order to send information to the WMU which executes the translation. Among the signals intercepted by the VMC, can be mentioned the address signal, the count signal (number of data requested by the HDL module) and the strobe signal. The virtual addresses refer to the unified virtual memory space and the physical addresses refer to the local memory on the card. A physical address is composed of an offset and a page number. The local memory (on the current PCMCIA card platform) is composed of 32 pages of 2 kB. The offset corresponds to the location of the data in the page. The software support library (on the host PC) fills the pages of the local memory with the requested data coming from the virtual memory. When the WMU receives an unknown virtual address, it raises an interrupt through the interrupt controller of the card. The interrupt is taken in charge by the software support (on the host PC) and the requested data are written from the host memory to the local memory.

From the designer/programmer point of view using the Virtual Memory Extension, the whole process of data transfers is completely transparent. The only issue the designer/programmer has to care of is to generate the virtual addresses accordingly to the data contained in the host memory space. The whole task of transferring data to local memory is done by the platform and its software support.

4.2 The Software Support: The Virtual Memory Window Library

The Virtual Memory Window (VMW) library is built on the FPGA card driver (Wildcard II API), the Virtual Socket API developed by Yifeng Qiu and Wael Badawy bases on the works [9] [10] and the WIN32 API.

The Virtual Socket platform can be used with or without the Virtual Memory Extension. The designer/programmer is free to choose if the data transfers between the main memory on the host and the local memory on the card are done automatically (virtual mode) or manually (explicit mode).

The following piece of C code shows how a HDL module can be easily called from the Reference Software by using the Virtual Memory Extension:

```
int main(int argc, char *argv[]) {

/* [. . .] Reference Software Algorithm stops here */

/* Beginning of the HDL module calling procedure */

/***** CONFIGURING THE PLATFORM *****/
  Platform_Init();    // Virtual Socket
  VMW_Init() ;       // Virtual Memory Extension

/***** PARAMETERS SETTINGS *****/
  Module_Param.nb_param = 4 ;           // number of parameters
  Module_Param.Param[0] = A ;           // parameter 1
  Module_Param.Param[1] = B ;           // parameter 2
  Module_Param.Param[2] = C ;           // parameter 3
  Module_Param.Param[3] = D ;           // parameter 4

/***** HDL MODULE START *****/
  Start_module(1, &Module_Param) ;

/***** CLOSING THE PLATFORM *****/
  VMW_Stop();         // Virtual Memory Extension
  Platform_Stop();    // Virtual Socket

/* End of the HDL module calling procedure */

/* [. . .] the Reference Software Algorithm continues*/

}
```

First the designer/programmer must configure the platform by using the `Platform_Init()` and `VMW_Init()` functions from the Virtual Socket API and VMW API. HDL modules are activated thanks to the function `Start_module()` from the VMW API. The designer/programmer must set a given number of parameters needed for the configuration of the HDL module. This can be done thanks to the data structure `Module_Param`. Sixteen parameters are available for each HDL module.

4.3 The Integration of the HDL Modules in the Platform

The HDL module is linked to the Virtual Socket platform thanks to a well-defined interface and a precise communication protocol.

Figure 3 illustrates the essential elements of the communication protocol. A HDL module can issue two types of requests: read or write data (in main or local memory, it depends on the operating mode: virtual or explicit mode).

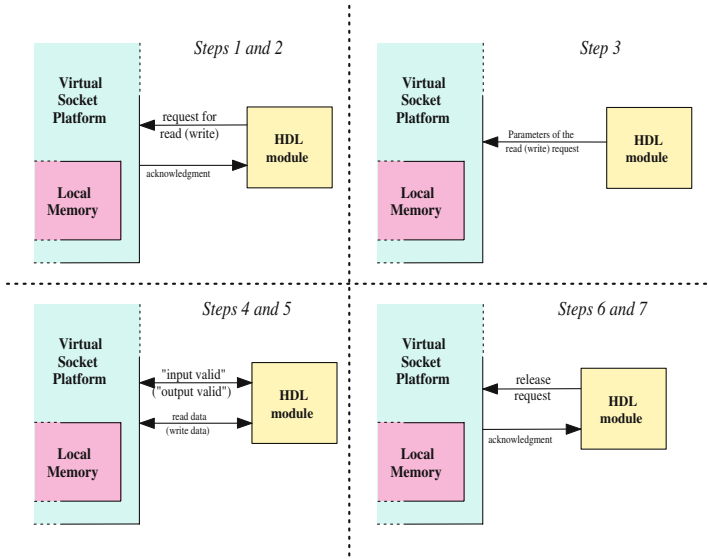


Fig. 3. The communication protocol between a HDL module and the Virtual Socket Platform

Read and write protocols are very similar. The following is the description of the communication protocol for the read (write) request:

1. The user HDL module asks to read (write) data, it issues a read request for reading (writing) the memory.
2. The platform accepts the reading (writing) request and if the data are available in the local memory, it generates an acknowledgement signal to the user HDL module. Otherwise the Virtual Memory Extension copies the requested data of the host memory into the local memory and then generates the acknowledgement.
3. Once the user HDL module receives the acknowledgement signal, it asks for reading (writing) some data directly from (to) the memory space. This request is performed by asserting a "memory read" ("output valid") signal together with setting up some other parameters signals (identification number of the HDL module used, the virtual address and how much data must be read (written)).
4. The platform accepts those signals and reads (writes) data from (to) the memory space. When the platform finishes each reading (writing), it asserts "input valid" ("output valid") and the data are ready to input of the user HDL module (platform).
5. The user HDL module receives (sends) the data from (to) the interface.
6. The user HDL module asserts a request to ask for releasing the reading (writing) operations when finished.
7. The platform generates an acknowledgement signal to release the reading (writing) operations.

In the Virtual mode, the read and write addresses contain the addresses of the data in the unified virtual memory space. It was like the HDL modules see the host memory.

5 Profiling Tools: Testing and Optimizing Data Transfers

Optimization of data transfers is a very important issue particularly for data dominated systems such as multimedia and video processing. Minimizing data transfer is also important for achieving low power implementations that are fundamental for mobile communication terminals. Data transfers contributes to the power dissipations and need to be optimized to achieve low power designs. The profiling tools supported by the platform allow the programmer to receive a feedback on the data requested by the HDL module.

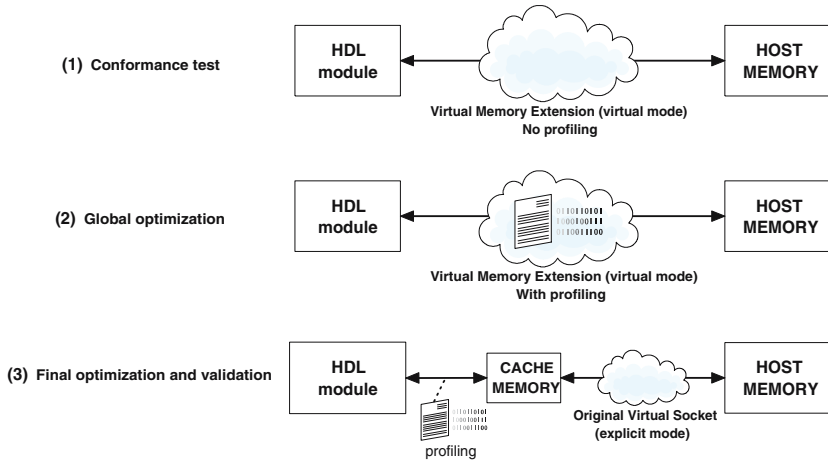


Fig. 4. The optimization methodology

Figure 4 shows the methodology to achieve an optimized hardware function (HDL module) relative to data transfers. The first step concerns the validation of the design. Using the Virtual Memory Extension, the equivalency of the C and HDL functions are verified. Virtual memory feature allows the designer/programmer to focus exclusively on the HDL module conformance checking. He can forget about the memory management during this phase. The second phase consists in understanding and having a global overview of the data transfers made between the platform and the HDL module. The way the data are accessed, the re-organization of data can be source of optimization. When the data required by the HDL module are profiled, the designer/programmer enters the last phase in which data transfers are optimized between HDL module and cache memory.

6 Conclusion

This paper describes the implementation of a platform capable of supporting the execution of algorithms described in mixed SW and HW form. The platform provide a seamless environment for migrating section of the SW into HDL modules that include a "virtual memory space" common to SW sections and to the HW modules. On one side conformance of the HDL modules with the reference SW is guaranteed at any stage of the design, on the other side the programmer/designer can focus on different aspects of the design. First design efforts can be focused on the module functionality without worrying about data transfers, then using the profiled data transfer on design of appropriate memory architectures or any other design optimization that matches the specific criteria of the design.

References

1. Annapolis Micro Systems, WILDCARD-II Reference Manual, 12968-000 Revision 2.6 (January 2004)
2. Koch, A.: A comprehensive prototyping-platform for hardware-software codesign. Rapid System Prototyping, 2000. In: RSP 2000 Proceedings. 11th International Workshop, 21-23 June, 2000, pp. 78–82 (2000)
3. Edwards, M., Fozard, B.: Rapid prototyping of mixed hardware and software systems. In: Digital System Design, 2002, Proceedings. Euromicro Symposium, 4-6 September, 2002, pp. 118–125 (2002)
4. Pradeep, R., Vinay, S., Burman, S., Kamakoti, V.: FPGA based agile algorithm-on-demand coprocessor, Design, Automation and Test in Europe 2005 (March 2005)
5. Plessl, C., Platzner, M.: TKDM - a reconfigurable co-processor in a PC's memory slot. In: Field-Programmable Technology (FPT) Proceedings. 2003 IEEE International Conference, 15-17 December, 2003, pp. 252–259. IEEE Computer Society Press, Los Alamitos (2003)
6. Sukhsawas, S., Benkrid, K., Crookes, D.: A reconfigurable high level FPGA-based coprocessor. In: Computer Architectures for Machine Perception, 2003 IEEE International Workshop, 12-16 May 2003, p. 4 (2003)
7. Schumacher, P., Mattavelli, M., Chirila-Rus, A., Turney, R.: A Virtual Socket Framework for Rapid Emulation of Video and Multimedia Designs. In: Multimedia and Expo, 2005 (ICME 2005) IEEE International Conference, 6-8 July, 2005, pp. 872–875 (2005)
8. Vuletic, M., Pozzi, L., Ienne, P.: Virtual memory window for application-specific reconfigurable coprocessors. In: Proceedings of the 41st Design Automation Conference, June 2004, San Diego, Calif (2004)
9. Amer, I., Rahman, C.A., Mohamed, T., Sayed, M., Badawy, W.: A hardware-accelerated framework with IP-blocks for application in MPEG-4. In: System-on-Chip for Real-Time Applications, Proceedings. Fifth International Workshop, 20-24 July, 2005, pp. 211–214 (2005)
10. Mohamed, T.S., Badawy, W.: Integrated hardware-software platform for image processing applications, In: System-on-Chip for Real-Time Applications, Proceedings. 4th IEEE International Workshop, IEEE Computer Society Press, Los Alamitos (2004)