A HW/SW codesign platform for Algorithm-Architecture Adaptation

Christophe Lucarz, Marco Mattavelli
Ecole Polytechnique Fédérale
de Lausanne (EPFL)
CH 1015 Lausanne, Switzerland

Julien Dubois
Université de Bourgogne,
Laboratoire LE2I
21000 Dijon, France

Abstract— **The increasing complexity of signal processing algorithms has lead to the need of developing the algorithms specifications using generic software implementations that become in practice the reference implementation. This fact can be particularly observed in the field of video and multimedia processing where reference software is the main normative reference. Adapting the algorithms specified by such software models into architectures composed by processors and dedicated HW elements becomes a very resource consuming task 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 adaptation of algorithms specified by generic non optimized software specifications into mixed SW and HW implementations. The platform is supported by profiling capabilities specifically developed to study data transfers between the SW and the HW modules. Such profiling and optimization capabilities can be used to achieve different objectives in the algorithm architecture adaptation process such as optimization of memory architectures or low power designs by the minimization of data transfers.**

I. INTRODUCTION

Algorithm-Architecture Adaptation is a very difficult task when dealing with very complex signal processing algorithms such as the one faced today in many application fields. It consists in finding the best architecture match for an algorithm that is usually written using a sequential program such as C or C++. The architecture can be composed of HW componets as well as other heterogeneous components such as DSP, processors and FPGAs. Mapping large software specifications onto a heterogeneous hardware platform is a complex and difficult process that cannot be achieved in one single shot. Usually the program is partitioned into smaller components in order to master the hardware design by dealing with components of manageable size. The problem arises when the hardware blocks need to be validated as single elements or when they are put together. The interfaces are sometime critical to be designed and appropriate test vectors need to be generated for conformance and performance testing. Furthermore, in the process of transforming the reference software into a real implementation (i.e. adapting the algorithm to architectures) the possibility of exploring different architectural solutions for specific modules and study the resulting data exchanges between components for defining optimal memory architectures is a very attractive approach.

The paper presents a platform that supports the profiling and testing of hardware modules as direct ”plug-ins” of the original reference software algorithm. The paper presents also the features of the profiling tool which enable the designer to measure the data transfers needed for the interface of the hardware component so that the designer can investigate different memory architectures optimizing data exchanges and the bandwidth between the different hardware modules.

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. Section 6 proposes an example of integration of the Motion Estimation module of MPEG-4 part 2.

II. STATE OF THE ART

Testing the implementation in HW of sections of a reference software is not a trivial task. It requires a platform which enables the designer to seameless ”call” the hardware component directly from the (reference) software. This is possible only if the hardware component is closely linked to the (reference) software environment. Some HW/SW co-design platforms can be used to support the algorithm-architecture adaptation methodology, but all of them suffer from the fact that there exist no simple 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 both hardware and software 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 must explicitly specify 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 existing, the specification of the data transfers that remains in charge of the designer might 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 is found in [6]. However, data are copied in the local memory at a pre-defined location. Thus, the HDL module must be aware of the physical addresses of the data in the local memory. Again the management of the addresses can be a non-trivial and resource consuming task when dealing with complex algorithms.

The Virtual Socket concept implemented in a support 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) specifications 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. Also for this platform 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.

The work described in this paper is based on the Virtual Socket platform extended by adding the virtual memory capability to allow automatic data transfers from the host, running the SW part, to the local HW memory. The goal of such 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 a 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 of such solution is that the data transfer needed to feed the 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.

III. 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).

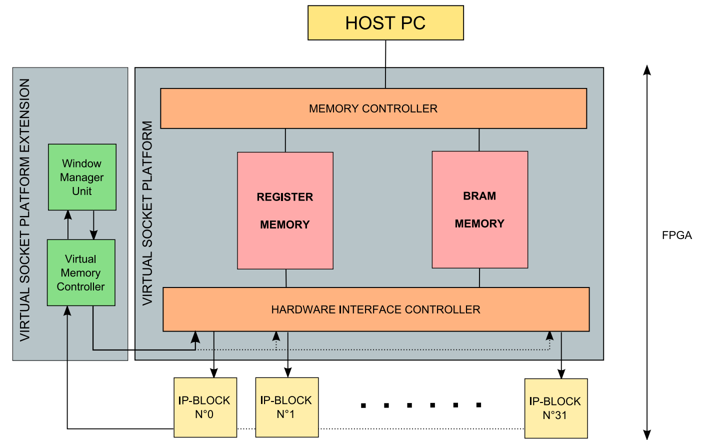

Fig. 1. The Virtual Socket platform overview

Given that the HDL modules are implemented on the FPGA, they have only a physical 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 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 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 interactions between the

unified virtual memory, the reference software algorithm, and the HDL module.

Given a reference software composed of several functions A, B, C, D and E. In order to test the HDL modules separately, the designer needs to execute some parts of the reference algorithm using the host processor and to test the hardware module. The Virtual Socket platform is the support for the hardware module for testing. 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, named unified virtual memory on figure 2. This memory is part of the host hardware and contains the data to process. This host memory space is trivially available by the processor which executes the reference software, but it is much less evident for the Virtual Socket platform which is on the FPGA and is the support for the HDL modules.

For instance the designer wants to run function C on the reference software and functions D and E together using one HDL module which merges the two functions. The section of the reference code the designer intends to execute in hardware is replaced by the following piece of code which is called the HDL module calling procedure:

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

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

/* Beginning of the HDL module calling procedure */

/******* OPEN / 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*/

}
```

The HDL module calling procedure is composed of the following steps:

1) the designer must configure the platform by using the ”Platform_Init( )” and ”VMW_Init( )” functions from the Virtual Socket API and VMW API
2) The designer 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.
3) the HDL call function is started. This function writes the parameters in the register memory of the Virtual Socket platform (see figure 1). ”Start_module()” drives the Virtual Socket platform and the VME to activate the HDL module. The function ”Start_module( )” is from the VMW API
4) when the entire job is finished, the platform is closed.

The VMW library manages all the data transfers between the main memory (unified virtual memory) and the local memory of the platform 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 designer. 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.

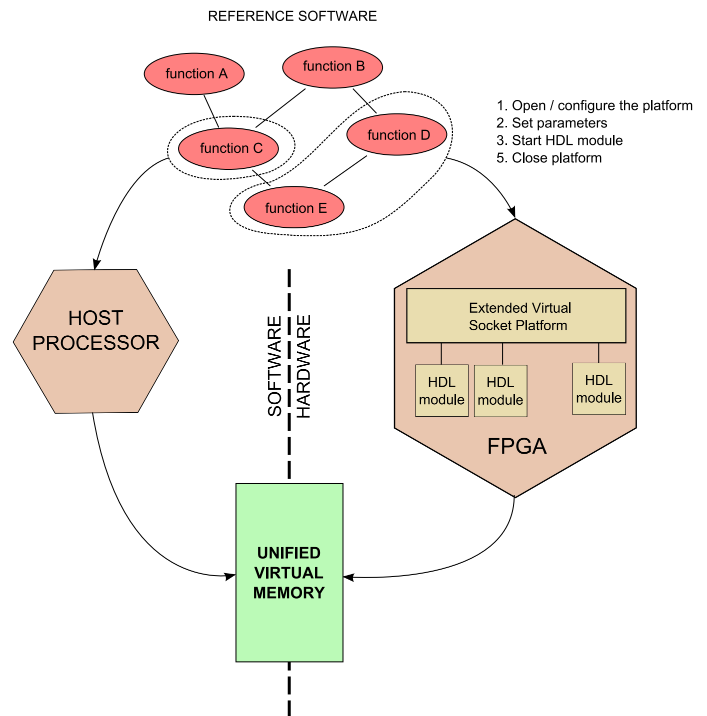

Fig. 2. interactions between the C function, the HDL module and the shared memory space

IV. 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.

A. *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 left part of figure 1 shows how the connections between a HDL module, the Virtual Socket platform and the Virtual Memory Extension are implemented. 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 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 point of view using the Virtual Memory Extension, the whole process of data transfers is completely transparent. The only issue the designer 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.

B. *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 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).

C. *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.

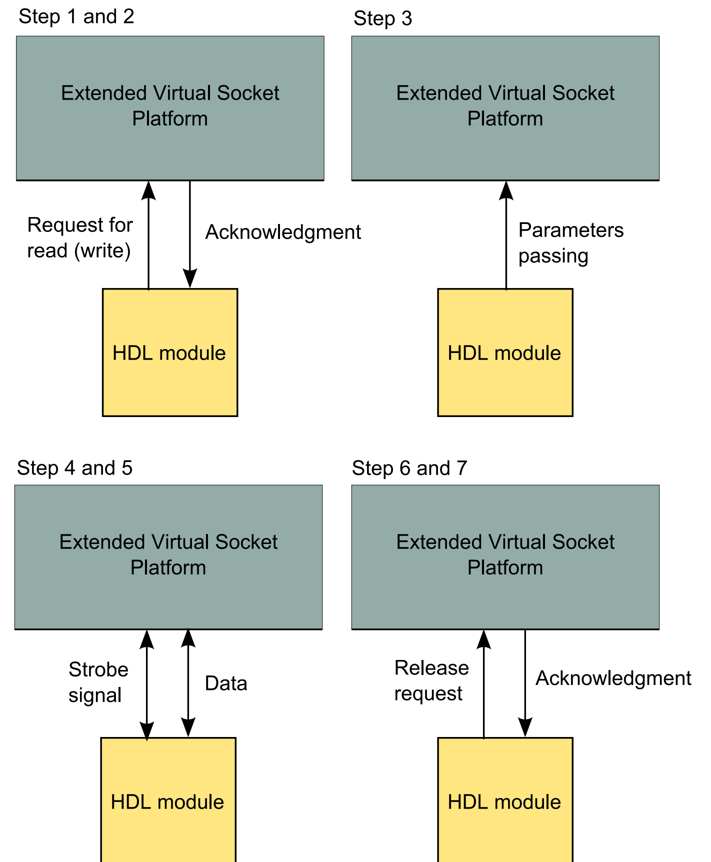

Fig. 3. the communication protocol between a HDL module and the Virtual Socket Platform

Figure 3 illustrates the protocol used by the HDL modules to communicate with the Virtual Socket platform. 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). There is a great similarity between the read and write protocols. Figure 3 is an illustration of the communication protocol. The following section describes the steps of the read protocol. The write protocol works exactly with the same steps.

1) The HDL module asks to read data, it issues a read request for reading the memory.
2) The platform accepts the read request and in the case the data are available in the local memory, the platform generates an acknowledgement signal to the user HDL module. In the other case, 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 some data directly

from the memory space. This request is performed by asserting a strobe 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).
4) The platform accepts those signals and reads data from the memory space. When the platform finishes each reading, it asserts a strobe signal and the data are ready to input of the user HDL module.
5) The user HDL module receives the data from the interface.
6) The user HDL module asserts a request to ask for releasing the reading operations when finished.
7) The platform generates an acknowledgement signal to release the reading 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.

V. Profiling tools: testing and optimizing data transfers

State of the art signal processing algorithms are essentially data dominated systems and the data flow between the modules must be carefully optimized so that to reach low power design, necessary for any embedded systems implementation. Data transfers provide a relevant contribution to the overall power dissipations and need to be optimized to achieve low power designs. The profiling tools supported by the platform allow the designer to receive feedback information on the data exchanges with the HDL module.

Figure 4 shows the methodology to develop an optimized hardware function (HDL module) versus its data exchanges. The first step is constituted by 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 to focus only on the HDL module conformance checking. The designer can forget everything about the memory management during this phase. The second phase consists in understanding and having a global overview of the data transfers exchanged between the platform and the HDL module. The way the data are accessed, the re-organization of data can be the object of accurate optimization. When the data exchanged by the HDL module are profiled, the designer enters the last phase in which data transfers are optimized between HDL module and cache memory.

VI. Example of SW-HW module integration: MPEG-4 motion estimation

A. Description of the HW module

The HW module performs the macroblock based motion estimation stage required by a frame based MPEG encoder [11]. A motion vector is obtained by selecting the best match between the reference macro-block and any position within a specified search window. The motion estimation algorithm is based on a reduced search strategy that reduces of up to two orders of magnitude the number of possible matchings within the search window, but requires flexibility in term of access latency to any position in the search window. In standard full-search implementations, an exhaustive search procedure is implemented calculating the matching function for all search window position. The approach is resource consuming, but the data access is perfectly regular. In a reduced search strategy configuration [11], [12], the block matching is only processed for a non-deterministic sub-set of block positions in the search window, therefore the access of any area in a search window is necessary. The set of positions is determined during runtime in function of the intermediate results and is calculated by a software processor (implemented by a MicroBlaze embedded in the FPGA in this example). On the next generation of Virtual Socket platforms based on a Wildcard IV board, the MicroBlaze could be replaced by a hardwired processor (PowerPC) embedded in the Virtex IV FPGA. In this example of module implementation, the search window width can be set up to 256 pixels, and the window height is fixed to 40 pixels. An external memory enables to store the full search window and the reference block. The internal memory permits to reduce the number of accesses to the external memory and any block in the internal memory can be accessible without any additional latency time. Any matching can be performed in less than 180 ns (with an overall clock frequency at 100 MHz) when accessing data in the internal memory. For instance, for a full-search configuration, the designed architecture processes at 14 frames per second at CIF video resolution format (i.e. 352x288 pixels frame size) with 41x25 pixels search range. As in other classical designs, the matching metric is based on the Mean Absolute Difference (MAD) evaluation. So as to obtain higher performances, the number of components that performs the MAD processing has been multiplied by a factor of 4 and a specific optimization of the data transfer architecture is needed to provide the required input-data bandwidth. The problem of optimizing such architecture consist on the complexity of the possible operating mode defined during runtime and on the requested module flexibility versus the variable image size and search window size.

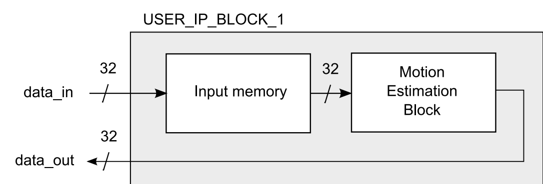

Fig. 5. an architecture of the MPEG Motion Estimation IP block

B. HW module design and integration with the reference SW

The interfacing of the HW motion estimation module results very simple thanks to the Virtual Memory feature and to the wrapper module (User_IP_Block). The designer just needs to specify the following parameters: two pointers respectively on the beginning of the two consecutive images, the image size (height and width), the search window and the block

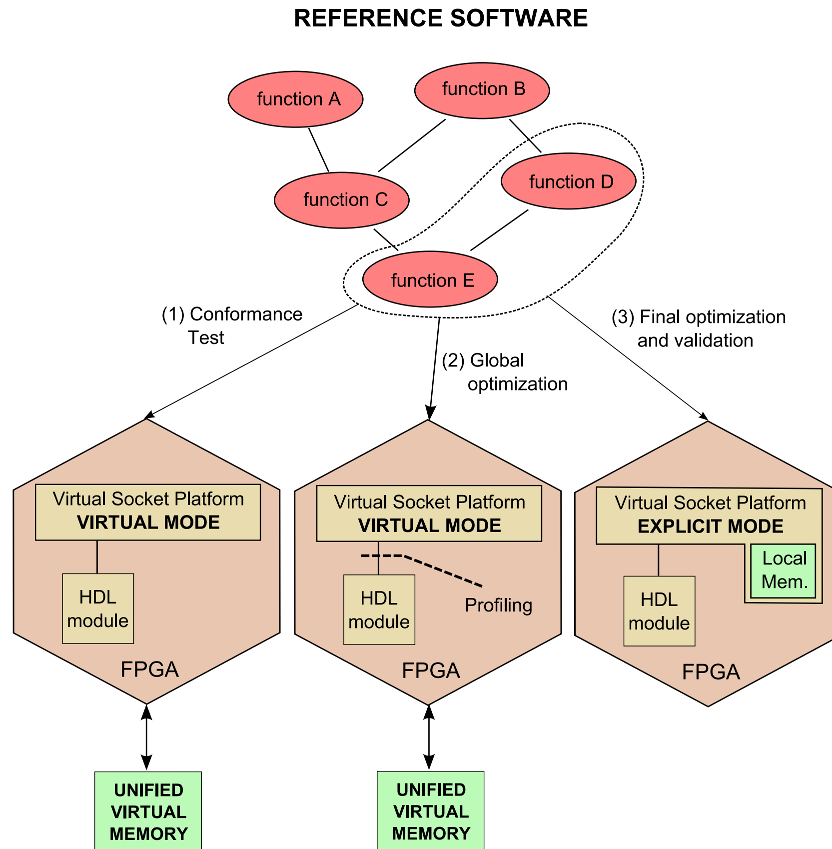

Fig. 4. the optimization methodology

sizes (height and width). All of the parameters are stored into the IP_Param array. The address generation is implicitly calculated inside of the wrapper with the different parameters. The motion estimation process is executed for each reference block and the associated search window. The first step is the validation of the module functionality. For this phase, one input memory has been implemented 5. The conformance tests with the reference SW have permitted the validation of the motion estimation process.

The second step is the optimization of the memory architecture. The goal is to reduce as much as possible the size of the internal memory without affecting the performance of the module. The different levels of cache memory have indeed an important influence on the system's performances and theyr behavious is not easily predictable for algorithms that changes during runtime. The profiling information extracted during the execution of the motion estimation process can be extremely useful to improve the architecture performance, cost and power dissipated by reducing the number of accesses to an external memory and by minimizing the internal memory size. Another possible optimization is the suppression of the latency between the two MAD evaluations. To obtain optimized matching task in parallel with a data-transfer task, the input-data memory can be split into two memory banks. A portion of the search window and the associated block can be stored in each memory bank. The processing runs on one bank while input data can be transferred simultaneously into the other one. The virtual memory extension provides the data transfer information that can be used for this optimization task.

VII. Conclusion

This paper describes the implementation of a platform capable of supporting the designer in the different steps aiming at achieving the algorithm-architecture adaptation of a processing system described by a reference software. The platform provides a seamless environment for testing hardware modules which have been transformed from the reference software into HDL hardware 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 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. RSP 2000. Proceedings. 11th International Workshop on 21-23 June 2000 page(s):78 - 82

[3] Edwards, M.; Fozard, B.: Rapid prototyping of mixed hardware and software systems, Digital System Design, 2002. Proceedings. Euromicro Symposium on 4-6 Sept. 2002 Page(s):118 - 125

[4] Pradeep, R. Vinay, S. Burman, S. Kamakoti, V.: FPGA based agile algorithm-on-demand coprocessor, Design, Automation and Test in Europe, 2005. Proceedings Publication Date: 7-11 March 2005

[5] Plessl, C.; Platzner, M.: TKDM - a reconfigurable co-processor in a PC's memory slot, Field-Programmable Technology (FPT), 2003. Proceedings. 2003 IEEE International Conference on 15-17 Dec. 2003 Page(s):252 - 259

[6] Sukhsawas, S.; Benkrid, K.; Crookes, D.: A reconfigurable high level FPGA-based coprocessor, Computer Architectures for Machine Perception, 2003 IEEE International Workshop on 12-16 May 2003 Page(s):4 pp.

[7] Schumacher P, Mattavelli M, Chirila-Rus A, Turney R: A Virtual Socket Framework for Rapid Emulation of Video and Multimedia Designs, Multimedia and Expo, 2005. ICME 2005. IEEE International Conference on 6-8 July 2005 Page(s): 872 - 875

[8] Miljan Vuletic, Laura Pozzi, and Paolo Ienne: Virtual memory window for application-specific reconfigurable coprocessors. In Proceedings of the 41st Design Automation Conference, San Diego, Calif., June 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, System-on-Chip for Real-Time Applications, 2005. Proceedings. Fifth International Workshop on 20-24 July 2005 Page(s):211 - 214

[10] Mohamed, T.S.; Badawy, W.: Integrated hardware-software platform for image processing applications, System-on-Chip for Real-Time Applications, 2004.Proceedings. 4th IEEE International Workshop

[11] Dubois, J.; Mattavelli, M.; Pierrefeu, L.; Miteran, J.: Configurable motion-estimation hardware accelerator module for the MPEG-4 reference hardware description platform, Image Processing, 2005. ICIP 2005. IEEE International Conference on Volume 3, 11-14 Sept. 2005

[12] Mattavelli, M., Zoia G., "Vector Tracing Algorithms for Motion Estimation in Large Search Windows", IEEE Transactions on Circuits and Systems for Video Technology, Vol. 10, No. 8, Dec 2000