The utilization of a fully configurable microprocessor development environment for rapid VHDL prototyping and implementation of 'C'-based algorithms

Herman Adriani, Françoise Harmsze and Henk Corporaal

TNO Physics and Electronics Laboratory
P.O. Box 96864
2509 JG The Hague, the Netherlands

Delft University of Technology
Faculty of Electrical Engineering
Delft, the Netherlands

Abstract— **In this paper we present the utilization of the Move framework during the development of a fully configurable microprocessor. The framework enables rapid ASIC prototyping of 'C' or 'C++' based algorithms using VHDL as the implementation language. The environment performs an automatic evaluation of a range of possible Move architectures. Using design constraints for area and performance, the hardware/software trade-off results in a processor optimized for either of these criteria: compared to a general purpose processor the Move processor can be much smaller giving a similar performance, or equal in size, but much better in performance. The automatic mapping of the 'C'-algorithm onto an efficient Move architecture overcomes the traditional disadvantages of application specific processor design, like long time-to-market and high non-recurrent engineering costs.**

The Move processor is based on Transport Triggered Architectures (TTAs), which means that it is composed of a transport network connecting several functional units with each other. The designer can add any specific functional unit required for the application, which is very convenient for miniaturized applications such as ASICs for mobile communication and multi-media.

To illustrate the Move processor application possibilities for rapid prototyping and hardware implementation we give two examples of ASICs using a Move-based processor.

Keywords— **Hardware/Software CoDesign, Simulation, Design environment framework**

I. Introduction

Many electronic products use embedded processors to create "intelligent" systems. Such systems have the advantage of flexibility, adaptability and often portability. To obtain an embedded system, the designer has a wide choice of processors and peripheral hardware. In many cases it is sufficient to use a standard processor and peripherals, which should be able to cover the system needs. The software actually determines the system functionality. The approach is that "any problems can be solved in software".

However, for a certain category of systems the requirements are much stricter concerning performance, area or power consumption. For such systems an optimal balance between hardware and software should be obtained. A standard hardware/software solution will not satisfy the requirements. A system satisfying the strict requirements can be obtained by hardware/software co-design. Hardware and software are balanced to each other. Traditionally, this approach can lead to high non-recurrent engineering costs and a long time-to-market.

The design goals for integration of an application specific processor and peripheral hardware are shown by the following list:

- Flexibility. The system must not be restricted to the application for which it is designed.
- Extendibility. The functionality of the system can be changed during the life-time of the system. This supports bug-fixes or future system extensions.
- Portability. A new comparable design based upon the existing design re-uses (parts of) the existing design and the design implementation should not be enforced by the availability of a target technology.
- Miniaturization and integration. The trade-off between functionality and number of components should lead to a physically small system. Integration of system functions into an ASIC further reduces the number of components.

In this paper we present the application of a fully-configurable microprocessor environment suitable for hardware/software co-design, called the Move framework. This framework has been developed by the Technical University of Delft, Faculty of Electrical Engineering, department of Computer Architecture and Digital Techniques (CARDIT). Owing to an automatic evaluation of a range of possible architectures and automatic mapping of the 'C'-algorithm onto an efficient architecture, the traditional disadvantages are avoided, while the design goals can be met.

In section II we will explain the principles of the Move architecture. The Move framework will be shown in section III and in section IV we will illustrate the design trajectory using two examples of ASICs which include a Move processor.

F. Harmsze is currently working for Philips Research, Eindhoven, the Netherlands.

II. Move processor architecture

Before we give a description of the Move processor architecture we will summarize the advantages offered by a processor based upon the Move processor architecture:

- Flexibility and expandability. The framework offers the user to design a fully configurable processor customized to the user's requirements. The partitioning of hardware and software for an (embedded) application is adjusted to the needs of the user application.
- Modularity. Specific applications may require specialized "instructions". The hardware blocks representing those instructions can easily be added to the processor. The framework is able to generate code which utilizes the new instructions.
- Scalability. The need for higher performance is satisfied by configuring the word-size, the number of bits in a word or the number of data transport buses.
- High level language support. A Gnu C/C++ compiler is part of the processor framework and is capable of generating code for each processor generated by the framework.
- Application specific instruction set. The user is able to define the instruction set layout of a generated processor. Military and commercial systems benefit from an application specific instruction set. The absence of (commercially) available debugging or emulation systems will provide a first level of security against people trying to copy the system.

The Move processor architecture belongs to the category of Transport Triggered Architectures (TTAs) [1]. A TTA can be regarded as a set of FUs and register files, connected by an interconnection network. The required data transports are programmed, while FU operations occur as side effects of these data transports. For most traditional architectures, such as RISC and VLIW, the program, has to specify the FU operations rather than the data transports, which means that the compiler has no control over the data transports. For TTAs, the compiler can control the data transports, which gives many advantages for data movement and operation scheduling.

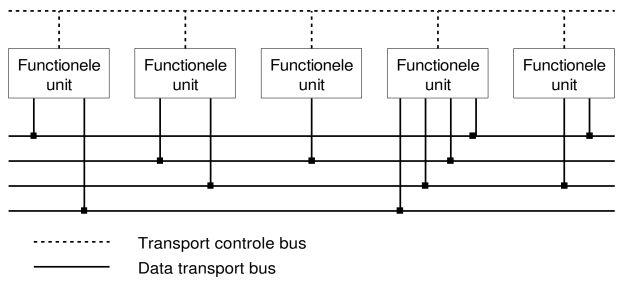

Fig. 1. Move processor architecture

Figure 1 gives a schematic overview of the Move processor architecture. This figure shows that the Move consists of Functional Units (FUs), data transport buses and transport control. An FU performs an operation on the available data. The designer determines which and how many functional units are present in the design. The data buses transport data between FUs and the transport control buses enable conditional Move operation. Examples of standard FUs are:

- Instruction fetch unit, to read the instructions from memory.
- Load/Store unit, to read data from and write data to external memory
- Register file unit: general purpose registers for use by the processor
- Integer unit: add and subtract operations
- Logic unit: and, or, xor operations.
- Data transport buses. The buses perform parallel transport of data between functional units.
- Control bus. The control bus controls the data flow through the data transport network. Each bus is controlled by locking and conditional transports.

A functional unit contains one or more input registers and output registers. The instruction set of a specific Move processor determines the transports from output registers to input registers. The registers are subdivided into three types:

1. The Operand (O) registers. An O register is an input register containing data to be used during an operation. Writing to an O register does not trigger an operation.
2. The Trigger (T) registers. A T register is an input register containing data being used during an operation. Writing to a T register results in activating the operation in the FU.
3. The Result (R) registers. An R register contains the result of the operation, which was started after a write to the T register. The contents of the R register can be transported to an O register or a T register.

While the Move framework offers a set of standard FUs which enables the design of generic Move processors, the user is able to design his own FU. This FU can be incorporated into the overall framework which, in turn, offers evaluation of the design including the the user specific additions. The generated code will use the user specific FU if the 'C'-algorithm uses the specific functionality.

III. Move framework

TTAs have served as a template for the Move framework. The Move framework has been designed to allow for easy exploration of the architecture design space for any given application. The basis for the framework is given by the architectural description which is provided by the user. The framework then evaluates, optimizes and generates the final processor hardware description. The evaluation and optimization process is guided by the user application software written in a high level language such as C or C++. The Move framework consists of 3 subsystems:

1. The Explorer
2. Hardware subsystem
3. Software subsystem

Figure 2 shows an overview of the framework.

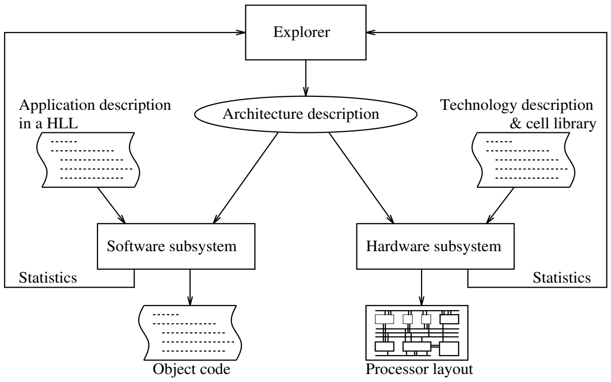

Fig. 2. Overview of the Move framework

The Explorer
The explorer is used for the automatic evaluation of a range of generated processors. It uses the 'C'-based algorithm for a certain application. The criteria and the weight for these evaluation criteria is defined by the user: for example chip area, number of pins and power consumption. After evaluation, the explorer shows an overview of the possible configurations, with the results for each of the criteria. The designer chooses the configuration which best suits the application needs.

Hardware subsystem
The hardware subsystem generates the processor layout from the architectural description. Figure 3 gives an overview. The processor generator generates VHDL code which can be converted to an ASIC layout by a silicon compiler (synthesizer).

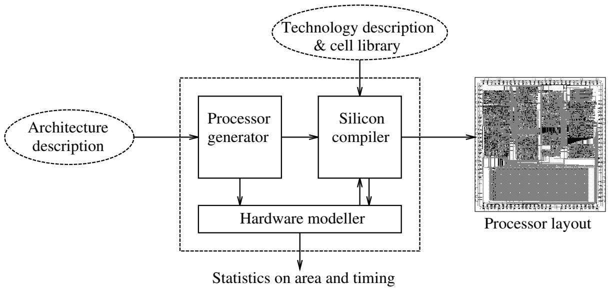

Fig. 3. Overview of the hardware subsystem

Software subsystem
The software subsystem, as shown in figure 4, consists of the compiler and the simulator. The front-end is a Gnu C/C++ compiler, delivering sequential TTA code to the compiler back-end. The sequential TTA code is used to collect profiling information which is used by the compiler back-end to optimize the TTA code for the specific architecture (parallel TTA code).

IV. Move applications

The move processor framework has been used during the development of a processor for two embedded systems:

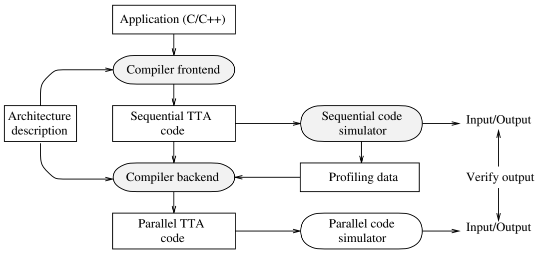

Fig. 4. Overview of the software subsystem

- VideoEnhancer. A system for contrast enhancement in monochrome video signals.
- Datalogger. A system for data acquisition and data storage.

Both systems use the same processor. We defined the following requirements for the development of the processor:

- Hardware/Software tuning of processor is based upon the VideoEnhancer application due to real-time constraints for certain calculations being performed.
- 16 bit data processing. Both the VideoEnhancer and the Datalogger applications are targeted to 16 bit data word processing.
- Simple instruction set. The operations performed by both applications are not complicated. An instruction set with complex operations like multiply, divide or barrel-shift is not necessary.
- Availability in VHDL. Due to the portability design goal, the processor must be available in VHDL for simulation and synthesis.
- Easily extendible. Both applications need specific processing hardware besides the processor. This hardware must be easily attachable to the processor.
- High level language support. Small applications can be coded in assembler. But larger applications do need the support of a high level language, preferably C or C++.

A. VideoEnhancer

The VideoEnhancer ASIC is utilized by systems which need monochrome contrast enhancement in real-time video signals. The systems are used where poor environmental conditions imply the need for video enhancement and the user cannot wait for better conditions, for instance: nightfall, fog, rainy weather or smoke. Surveillance of industrial or harbour terrains or buildings and many other applications may need the contrast enhancement provided by the VideoEnhancer. Figures 5 and 6 show images taken before processing and after processing respectively.

The ASIC operates with a minimum of external components which allows integration into a video camera. Combined with the real-time operation capacity of the VideoEnhancement, surveillance applications are easily extended with real-time contrast enhancement. The level of enhancement is automatically adjusted according to the ever changing environmental conditions.

The VideoEnhancer ASIC is also prepared for contrast enhancement of color video signals.

Fig. 5. Image of the dunes in the Hague on a foggy day, before processing

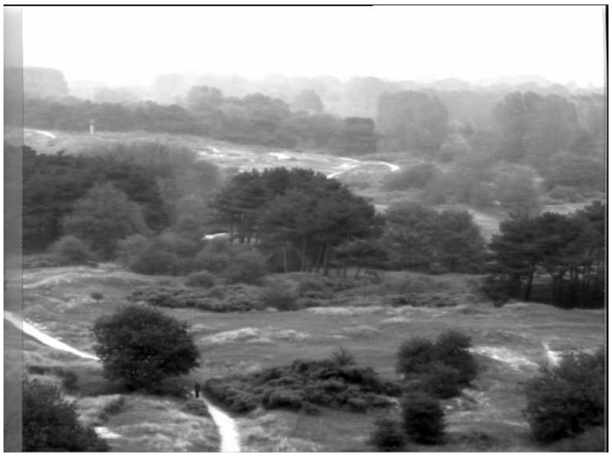

Fig. 6. Image of the dunes in the Hague on a foggy day, after processing

B. Datalogger

The Datalogger ASIC is aimed at systems which autonomously perform data acquisition and storage for longer periods of time (weeks to months). The number of external components is kept to a minimum by integrating as many functions into the ASIC as possible. The support for low power operation enables the possibility of long periods of autonomous operation.

Low power control is performed by disabling the power fed to the processor and its immediate hardware support (memory access controller and digital I/O). A small portion of the ASIC is kept active. An interval timer and an external trigger detector enable the controlled wake-up the ASIC. Acquisition and storage of data is performed during the active period of the complete ASIC. Low power mode is re-activated after performing acquisition and data storage.

C. TNO-FEL μMove processor

Taking the constraints and the C-program from the VideoEnhancer we developed the so-called μMove processor, using the Move processor framework. The μMove processor has been extended with two application specific Functional Units: an RS232 interface and a special Input/Output interface. Extra hardware has been added for each application: the VideoEnhancer uses hardware for video processing and the Datalogger contains a power control unit.

The μMove processor has the following characteristics:

- 16 bits processor architecture
- Single data transport bus
- Instruction fetch unit for 64 KB code
- Load/Store unit for 128 KB data and support for multi-megabyte memory banks
- Integer unit (add/substract)
- Logic unit (AND/OR/XOR)
- Simple shifter unit (left and right shifts)
- Two compare units
- Immediate unit for 6 bit immediates
- Guard unit for conditionally executed moves
- Double buffered RS232 interface
- Digital input/output

V. Conclusions

As presented in this paper, we have successfully utilized a method of hardware and software co-design during the development of the two described applications. The Move processor framework provides an environment for designing a fully configurable application specific processor tuned to our application and algorithmic needs. Based upon the 'C' algorithm of the VideoEnhancer application and the tools of framework, a trade-off between hardware and software has been made. The availability of a VHDL description of the generated Move processor enabled co-simulation of the ASIC in combination with the target system software right from the start of the development phase. This offered us the opportunity to compare the hardware and software combination with with the set of application requirements.

Portability is an important issue during system design. The Move processor framework does offer portability. It is possible to design a range of completely different processors for different types of systems without touching the underlying processor architecture and the implementation technology. The support by the processor independent C/C++ compiler tool is of great value. Whatever processor is generated by the framework, the compiler tools produce code optimized for the generated processor.

The use of the Move framework offers a good alternative to the application of standard processors for systems requiring a high level of integration and miniaturization.

References

[1] Henk Corporaal *Transport Triggered Architectures, Design and Evaluation* Thesis Delft University of Technology, ISBN 90-9008662-5