The Allele Search Lab to Improve Heterogeneous Reconfigurable Platform Design Skills

Yvan Eustache, Jean-Philippe Diguet, Guy Gogniat
LESTER, CNRS/UBS University, Lorient, France, (email : yvan.eustache@univ-ubs.fr)

Abstract

This paper describes a co-design course that is performed within the Université de Bretagne Sud in Lorient, France. The objective of this course is to improve the students embedded system design skills through a pedagogical application. The course starts with a lecture about general concepts and tools in the domain of Electronic System Level design. Two axis are followed: firstly co-design and high level synthesis concepts are presented and secondly new architectures offered by embedded systems are described. Throughout this course, a lab provides the students with practical experiences in designing HW systems and SW tasks. The students explore different architecture alternatives, implement them and evaluate impact of their choices in terms of performance and flexibility. Finally, a design contest concludes the course. The main objective is to design the fastest allele search embedded system.
Key Words : Reconfigurable architecture learning, Hardware software architectures, design space exploration, DNA applications.

1 Introduction

Nowadays, commercial reconfigurable and heterogeneous FPGA-based platforms can be used to design high performance embedded systems. Build from processors, for flexibility reasons, and hardware accelerators, for performances issues, such complex systems require designers with a large scope of skills from high level CAD tool down to low level specification and optimization in both software and hardware areas. The development of heterogeneous systems generally starts from a functional validation using a C specification. Extensive profiling then allows the identification of critical sections within the application that are potential candidates for HW acceleration. Coprocessor and/or accelerators can be targeted to provide various tradeoffs in terms of area overhead and speed-up factor. Parallelism exploration and communication penalties are also very important aspects that must be considered during the design cycle to reach an efficient solution.

To address these issues this paper presents a comprehensive design experience on which are based a lab and a project. Section 2 describes the course organization and Section 3 describes the DNA application and three hardware / software implementations used to illustrate the design key issues that must be acquired by students. Design results and students feedbacks are given according to the different design steps. A discussion and some perspectives conclude the paper.

2 The Electronic System Design Course

This ESL (Electronics System Level design) course is based on three steps, which are general concepts lectures, a lab and finally a personal project. The objective is to explore an embedded system design flow of heterogeneous reconfigurable architectures. The main goal is to improve architecture and application co-design skills, from architecture specification, parallelism exploration, communication analysis and C/VHDL specifications down to bitstream and binary programs. The study is based on an interesting DNA identification application, which offers a large panel of design alternatives.

2.1 Course Description

The course starts with a lecture addressing Electronic System Level design. An introduction presents the needs for co-design resulting from the number of embedded systems and the complexity of these circuits (multi-processors dealing with multi-communications for multi-peripherals) under constraints of cost, performance, energy limitation and time to market. Then the lecture focuses on two major topics: system level design tools and new concepts of design. The first part addresses the question of computer aided co-design, computer-based tools that assist engineers in design space exploration, application models, HW/SW partitioning, HW/SW co-simulation and System/HW/SW high to low level estimations. The aim of the second part of the lecture is based on one of the LESTER research area, namely co-design of OS-based reconfigurable systems where the abstraction level of the OS services targets the SW and the HW tasks. Network on Chip (NoC), configurable processors and adaptive architectures are also presented as alternatives of current systems (bus hierarchy, general purpose processor and dedicated architectures).

Following the lecture, a lab session is proposed around The Altera Nios II Design Kit with Stratix II board. Lab sessions are organized around three steps and are followed by a personal project. In a first

step, students have to implement a DNA application for allele counting as a software application running on the Nios II processor. In a second step, students have to speed up their architecture by implementing a co-processor dedicated to the application critical part (namely Custom-Instruction as an extension of the native instruction set). In a third step, a dedicated slave module is designed and added to the processor Avalon bus in order to alleviate processor workload and speed-up DNA allele counting. Students have to develop and to evaluate performances (e.g. time, area, memory use) of the three implementations for the given application. The course ends with a design contest based on the Stratix II board, namely this is a project where the challenge is to design the fastest architecture they can imagine using previous concepts (Figure 1).

2.2 Audience and Prerequisites

This course is mainly followed by master students in electronic system design. Students have a general background in computer sciences including digital electronics, HDL and software. Most of them have already developed some parts of embedded systems (web server, robotics, Viterbi codec,...) using micro-controllers or FPGAs during previous training courses.

Students must have basic knowledge on digital designs techniques and procedures (e.g. FPGA design flow from RTL architecture specification, synthesis, Place & Route steps, to the download of the bitstream on a device). Good knowledge on how to write VHDL for RTL Synthesis and C programming skills are also required. However no knowledge on genomic is necessary. The book [1] takes a modern structured, layered approach to understanding computer systems.

2.3 Course Goals

The main goal of this course is to improve the students knowledge and skills in the domain of heterogeneous reconfigurable systems design. Throughout this course, the lab exercises provide them with practical experiences in designing HW and SW tasks. The students explore different architecture alternatives (co-processor, accelerator, computation parallelism, communication overhead) and evaluate impact of their choices in terms of performance and flexibility. They also manage a complete heterogeneous programmable device design flow using VHDL and C languages, from design entry through synthesis and place & route to device programming, and the software compilation and programming of the Nios II processor. After completion of this course, students should have a firm understanding of FPGA technology and the relevant issues surrounding its use in system design and should be prepared to conduct fruitful projects related to architectures, tools, and applications on programmable logic. The lab developed skills in hardware design and software programming with commercial design tools. Moreover the implementation of the system on a board (not limited to the simulation) improves the programming skills (VHDL RTL) while being a good educational approach. To conclude this part, teaching embedded systems courses should not be just a lecture since experiences is absolutely necessary to handle real problems. Labs and projects are the main parts of the learning process. We observe that students skills and know-how are greatly improved after the implementation of a real-system. During this course, students will acquire knowledge and skills in the domain of heterogeneous reconfigurable architectures, co-design methodologies and tools for hardware / software simulation, synthesis, programming and debugging.

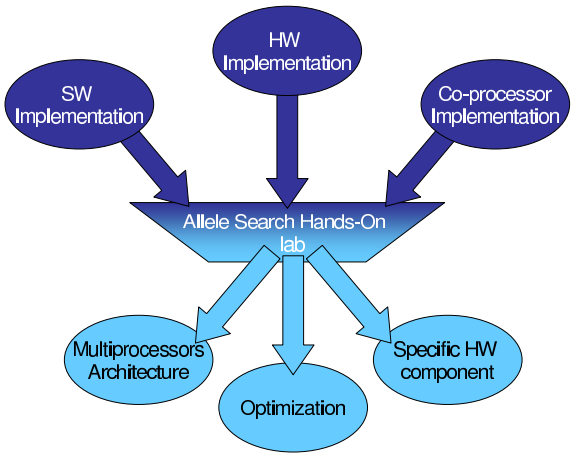

Figure 1: Overview of "the lab to the contest" co-design course

3 The Allele Search lab

This lab was developed to teach the students about how to be a good system design engineer. It deals with the topics of hardware & Software designs and reveals the different difficulties of HW/SW communications or profiling of the application while being attracting. The Allele search is a good exercise, neither too hard to be realized in a three-lab-session nor too easy to raise a maximum of engineering problems. Programmed in software, it needs the utilization of data pointers, memory management and loop optimizations whereas the hardware implementation raises the questions of HW / SW communications and finally the co-processor can be designed as a synchronous or asynchronous module according to the critical path. The system has to respect the specification and the debugging step is needed. The Altera development tools allow the HW & SW co-debugging respectively with a signal analyzer and a traditional software debugger. Step by step, the program evolution and the HW module signals can be

dynamically observed. The final objective of the lab is to optimize the design of the system to speed up the computation. An evaluation of the improvement is needed. The system is then characterized in term of the execution time, memory size or area utilization. Three four-hour guided sessions are required for the software programming, the co-processor utilization and the hardware module design. A design contest then follows to allow students to extend their knowledge and practical experience by themselves. The students have one month to improve their systems mainly in term of execution time while respecting the specification. All optimizations or architectures relying on the Nios II processor parametrization to multi-processors or multi-HW accelerators are allowed and encouraged. The limit of this lab is that, considering the important number of logic cells composing The Altera Nios II Design Kit with Stratix II board, the students are not directly confronted to area constraints. The size of the hardware implementation is not a critical issue during the lab however the resources allocation limits might be raised during the design contest. In a same way, an hard time constraint for the hardware resources might be considered during the contest.

3.1 DNA allele counting application

A human is composed of approximately 220 different kinds of cells e.g. neuron, red blood cells and so on. All of them and more generally all known cellular life are based on nucleic acids that contain the genetic instructions specifying their biological developments (reference [2]). The genes composing the genome is a unit of heredity. An alternative of a gene, characterized by a sequence of nucleotide, is named an allele. There are four types of nucleotide for the DNA: Guanine (G), Adenine (A), Thymine (T) and Cytosine (C). The abstract objective of the Allele Search lab is to count the number of occurrences of an allele in a genome. In other terms, the objective is to count the number of a particular sequence of nucleotide (e.g. "ACTGGACT") into a table of nucleotides randomly selected. For this lab, the allele is composed of four nucleotides. Each of them composing the genome and the allele is an eight-bit variable coded according to the ASCII character set (A = 41 hex, C = 43 hex, G = 47 hex and T = 54 hex). The allele reference is then a 32-bit word. It is a pedagogical exercise of a usual computation of pattern matching that appeals to a "sliding window" method (e.g. the allele "ACTA" exists one time in "TACTATCA") which can be parallelized in order to allow multiple computations at the same time. The hardware accelerator will be then more efficient in term of execution time. The Allele search application is limited to a forward search. No backward search is computed, for instance the allele "ACTG" is counted once in the following sequence "TACTGTCA".

3.2 The Altera Nios II Development Kit, Stratix II Edition, the tools and the board

Each student has to design the system on an embedded board composed of a processor and logic cells into an unique FPGA component. The selection of the Altera target [3] was guided by the available boards within the Electrical and Computer Engineering department. A strong point of the Altera Nios II processor [4] compared to an equivalent commercial development kit (e.g. the Xilinx Virtex II development kit with microBlaze processor) is the possibility to extend the instruction set with custom instruction, an hardware module called by a software instruction. Moreover the development framework is simple enough to rapidly acquire first design skills. The Nios II processor is a good support for the design lab. This embedded processor is a general purpose RISC processor core with a 32 bits instruction set, data path and address space. It provides 32 external interrupt sources, access to on-chip peripherals and interfaces to off-chip memories and peripherals and supports also a Real Time Operating System, RTOS, as μC/Linux or μC/OSII. It is provided as a soft design IP and can be targeted to any Altera FPGA family. The soft-core allows thus the configuration of the processor improving the flexibility. In others terms, the engineer builds easily an exact-fit processor system.

The students define the architecture of the system with the SOPC-Builder tool. The parametrization of the processor is facilitated by a user friendly interface. three standard processors can be implemented according to the trade-off between logic cells allocation and performance. The main differences are the utilization of data or instruction caches (which sizes can be parameterized), an hardware multiply and divide module, a branch prediction, and the pipeline size. The custom instructions are also defined. The HDL module is called and declared as a custom instruction. Different kinds of custom instructions exist : combinatorial, synchronous, asynchronous, multi-cycle. In a same way, the components connected to the Altera's proprietary bus, namely Avalon Bus, can be defined (e.g. off-the-shelf components like the memories, the timers, the Ethernet controller and the user hardware modules...). Thus, the students have to plug the hardware accelerator they design on it. Finally the SOPC-Builder tool builds the system interconnect fabric optimized for the requirements and generates the corresponding HDL and synthesis files used for the hardware design and the software programming.

The second step of the flow is the generation of a bitstream to configure the logic cells composing the FPGA while using the Altera Quartus II tool. The synthesis and the fitter computation parameters (the trade-off between the area and the speed) allow the optimization of the system characteristics. The students may observe the resulting resources allocation

with the floorplan tool and the reports of the synthesis and fitting steps give information concerning the clock frequency and the area. The bitstream is finally downloaded into the FPGA via the JTAG connection. During the run, the Quartus SignalTap service acts as a logic analyzer tool to observe the registers and signals evolution.

In parallel, the programming process may be done with the NiosII IDE. The software is programmed in C language. An Hardware Abstraction Layer (HAL) system library, generated by the SOPC-Builder tool, provides services as device drivers to reach each device in the system or an abstracted Application Programming Interface (API) to define a standard interface (e.g. the device access, the interrupt handling...). Thus the software may communicate directly with the hardware modules via an unit of standard instruction. After building, the binary program is downloaded into the memory of the FPGA and started. Nios II IDE provides also a usual software debugger environment.

3.3 A First Implementation: the Software Computing research of Specific Allele

Software is the most comfortable way to develop a first version of an embedded system. The programming time, the building and downloading time and the debugging time are negligible compared to the design and debugging time of an HW module (ratio 1mn/30mn). And moreover the students background is mainly oriented toward software implementation. Hardware languages and implementations are studied later. So the first implementation concerns the software programming. The students have to develop a software application counting the number of occurrences of a given 4-character pattern among a 200,000-nucleotide table stored into an external SDRAM memory. The allele reference is fixed in a constant variable as a concatenation of four nucleotides (a word of 32 bits). The SW function (figure 2) declares two pointers initialized at the physical data address in memory. In the search loop core two successive 32 bits words are pointed and four comparisons are computed sequentially. A software counter is incremented if the comparison returns a perfect match between the reference and the current allele. The pointers are incremented and the process iterates. A large scale hardware timer, which has been previously designed is provided to students as an HDL file, that they have to implement as a slave connected to the Avalon bus.

The software implementation is profiled using the hardware timer which counts the number of clock cycles from the start to the end of the process. This first implementation raises different issues. Students can observe the question of memory mapping, pointers utilization at physical level and emphasizes the influence of loop core choices to reduce the execution time. Moreover students can handle the problem of computation parallelism which is limited by data dependencies especially while designing the sliding window.

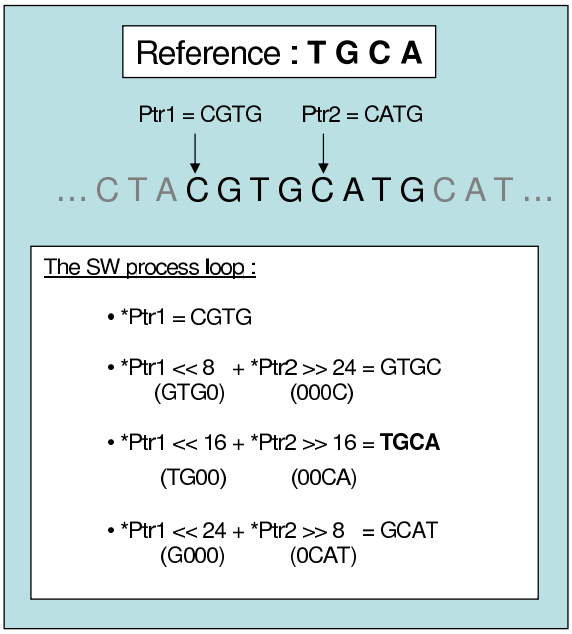

Figure 2: The allele search programming in SW

3.4 The Co-processor Implementation

The software implementation can be improved with the utilization of a co-processor. In this case, the matching function (which corresponds to a sequence of instructions in assembler) is not done sequentially by the processor but performed through a single instruction call of an hardware module. This is equivalent to the integration of an hardware accelerator into the Arithmetic and Logic Unit (ALU) of the processor and is named Custom Instruction (CI) as an extension of the Nios II initial instruction set. The parameters of the function call (maximum two 32 bits words) are contained in two registers of the processor and the co-processor returns the result (a 32 bits word) in another processor register. According to the critical path and its static or dynamic behavior, the CI is defined as combinatorial when the result is computed in a single clock cycle, as multi-cycle synchronous or asynchronous when the computation requires two or more clock cycles to complete and respectively when the number of clock cycles is fixed or variable. In this project, the hardware module is composed of two 32 bits inputs (eight nucleotides), an 11 by 8 bits-size shift register (an eleven nucleotides-size register), eight comparators in parallel, an eight inputs adder and a result register. The eight comparisons need to store three precedent nucleotides. The software program has to call the co-processor with parameters read from the memory. For each call, the hardware module receives eight nu-

cleotides and shifts the last three precedent nucleotides (figure 3),the comparisons are done in a combinatorial process then a second clock cycle allows the additions. The co-processor is thus a two-cycle synchronous or asynchronous CI. The result is added during each process cycle and returned to the software at the end.

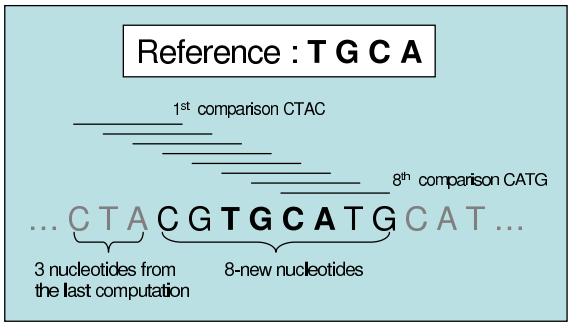

Figure 3: The co-processor allele search principle

The co-processor based system is an interesting experience for improving different skills such as the matching between application and instruction set. It also points out the problems of critical paths along with synthesis tools. Actually, the adder can be implemented as a seven-adder chain or as a seven-adder tree. The combinatorial logic between two registers, namely the critical path, is more important in the second case. If the co-processor is designed as a synchronous CI with an adder chain process, the result acquired after two cycles may be corrupted (the adder result will not be computed before the next clock cycle). Three solutions are explored to solve this problem. Firstly, students perform a tree height reduction in order to minimize the critical path. Secondly they learn how to define synthesis tool parameters (synthesis, fitting) in order to improve final design clock-speed and thirdly they design an asynchronous co-processor, which is independent from the clock period.

3.5 The Hardware Computation

The last implementation presents another architecture for the allele search computation. This system is composed of a processor as a unique master plugged onto the Avalon bus and a slave hardware accelerator. To minimize the design and debugging time during the lab, the hardware accelerator specification is limited to one reading and writing slave port. Thus, the unique master, the processor, has to feed the hardware module with data and to read back the result from a register of the slave. The functional behavior of the accelerator is close to the co-processor. The differences are the number of parallel comparisons which are performed simultaneously, which increase from 8 to 128. Consequently, the size of the shift register (128 new nucleotides + 3 precedent nucleotides) and the adder (128 inputs) are also increased and require the use of the GENERATE VHDL instructions. The first state of the process Finite State Machine (FSM) deals with the loading of the data. The HW module receives 128 nucleotides sequentially via the slave port and strore them. The three precedent nucleotides are shifted. The comparisons are done in parallel in combinatorial processes. Then, the addition is computed during the second state of the process FSM. This process is repeated until the end of the genome.

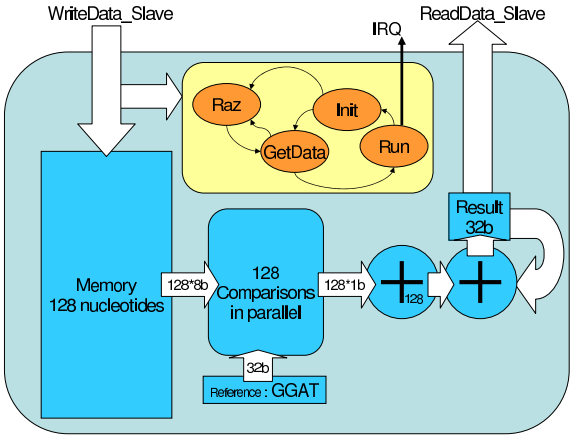

Figure 4: The HW allele search component

This last implementation raises the problems of HW/SW data communications. The slave hardware module reduces the design time but is not fully optimized. Indeed, in this architecture, the processor consumes a lot of time to transmit data to the accelerator and most of the time, the hardware module does not compute anything and waits data. Two solutions can solve this limitation: the accelerator is still a slave component but it contains a ping-pong shift registers that provides a double-buffering method for acquiring data. In the other solution, the hardware accelerator can reach the memory via a master port and a Direct Memory Access mechanism. The processor controls the accelerator via the slave port. This implementation deals also with the questions of the Nios II exception handling service. At the end of a process cycle, the hardware sends an IRQ signal that interrupts the processor computation. The exception handling has to read the result register via the slave port and to acknowledge the interruption to pull down the IRQ signal.

3.6 Results

The profiling of the different implementations (Table 1) shows the number of clock cycles consumed for the allele search. The results show that the hardware acceleration (with a co-processor or an HW component) exists and is important compared to the execution time of a SW program. The HW accelerator and the HW

co-processor solutions are not so distant. However the level of parallelization are not comparable (ratio 128/8). The difference between these two implementations is the data feeding. Whereas the co-processor receives data through the internal register of the Nios II processor like the ALU, the data for the HW accelerator must be written by the software program through the Avalon bus.

Implementation	clock cycle	speed up
SW	20 065 046 cy	
Synchr. co-pro.	4 115 666 cy	4.87
Asynchr. co-pro.	4 158 600 cy	4.82
HW	3 321 165 cy	6.04

Table 1: Implementations profiling

The table 2 presents the time spend by the hardware accelerator when it waits data and when it computes them. The weight of the HW/SW communications is significant compared to the computation and it explains the results of speed-up (table 1) between the co-processor and the HW experiments.

HW State	clock cycle
GetData	1319 cycles
Run	2 cycles

Table 2: Implementations profiling

3.7 The design Contest

At the end of the lab, the students have done the experiments of software programming with memory accesses through pointers, a co-processor implementation with a first version of an hardware module and an hardware accelerator which deals with HW/SW communications and synchronization by IRQ. With the skills of computation in HW, memory mapping, memory accesses and HW/SW communications and synchronization, the students are ready to design the faster allele search engine. Original architectures are allowed as long as they respect the specifications (the genome is contained in a memory, the allele search is fixed, the result provides the number of occurrences of the allele in the genome). Three kinds of optimization can be targeted : the architecture, the parametrization and the data. The architecture exploration raises the following questions: "processor or hardware accelerator with DMA and master ports ?", "architecture mono, multi-processor ?", "memory mapping and management"... One modification is expensive in term of design time but can significantly improve the system execution time. One the other side, the processors can be parameterized to design an exact-fit component. Moreover, synthesis and fitter parameters can optimize the system (speed or area preferences). Finally, the students can modify the data with a pre-processing (e.g. the translation of the 8 bits-ASCII data into an 2 bits data (four kinds of nucleotide)) and compute the allele search on the lightweight genome. The contest is currently in progress and more accurate results will be presented in the final paper.

4 Conclusion

In this paper, we have presented a co-design course that has been carried out within the "Université de Bretagne Sud", in France. It deals with the topics of the design of embedded systems on an heterogeneous reconfigurable platform (processors + hardware accelerators on chip) through a contest approach. The course and the lab allowed the students to improve their knowledge and skills in system design applied to a pedagogical pattern matching application. The lab raises the communication, synchronization, programming and memory management difficulties through the pedagogical example of the allele search. They had to handle HW/SW architectures exploration and design tools optimizations which provide a good overview of an embedded system design. A design contest is then organized between each student with the main objective to design the fastest allele search engine on an embedded board. The students have to explore different architectures alternatives, to program them, to evaluate the computation times and to optimize the system. Currently, within the LESTER laboratory, a second project is performed based on partial dynamic reconfiguration of FPGA architectures. In the future, we plan to combine the two projects. Thus, according to the size of the allele reference, different co-processors or hardware accelerators could be implemented at runtime.

References

[1] Andrew S. Tanenbaum, *Structured Computer Organization*, Prentice Hall, 5th edition edition, 2005.

[2] J.W. Dale and M. von Schantz, *From genes to genomes: Concepts and applications of DNA technology*, Wiley InterScience Editions, Biochemistry and Molecular Biology Education, 3 edition, 2003.

[3] Altera, *Startix II Device Handbook*, 2004, Available from http://www.altera.com.

[4] Altera, *Nios II Reference Handbook*, 2007, Available from http://www.altera.com.