Verification of a Microcontroller IP Core for System-on-a-Chip Designs Using Low-Cost Prototyping Environments*

Stephen Schmitt[1]
[1]Eberhard-Karls-University of Tuebingen
Departement for Computer Engineering
Sand 13, 72076 Tuebingen, Germany
schmitt@informatik.uni-tuebingen.de

Wolfgang Rosenstiel[1,2]
[2]Forschungszentrum Informatik (FZI)
Microelectronic System Design
Haid-und-Neu-Str. 10-14, 76131 Karlsruhe, Germany
rosenstiel@informatik.uni-tuebingen.de

Abstract

Rapid prototyping is a fast and efficient way for the functional verification of Systems-on-a-Chip in an early stage of the design process. Because of the rising part of software in those systems the use and reuse of microcontroller IP cores is necessary to keep development cycles short. Today, prototyping of such IP cores is done with large and expensive hardware emulation machines consisting of many processor or FPGA-based prototyping boards. In this paper the authors describe an alternative prototyping method for microcontrollers using one low-cost FPGA-based prototyping board. The method is based on the efficient usage of all resources of the prototyping system to emulate special parts of the microcontroller.

1. Introduction

Software is playing an increasingly important role in the development of Systems-on-a-Chip (SoC) today. It helps to keep the SoC more flexible and adaptable to new market needs without the disadvantage of a chip redesign to add or change features of the system. Therefore, microcontroller IP cores are an integral part of SOC designs. For example, the StarIP DesignWare Library from Synopsys [9] offers PowerPC 440, NEC V850E, C166, TriCore1 and MIPS32 microcontrollers with different capabilities. The verification of these cores is done with Vera [10] testbenches which offer models of the microcontroller environments like RAM and ROM or realize build in monitors used for tracing signals.

The verification of the whole system and the verification of embedded software running on those cores with these traditional testbenches is, however, very time consuming. For example, the 70 delivered test programmes with the TriCore1 DesignWare component take about 6 to 10 hours for simulation. Therefore, for the verification of hardware and the debugging of software in an early stage of the design process other verification techniques besides simulation are needed.

Rapid prototyping is a fast and efficient way for the functional verification of SOC's in an early stage of the design process. For example, the verification of embedded software can be done with instruction set simulators (ISS). Unfortunately, ISS have a few disadvantages. One problem is the integration of the ISS with additional hardware modules using a hardware simulator. This will decrease the simulation speed. Another problem is the integration of the ISS as a prototype into the target system.

Today, prototyping of hardware IP cores is done with large hardware emulation machines consisting of many processor or FPGA-based prototyping boards providing abundant resources [6, 2]. These emulation systems have the tendency of being very resource consuming and expensive. On the other hand, a couple of low-cost FPGA development boards exist [3]. These boards offer a very efficient platform for prototyping while keeping the budget low. Unfortunately, their resources are limited and several problems must be solved when using such prototyping boards for hardware emulation. In our contribution we point out the problems arising when such low-cost prototyping boards are used, show state of the art solutions and present an alternative approach to solve the resource problem.

State of the art solutions have specific disadvantages. Some try to use a lot of specialized hardware leading to high emulator costs, others try to use common of the shelf hardware leading to a decrease in emulation speed. We propose a solution to the problem based on the efficient usage of all the resources of the prototyping system to get the best cost/performance ratio out of the prototyping system.

To show the usefulness of our approach we discuss the prototyping of the TriCore1 microcontroller IP core from

* This work was partially funded by the DFG priority program Embedded Systems and Infi neon Technologies AG.

Infineon using a low-cost FPGA-based emulation system called Spyder as a real world example. Special care was taken on an efficent usage of the resources provided by the prototyping platform to emulate all parts of the microcontroller and its environment.

In *section 2* the TriCore1 microcontroller IP and the rapid prototyping system Spyder are described in more detail. *Section 3* shows problems and solutions which must be solved during the integration of IP cores in FPGA's. Then a resource aware prototyping method for the above mentioned real world example is discussed in *section 4*. Results of our work are presented in *section 5* followed by a conclusion.

2. Microcontroller IP cores and rapid prototyping systems

The embedded systems market faces a huge amount of microcontrollers ranging from low-cost 8-Bit to high-end 32-Bit microcontrollers with large RAM's, ROM's, MMU's and several external interfaces integrated on one chip. Originally the microcontrollers were developed with specific fabrication processes and applications in mind (ASIC's), but should now also be reused as IP cores in new SoC designs. This introduces problems when a prototyp of such an IP core should be implemented in an FPGA.

To show problems and solutions of this task we use the TriCore1 IP core from Infineon [4] and the Spyder rapid prototyping system [12] as an example.

2.1. TriCore1 microcontroller IP core

The TriCore1 is a single-core 32-bit MCU-DSP architecture optimized for real-time embedded systems. The processor is configurable in terms of on-chip memory, caches and the presence or absence of a MMU. It features:

- 32-bit load-store Harvard architecture
- Superscalar execution
- 16 address and 16 data registers
- Fast context switch and low interrupt latency
- 16-bit and 32-bit instruction formats

The architecture of the microcontroller core is shown in *figure 1*. Besides the Scratch Pad RAM (SPR) connected to the P-MEM and D-MEM interfaces additional memory can be attached via the local memory bus (LMB). Peripheral and slow modules are connected to the flexible peripheral interface (FPI) bus. The size of the SPR, caches and TAG's is configurable. The core can be implemented with or without an MMU.

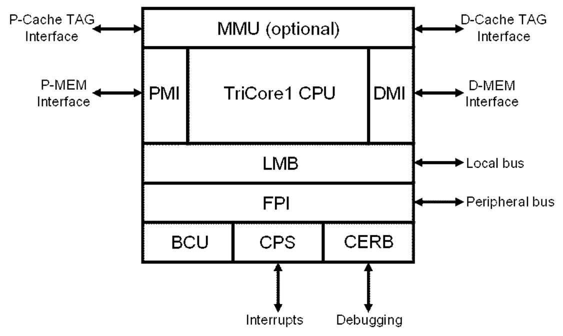

Figure 1. Architecture of the TriCore1 IP core [5].

Infineon Technologies offers the TriCore1 together with a reusable verification infrastructure for SoC development [5]. The environment consists of the RTL code of the Microcontroller, C/C++ compiler, instruction set simulator and simulation models of peripheral modules like memories.

As mentioned above the TriCore1 core is also available as a DesignWare Star IP component with an additional Vera verification environment from Synopsys [9]. There the peripheral modules are implemented in Vera and allow a transparent execution of programmes. The integration of additional peripheral modules can be done via the *Flexible Peripheral Interface (FPI)* bus developed by Infineon or with an AMBA interface which is connected over a AMBA-FPI bridge to the FPI bus.

Since the whole system is simulated with Vera and Modelsim the verification time is extremly high even for small programmes. Therefore the simulation of large portions of code using, for example, a RTOS can take a couple of hours or days.

2.2. Spyder rapid prototyping system

To overcome the problems of the simulation of microcontrollers with software programmes a FPGA-based emulation system can be used. To keep the costs of the emulator small a general purpose low-cost prototyping system should be considered. We took the FPGA-based Spyder-Board [12] as an example for a low-cost rapid prototyping system to integrate parts of the TriCore1 microcontroller IP core.

The Spyder-Board is a PCI-based add-on card for PC's. The board can be equipped with Xilinx FPGA's XCV400 to XCV2000E, thus offering 400 thousand to two million system gates. As a rule of thumb, one can say that this value must be divided by ten compared to ASIC gates. This means that with a Virtex XCV2000E FPGA ASIC designs of approximately 300.000 gates can be realized.

To overcome the problem of limited resources the FPGA vendors tend to integrate additional components into the FPGA's which can be used instead of the general purpose slices. For example, the Virtex XCV2000E offers pipelined multiplier, address decoder and block RAM.

Besides this additional resources inside the FPGA there are several additional components located on the Spyder-Board which can be used for the prototype. The PCI interface offers a fast connection to a PC. Two independent SRAM banks with 2MB each are connected to the FPGA and up to 4 Spyder boards can be combined via two VG96 extension headers using a backplane. Configuration of the board can be done via PCI or JTAG from a host PC or stand-alone with on-board EEPROM's.

3. Problems of ASIC IP core prototyping on FPGA's

The task of emulating a complex ASIC IP core on a low-cost FPGA platform is difficult in many ways. In this section we outline problems which arise during the integration process of IP cores on FPGA's. These problems can be classified into two domains. Firstly, the target technologies ASIC and FPGA differ significantly. Secondly, as mentioned above, FPGA's offer only limited resources. For example, the two million system gates of a XCV2000E FPGA from Xilinx can be used to prototype ASIC-IP cores of approximately 300.000 gates. The overall problem classification is shown in *figure 2*.

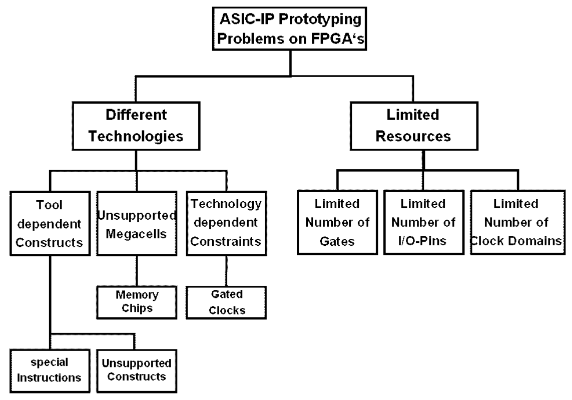
ASIC-IP Prototyping Problems on FPGA's
Different Technologies
Limited Resources
Tool dependent Constructs
Unsupported Megacells
Technology dependent Constraints
Limited Number of Gates
Limited Number of I/O-Pins
Limited Number of Clock Domains
Memory Chips
Gated Clocks
special Instructions
Unsupported Constructs

Figure 2. Problem classification for the ASIC-IP core integration on FPGA's.

3.1. Problems arising from differences in target technology

The problem of different technologies is not only present when a specific ASIC-IP component should be emulated on a FPGA. Every IP integrator is faced with this problem since the fabrication processes of integrated circuits can differ. For example, the ASIC-IP vendor could use a different synthesis tool supporting different language constructs. Considering VHDL the *generic* construct can be used to keep a design configurable. If this construct is not supported by the synthesis tool the IP integrator has to replace these generics by constants. Another problem are megacells like memories which must be transformed into the required target technology for the IP.

Technology dependent constraints are a specific problem in FPGA-based designs. For example, gated clocks are widely used in embedded systems design to reduce power consumption by disabling dedicated components not used during operation. However, in FPGA's gated clocks have to be transformed since only a limited number of clock domains are available. A common strategy to implement gated clocks in FPGA's is to use the clock enable line of the slice Flip Flops instead. The result is a tremendous intrusion into the design which is only possible with tool support [11].

3.2. Problems arising from limited FPGA resources

In the remaining part of this paper we will concentrate on issues regarding the second problem of ASIC-IP core integration. This problem results from the limited resources offered by FPGA's.

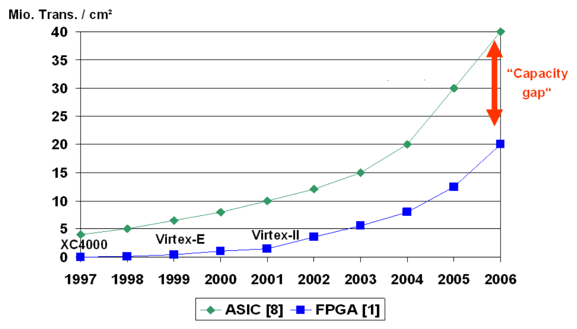

Figure 3. Comparison of the development of ASIC and FPGA gate capacities.

Figure 3 shows the estimated amount of gates offered by past and future ASIC and FPGA technologies. The FPGA gate count is transformed with the above mentioned formula of one to ten. Yet, there are three different solutions to overcome the resource problem of limited gates.

The first solution is straight forward. If an ASIC-IP core doesn't fit into a FPGA, use a bigger one. This solution has several drawbacks, and is only suitable in a few situations. Firstly, regarding *figure 3* there will always be an IP core or even a whole SoC design that doesn't fit into the biggest available FPGA. Secondly, using a bigger FPGA means using a different prototyping system which introduces additional costs to the IP integrator.

Another solution is the partitioning of the design into different FPGA's. This technique is used by industrial hardware emulation systems [6, 2]. Since on RTL level the connections between modules can consist of hundreds of signals a complex signal multiplexing infrastructure has to be established. This in conjunction with complex partitioning software is leading to the high prices of those emulation systems.

To overcome this disadvantage the partitioning of the design could be established in a different way where one part of the design could be emulated in the FPGA and the other part could be simulated on a workstation [7]. This solution has the advantage of being less expensive than the partitioning of the design into several FPGA's. The disadvantage of this approach is twofold. Firstly, the connection between simulator and emulator forms a bottleneck and secondly the simulator reduces the overall emulation speed of the whole system.

As a result, the problem of restricted resources in FPGA's could be considered as serious. Using more emulation hardware means fast emulation speed but higher costs in terms of hardware and software, whereas using simulation and emulation in parallel means lower costs but lower emulation speed too.

4. Resource aware rapid prototyping with FPGA's

To alleviate the resource problem we suggest a different approach of prototyping ASIC-IP cores on FPGA-based rapid prototyping platforms. The idea is based on the fact that FPGA's and prototyping systems offer a certain infrastructure to the IP integrator that should be used as efficent as possible. We will show this approach with the TriCore1 ASIC-IP microcontroller core and the Spyder rapid prototyping system introduced above.

Regarding *figure 1* it turned out that prototyping of the whole IP core is not possible in one XCV2000E FPGA. However, finding the largest possible working subset of the TriCore1 core is a extremely difficult task since the integrator has to provide a testbench with each subset of the core and has to try if the subset fits into the FPGA.

4.1. Finding an appropriate starting position

Therefore, we have chosen the opposite direction starting with a minimal subset of the core and adding more modules in later stages of the integration process. Considering the TriCore1 core it turns out, that the smallest reasonable subset is the CPU core itself since it enables a software engineer to compile and run programmes on the CPU. However, memories and the corresponding memory interfaces are needed for the programme code. Implementing memory in a FPGA consumes a lot of resources since one slice of a FPGA could only hold 2 bits of data.

New FPGA generations offer additional on-chip RAM to the designer. For example, the XCV2000E FPGA from Xilinx contains 160 *SelectRAM* blocks with 4096 bits each. The total amount of memory adds up to 82 KByte which can be enough space for small embedded applications. Thus, a possible solution for a restricted version of the TriCore1 IP core on Spyder is depicted in *figure 4(a)*.

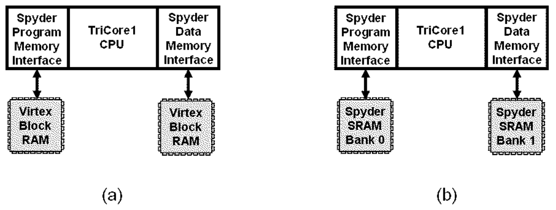
Spyder Program Memory Interface
TriCore1 CPU
Spyder Data Memory Interface
Virtex Block RAM
Virtex Block RAM
(a)
Spyder Program Memory Interface
TriCore1 CPU
Spyder Data Memory Interface
Spyder SRAM Bank 0
Spyder SRAM Bank 1
(b)

Figure 4. TriCore1 CPU module and RAM solutions.

Figure 4(b) shows an alternative solution for the TriCore1 integration on the Spyder-System. Using the Spyder SRAM's provided by the rapid prototyping board offers some advantages compared to solution (a). First of all, to emulate different programmes with solution (a) implies that the configuration bitstream of the FPGA has to be configured with the programme data. Therefore additional software is necessary to translate object code into a bit representation for the Virtex Block RAM's. The second disadvantage is the limitation of memory to 82KBytes. Thirdly, the routing is very complex when using all *SelectRAM blocks*.

On the other hand, the Spyder SRAM's offer 2MByte each. Therefore, in a first step the TriCore1 CPU was combined with glue-code for the Spyder SRAM's. The emulation of software with this prototype led to a significant speedup compared to RTL level simulation of the TriCore1 CPU. This solution had a few disadvantages but offered a good starting point for the enlargement of the TriCore1 IP core.

4.2. Enlargement of the small TriCore1

One of the disadvantages was the lack of an interface to connect additional modules to the TriCore1 CPU. To realize additional peripheral modules a processor bus is needed. Thus, solution (b) of *figure 4* was augmented by the original programme and data interfaces of the TriCore1 to have the local memory bus (LMB) present.

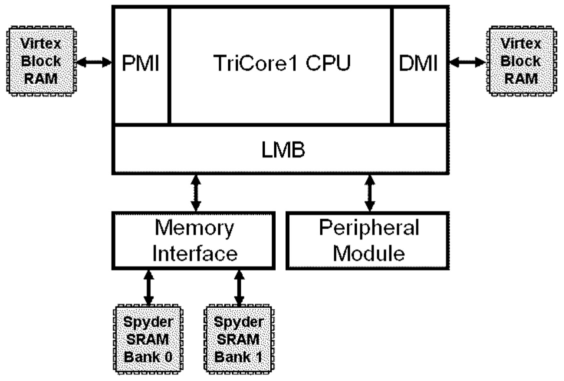

Figure 5. Augmented TriCore1 with LMB interface.

The resulting system design is shown in *figure 5*. This realization uses both types of memories available on the Spyder prototyping system. The internal Virtex block RAM's were used to emulate the on-chip Scratch Pad RAM of the TriCore1 core. The Spyder SRAM's emulate external memory connected via LMB.

Since in this implementation a processor bus is available additional peripheral modules can be connected to the TriCore1 realizing a real SoC prototype. Due to the restricted space left besides the TriCore1 these additional modules can only be small.

4.3. TriCore1 emulation flow on Spyder

Because of the limited environment of the emulated TriCore1 it is not possible to connect a software debugger to the system. Therefore the download and emulation of programmes must be established in an alternative way. The emulation flow is depicted in *figure 6*.

After compiling and linking of the application with some startup code to initialize the processor the *Spyder SRAM Generator* initializes VHDL models of the Spyder SRAM banks zero and one. with the programme to execute. These models can be integrated into the TriCore1 testbench we created for the restricted TriCore1 IP core. Simulation runtimes for a few benchmarks are listed in *table 1*.

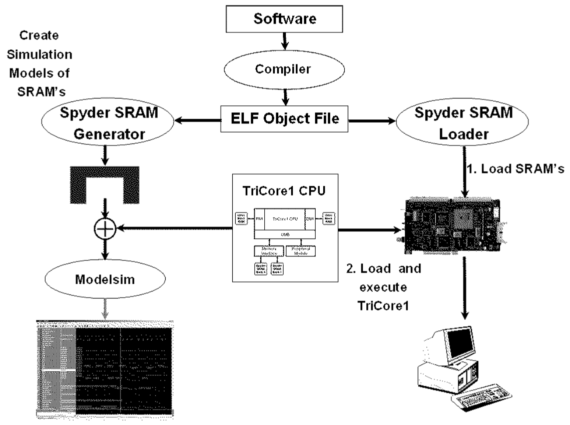

Figure 6. Integration and emulation flow for Spyder.

The emulation of the restricted TriCore1 IP core is done in two steps. In the first step the application object file is loaded by the *Spyder SRAM Loader* into the two SRAM banks of the Spyder board via PCI. In a second step the restricted TriCore1 IP core is loaded into the FPGA and the programme execution starts immediately after the download.

The result of the programme execution can be written to a reserved address located in the LMB address space. A communication module attached to the LM bus and sensitive to this address can read the result and can generate an interrupt at the PCI bus of the attached PC. There an interrupt service routine can determine the result and display it to the application engineer.

5. Results

The described prototyping method with usage of additional FPGA and prototyping board resources delivered a significant speedup of execution speed of programmes. *Table 1* shows a comparison of software runtime of RTL simulation on a workstation and emulation on the Spyder rapid prototyping system.

Benchmark one is Euclid's algorithm to calculate the greatest common divisor. The second benchmark determines the Fibonacci number for input value 10,000 and the third benchmark searches the first 1000 prim numbers using the sieve of Erastothenes algorithm. The measurements on Spyder were made with an emulation frequency of 8MHz which is the highest possible frequency when optimizing for area during the synthesis process.

Figure 7 shows a comparison of the number of FPGA slices needed for the whole TriCore1 IP and two prototypes of the microcontroller on the Spyder-Board. By ap-

	Euclid's Algorithm	Fibonacci Numbers	Sieve of Erastothenes
Number of instructions executed	1484	41419	20779
Simulation on workstation	approx. 28 Sec.	approx. 10 Min.	approx. 18 Min.
Emulation on Spyder	312 μs	3.9 ms	21.8 ms

Table 1. Software runtime comparison.

plying our proposed prototyping method of partially replacing parts of an ASIC design with resources of the prototyping system and augmenting the design in a stepwise manner the prototyping of an integral part of the microcontroller was possible.

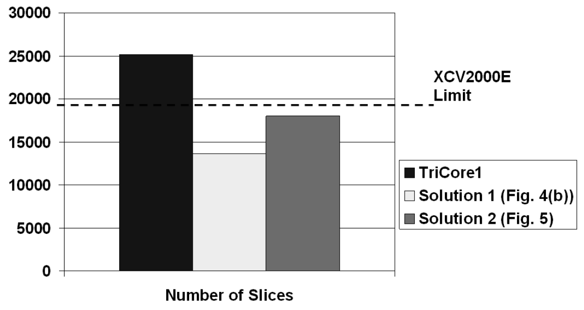

Figure 7. Synthesis results for the TriCore1 IP core on Spyder.

Two interesting numbers which approve resource aware rapid prototyping are given in *table 2*. The *total equivalent gate count* represents the number of gates a design would need as ASIC and is provided by the Xilinx *map* tool.

	Solution 1 (*Fig. 4(b)*)	Solution 2 (*Fig. 5*)
Total equivalent gate count	256,020	529,573

Table 2. Total equivalent gate count provided by the Xilinx *map* tool.

As mentioned in *Section 2* the two million system gates of a XCV2000E FPGA can be used to prototype ASIC-IP cores of approximately 300.000 gates. By using additional resources of the prototyping system this barrier could be exceeded. Solution two shown in *figure 5* uses block RAM's of the FPGA to emulate 8KByte of *Scratch Pad RAM* of the TriCore1 core. Since this additional gates are considered by the *map* tool too, there the total equivalent gate count is 529,573 which exceeds the theoretical value of 300,000 gates.

6. Conclusion

In this paper a rapid prototyping method was introduced which is based on the efficient usage of resources offered by the rapid prototyping system. This can help to reduce costs for the prototyping system in terms of hardware and software tools since one hardware board is enough and no additional partitioning and synthesis software is needed.

As an example the adaptation of the TriCore1 microcontroller ASIC-IP core to the rapid prototyping system Spyder was shown. There both the Virtex block RAM's and Spyder SRAM banks where used to emulate the TriCore1 memory infrastructure. With the restricted model the execution of programmes is possible leading to a significant speedup of runtime compared to simulation.

References

[1] P. Alfke. Die Fpga-Evolution. *Elektronik Praxis*, (1), January 2002.

[2] Aptix. *Prototype Studio: RTL to PSP, pre-silicon prototypes for System-on-Chip designs*. http://www.aptix.com.

[3] FPGA-FAQ. *FGPA prototype boards*. http://www.fpga-faq.com/FPGA_Boards.shtml.

[4] Infi neon. *32-Bit microcontroller TriCore1*. http://www.infi neon.com/cgi/ecrm/scripts/prodcat.jsp?oid=8138.

[5] K. Khalilian, S. Brain, R. Tuck, and G. Farrall. Reusable verifi cation infrastructure for a processor platform to deliver fast SoC development. *Proceedings of the International Workshop on IP-based SoC Design*, 2002.

[6] Quickturn. *Verifi cation by simulation acceleration and emulation*. http://www.quickturn.com/products/palladium.htm.

[7] S. B. Sarmadi, S. G. Miremadi, G. Asadi, and A. R. Ejlali. Fast prototyping with co-operation of simulation and emulation. *Proceedings of the 12th International Conference on Field Programmable Logic and Application (FPL 2002)*, LNCS 2438:15–25, 2002.

[8] SIA. *SIA roadmap for ASIC gate development*. http://www.sia.com.

[9] Synopsys. *DesignWare StarIP cores*. http://www.synopsys.com/products/designware/designware.html.

[10] Synopsys. *Vera*. http://www.synopsys.com/products/vera/vera.html.

[11] Synplicity. *Ceritify ASIC prototyping solution*. http://www.synplicity.com/products/certify/index.html.

[12] X2E. *The Spyder rapid prototyping system*. http://www.x2e.de.