An SoC Design Methodology Using FPGAs and Embedded Microprocessors

Nobuyuki Ohba
ooba@jp.ibm.com

Kohji Takano
chano@jp.ibm.com

IBM Research, Tokyo Research Laboratory, IBM Japan Ltd.
1623-14 Shimotsuruma, Yamato city, Kanagawa, Japan

ABSTRACT

In System on Chip (SoC) design, growing design complexity has forced designers to start designs at higher abstraction levels. This paper proposes an SoC design methodology that makes full use of FPGA capabilities. Design modules in different abstraction levels are all combined and run together in an FPGA prototyping system that fully emulates the target SoC. The higher abstraction level design modules run on microprocessors embedded in the FPGAs, while lower-level synthesizable RTL design modules are directly mapped onto FPGA reconfigurable cells. We made a hardware wrapper that gets the embedded microprocessors to interface with the fully synthesized modules through IBM CoreConnect buses. Using this methodology, we developed an image processor SoC with cryptographic functions, and we verified the design by running real firmware and application programs. For the designs that are too large to be fit into an FPGA, dynamic reconfiguration method is used.

Categories and Subject Descriptors

B.6.3 [Logic Design]: Design Aids – Verification.

General Terms

Design, Verification.

Keywords

SoC, ASIC, FPGA prototyping, and mixed-level verification.

1. INTRODUCTION

Thanks to progress in the silicon technologies, many more gates can be integrated into System On Chip (SoC), which increases performance and reliability while reducing overall system costs. However, as design complexity increases, well-organized consistent design and effective verification have become indispensable for reducing the time to launch the products into the market.

For SoC design, a top-down design approach is commonly used to maintain design consistency from the system level to the net-list level. The target design is first modeled at the top abstraction level, called the specification level, where the designers describe what the chip does. The top design is gradually broken down from the application level to the register transfer level in order to make it easier for the designers to check the equivalence of the designs in adjacent levels, that is, transaction level, cycle accurate level, and register transfer level, as shown in Table 1.

Table 1. Abstraction levels

Abstraction level	Definition	Timing	Implementation
Specification level	Concept right	Untimed	Software
Transaction level	Architecture right	Timed /untimed	Software
Cycle accurate level	Micro-architecture right	Timed	Software
Register transfer level	Implementation right	Timed	Hardware

To complete the design and verification more quickly, system-level design and hardware/software co-verification are used. FPGA-based systems have become popular in co-verification and rapid prototyping [1] [2] [3] [4]. Mapping the entire design of the target SoC into an FPGA gives an accurate and fast representation, but it is not always an easy job because:

- The target SoC may consist of two or more functional modules that are written at different abstraction levels in the course of different development phases. Only synthesizable modules can be mapped into an FPGA and run for debugging. In conventional methods, modules written at the system or behavior levels should be run on a host workstation and be interfaced to the hardware prototyping system. Overhead of the data exchange between the host workstation and the hardware prototype often limits the verification speed.
- An Instruction Set Simulator (ISS) is important for making the debugging operation effective. In many cases, however, it is still too slow to run real applications.
- Very large designs cannot be fit into FPGAs.

This paper presents a design methodology that addresses these problems, and shows the development procedure for an image processing SoC as an example.

DAC 2004, June 7–11, 2004, San Diego, California, USA
Copyright 2004 ACM 1-58113-828-8/04/0006…$5.00.

2. DESIGN METHODOLOGY

2.1 Hybrid-Level Design and Verification

For SoC development, we make full use of the FPGA capabilities for the design and verification processes. Modules designed at different abstraction levels are implemented into FPGAs, interfaced with each other, and run together to emulate the target SoC. For our designs to date, we have used the Xilinx Virtex-II Pro FPGA [5], which has multiple PowerPC hardware cores. The modules designed at higher levels are run on PowerPC cores, and those at lower levels, such as the register transfer and netlist levels, are directly mapped to the FPGA cells. To seamlessly interconnect the modules from different levels, we have made a hardware wrapper for the PowerPC cores.

The design work of the processor was carried out in a fully top-down manner. Here we describe the procedure step by step

2.2 Specification Level Design

The target SoC is first described at the specification level. The design is created and verified by using a host PC/workstation, and then transferred to the FPGA prototyping system so that it can be run on the PowerPC hardware cores embedded in the FPGAs, as shown in Figure 1.

However, it is not always possible to make the design at the specification level run on the embedded PowerPCs, because the entire design might be too large or complex to fit into the embedded PowerPCs. In addition, some designs may need specific libraries to be linked. For this reason, the full design transfer to the embedded FPGA for this level is optional.

2.3 Transaction Level Design

Modules are then described at the transaction level, and simulated on the PowerPC hardware cores in the FPGA. Figure 2 is an example how the transaction-level modules are mapped into the FPGA, which is assumed to have three PowerPC cores.

PowerPC A works as the microcontroller of the SoC. The transaction-level design modules are simulated by PowerPCs B and C, which are attached to the system bus through hardware wrappers. Two modules at the register transfer level are attached to the system bus. The hardware wrapper executes a request-acknowledge type of handshake with the other modules via the systems bus. It also communicates with the associated PowerPC through some hardware registers implemented in the hardware wrapper. For example, when the hardware wrapper receives a read request from another module, it generates an interrupt for the PowerPC. The PowerPC takes the appropriate action and stores the requested data in a register in the hardware wrapper. The hardware wrapper then returns the acknowledge signal with the read data to the requester.

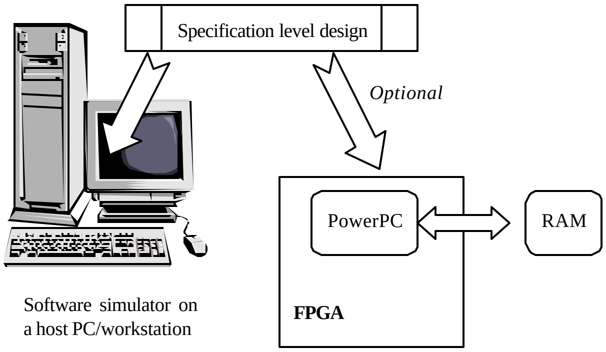
Specification level design
Optional
PowerPC
RAM
Software simulator on a host PC/workstation
FPGA

Figure 1. Specification level design

Each PowerPC core uses dedicated memory to store the instruction code and data. In this example, PowerPCs A and B uses large memory devices outside of the FPGA, while PowerPC C uses the FPGA's faster embedded memory.

PowerPC cores are controlled by using the RISCWatch debugger via the IEEE 1149.1 (JTAG) interface [6]. The source-level debugger and processor-control features provide designers with the tools needed to develop and debug hardware and software quickly and efficiently. All of the registers in the hardware wrapper are accessible to the RISCWatch debugger, so that the engineer can easily trace and debug the design.

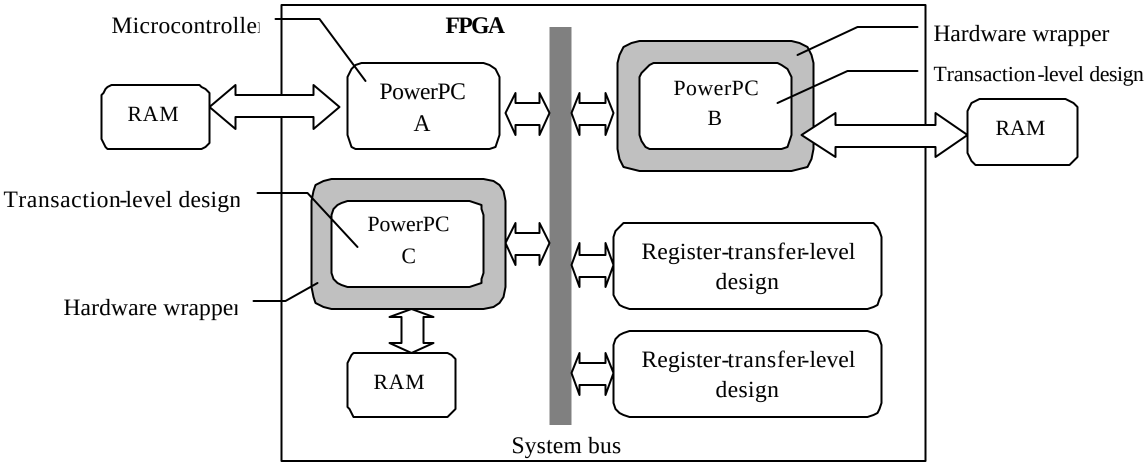

Figure 2. Transaction-level design modules coexisting with register-transfer-level design modules

2.4 Cycle Accurate Level Design

To run cycle-accurate-level modules on the PowerPC hardware core, we must make them keep pace with other modules written at different levels. When each PowerPC completes the simulation job for a cycle, it asserts the cycle complete signal to the clock pacer, as shown in Figure 3. The clock pacer checks if all of the jobs that must be executed in the cycle are done, and asserts the step forward signal to all of the modules. After the modules receive the step forward signal, they start the job in the next cycle.

In our current implementation, the PowerPC cores run at 280 MHz, and therefore the simulation runs of the cycle-accurate-level design modules are not as fast as on a cutting-edge microprocessor, which could run at 3 GHz or faster. However, we still find this approach useful for the SoC design, because: 1) the PowerPC is tightly coupled with the other modules within the FPGA and thus the overhead of the handshake is minimal, and 2) it gives a smooth transition from the transaction-level design to the register-transfer-level design.

2.5 Register Transfer Level Design

Register-transfer-level design modules are fully synthesized and mapped into the FPGA. Figure 4 is an example of a chip-level design, which has a microcontroller and four modules at the register transfer level. It works as a full prototype chip of the target SoC.

2.6 Advantages

In addition to the verification speed, the proposed methodology has the following advantages:

- Design modules from different levels can be combined and run together on the prototyping system.
- The design work can be effectively shared by two or more designers, since the interfaces between modules are clearly defined and the combination of components at multiple levels can be executed at a time.
- The development and test of the software can be started at the system integration level even in the phase where register-transfer-level designs are not available for all the components. Using the proposed mixed-level verification method, the system level verification can be started earlier, and thus the number of bugs remaining after the sign-off will be decreased.

3. DESIGN EXAMPLE

3.1 Image Processor with Cryptographic Functions

This section shows the development procedure using an example. The target SoC is an image processor with cryptographic functions. Figure 5 is the block diagram of the SoC. The microcontroller is an IBM PowerPC 405, which is interconnected with a memory interface, an MPEG encoder/decoder, and a DES/AES cryptographic engine through the Processor Local Bus (PLB). The PLB is connected to the On-chip Peripheral Bus (OPB) via the PLB/OPB bridge. An RSA/ECC cryptographic engine, random number generator, and general-purpose I/O (GPIO) are connected to the OPB.

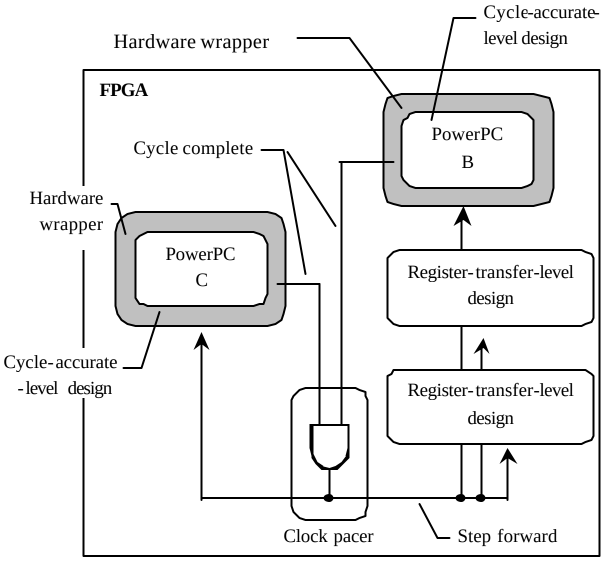
Hardware wrapper
Cycle-accurate-level design
FPGA
Cycle complete
PowerPC B
Hardware wrapper
PowerPC C
Register-transfer-level design
Cycle-accurate-level design
Register-transfer-level design
Clock pacer
Step forward

Figure 3. Cycle-accurate-level design modules

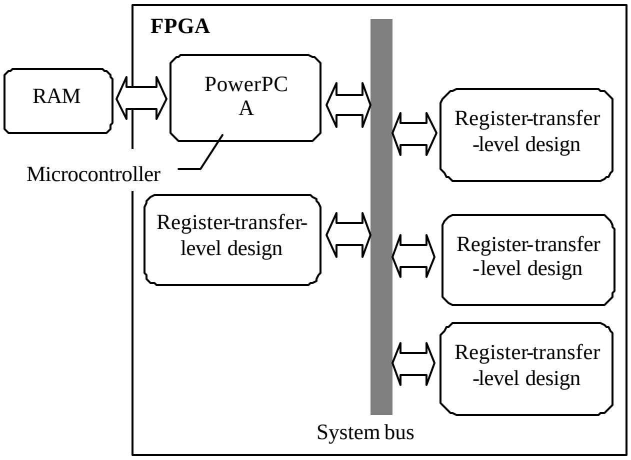
FPGA
RAM
PowerPC A
Microcontroller
Register-transfer-level design
Register-transfer-level design
Register-transfer-level design
Register-transfer-level design
System bus

Figure 4. PowerPC and register transfer level design modules

3.2 Design Procedure

We used a top-down design approach, in which the target design is gradually broken down from the specification level to the register-transfer level, makes it easier for the designers to check the equivalence of the designs in adjacent levels. We used C++ and VHDL for modeling the processor to maintain the design consistency in the top-down design approach.

The development process proceeds through four levels, as shown in Figure 6.

- Specification level: The target SoC is an image processor with cryptographic functions. The image processing is based on the MPEG4 standard. The cryptography is used for Secure

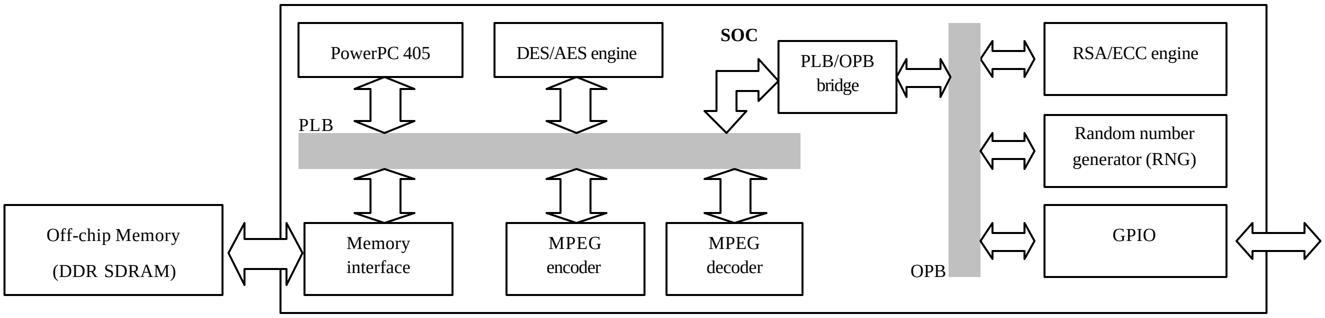

Figure 5. Block diagram of the image processor with cryptographic functions

Socket Layer protocol (SSL) and data encryption. RSA/ECC and DES/AES are the core functions for SSL. The functions are written in C++.

- Transaction level: We looked through the C++ source code of the applications, and analyzed where and how the image and cryptography operations should be implemented in the hardware engines. From the functional perspective, we decomposed each cryptographic function into hardware basic blocks, such as a controller (sequencer), arithmetic calculator, and memory.
- Cycle accurate level: All the basic entities are synchronized with the global clock. It should be noted that the models at the above levels are event-driven; the models at this level, on the other hand, are clock-driven. The modules are further broken down to basic entities. The controller is implemented with registers, decoders, finite state machines (FSM), and counters. The arithmetic operators involve adders, multipliers, selectors, and registers. The memory consists of conventional memory devices and possibly high-speed caches. The conditional branches, such as ‘if’ and ‘loop’ statements, in the C++ code are translated into the equivalent code that can be straightforwardly implemented on hardware.

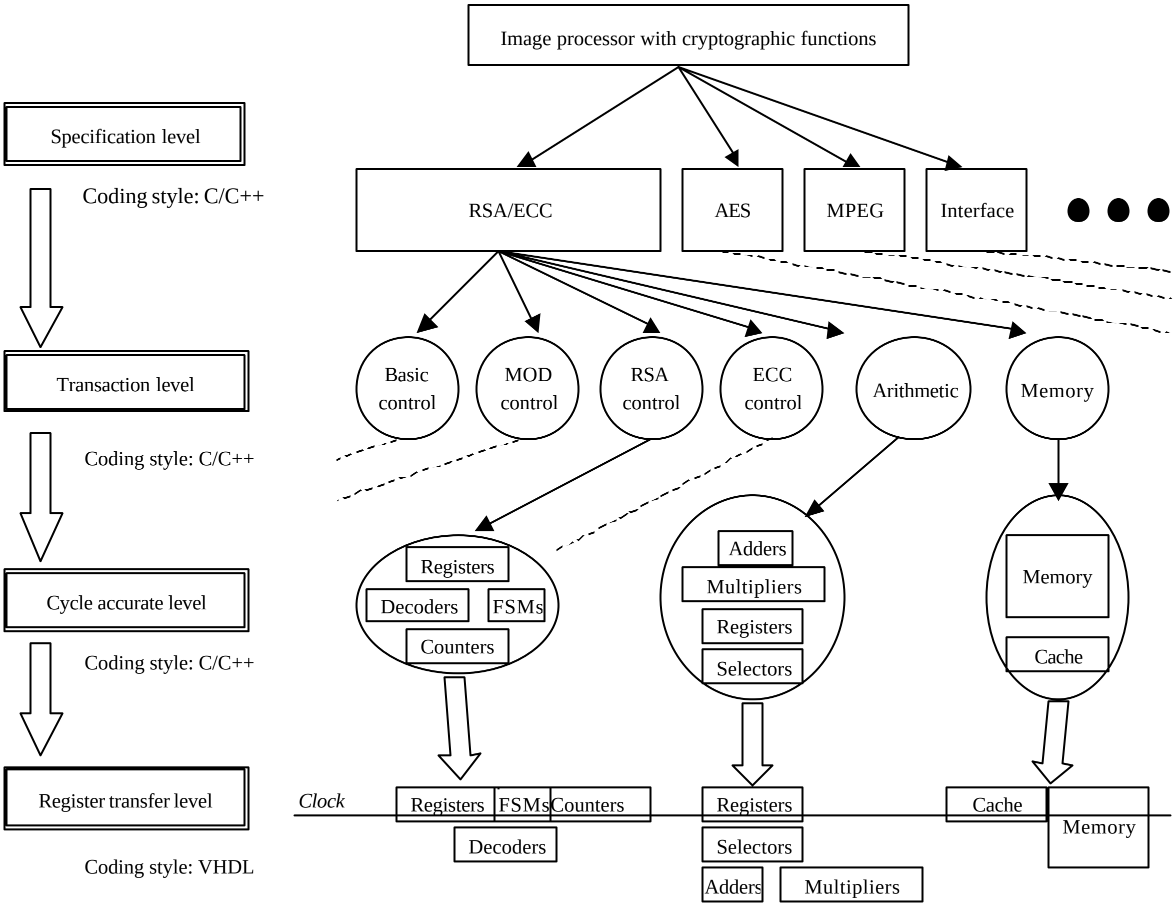

Figure 6. Design levels

- Register transfer level: The C++ design is translated into HDL. We wrote a simple Perl script that translates an FSM in C++ to the equivalent VHDL code. A program for fully automatic translation from C++ to VHDL is under development. The design of this level is fully synthesizable, and therefore can be directly implemented in an FPGA.

The most challenging part of the development was the logic verification, because:

- Public key cryptography (e.g., RSA and ECC) requires a large number of clock cycles to complete one calculation. A 2,048-bit RSA calculation, for example, takes more than 100,000,000 clock cycles. It takes substantial time to verify these operations with software simulators. In fact, a software simulation of this RSA computation requires several days to complete even a single test case. In addition, to make the intermediate state of the signals traceable after the simulation, the signal state at each clock cycle must be recorded in storage, such as a hard disk. The designer must provide substantial amounts of disk space, or must reduce the number of signals to be traced. In addition, most commercial EDA simulation tools allow only one simulation to run per license. During a long simulation, the designer cannot run other simulations and this makes the design work inefficient.

- In the gate-level design, we would like to test the correctness of the circuit. The term "correctness" here means that the circuit gives correct answers for all the possible inputs AES and RSA/ECC use operations on Galois Fields, and those operations require complex hardware circuits.

- A practical secure operation is composed of several cryptographic calculations. An SSL session, for example, contains several public-key and common-key encryptions and decryptions. It is conceivable that many variations of the combinations of cryptographic operations could be used, and they could require billions of clock cycles to complete.

- The cryptographic operations should be encapsulated within the chip for two reasons: (1) to reduce the load on the host microprocessor, and (2) to keep the secret information inside the chip. The interaction between the control software and the cryptographic processor should be kept as limited as possible.

The basic algorithms implemented in the core are modular exponentiation for the RSA operation and EC scalar multiplication over GF(p) for the ECC operation. The RSA and ECC operations were modeled, designed, and verified using C++. However, it is difficult to make the RTL design directly from the specification-level C++ model, because the C++ model is composed of many complex control sequences and data manipulations. Therefore, the design methodology presented in the previous section was used to create the RTL design from the cycle-accurate-level C++ model.

All of the data input and output operations for the classes are modeled as method calls, and the data values are stored in private variables. At the register transfer level, these method calls are translated into signal entity descriptions of HDL, and private variables are mapped to register descriptions. Each class has only one method that executes all of the operations specified in the class.

Table 2. Number of cycles, software simulation time, and FPGA emulation time for RSA calculations

Operation	Number of cycles	Software simulation	FPGA emulation
1,024-bit RSA with a 32-bit core	4 million	6 to 7 hours	0.3 seconds
2,048-bit RSA with a 32-bit core	32 million	2 to 3 days	3 seconds
1,024-bit RSA with a 16-bit core	16 million	1 day	1.5 seconds
2,048-bit RSA with a 16-bit core	102 million	1 week	10 seconds

3.3 Verification

To boost the speed of the verification, we implemented the RSA/ECC core in the FPGA. Table 2 shows the number of clock cycles, the software simulation time, and the FPGA emulation time. The software simulations were performed with Model Technology ModelSim SE version 5.4 running on IBM Intellistation MPRO (2 GHz Pentium IV, Windows 2000 Professional). The FPGA emulations were done with the Xilinx FPGA XC2VP20 running at 33 MHz.

The FPGA emulations are about 100,000 times faster than the software simulations, and thus make the debugging very efficient.

Although the FPGA prototyping system offers powerful design aids, the user needs to pay attention to the following points:

- The setup and hold timings of the FPGA are obviously different from those of a real ASIC. Intensive timing simulations and actual chip tests must be incorporated.

- The FPGA behavior does not always turn out to be identical to the SoC. For example, technology-specific macros, such as PLL, tend to behave differently in an FPGA and an SoC.

3.4 Dynamic Reconfiguration

Since the FPGA has a limited capacity, the target design modules that can be fit into a single FPGA are limited. In fact, we could not fit all of the modules shown in Figure 5 into one FPGA. For this reason, we had to verify each module in a different phase.

We use a new debugging scheme, called dynamic FPGA reconfiguration, which configures and reconfigures the FPGA on-the-fly during the emulation. Figure 7 shows an example of the scheme. The FPGA board is installed in a PCI slot of the host PC, and controlled by a debugging program running on the host PC. The debugging control program has the XVF player [7], with which the main control program configures the FPGA through the configuration cable attached to the parallel port of the host PC.

The verification procedure is:

1. The debugging control program downloads the RNG (random number generator) core to FPGA.
2. The debugging control program starts the RNG function. The FPGA generates a random number, stores it in memory, and sets a flag in the PCI interface.
3. After seeing the flag set by the FPGA, the debugging program downloads the DES core to the FPGA.
4. The FPGA loads the random number to the DES core. In this example, DES is used for smoothing the numbers and generating a secret key. The FPGA stored the key in memory, and sets the flag in the PCI interface.
5. After seeing the flag set by the FPGA, the debugging program downloads the RSA core to the FPGA.
6. The FPGA encrypts the data with the key stored in memory.

The reconfiguration overhead of this procedure is acceptable, as it takes less than a minute to reconfigure the FPGA.

This technique is effective not only for the verification of designs that cannot be fit into an FPGA but also for reducing the cell utilization of the FPGA. During the generation of the FPGA configuration data, the FPGA design tool performs a “fitting” that maps the netlist onto the cells. The fitting phase takes substantial time if the design requires over 90% of the cells of the FPGA. It often takes several hours to accomplish one such “fitting.” If the design can be decomposed into smaller parts and each part can be individually fit into the FPGA, the time for fitting is reduced. Remember that in many cases just a few lines of the source code are rewritten to correct a bug. It is not a good idea to do synthesis and fitting over and over every time when only a few source lines are rewritten. By decomposing the target design into smaller modules, the time for the synthesis and fitting is greatly shortened. Using the dynamic FPGA reconfiguration technique, the decomposed parts are sequentially downloaded to the FPGA during the emulation. This eventually improves the verification efficiency.

4. CLOSING REMARKS

In this paper, we presented a design methodology that makes full use of the FPGA and its embedded microprocessors. We used a top-down design approach, in which the software models at the specification level were decomposed into submodels at lower levels until the register transfer level code was generated. Cryptographic functions use complex operations and it was not easy for us to make a bug-free circuit. RSA and ECC calculations require a large number of clock cycles, and it would take a week to complete the simulation of a single RSA computation. It would take even more time to simulate the whole processor design including the external interfaces. To make the design solid, we had to run as many test cases as possible. To overcome these difficulties, we used the FPGA intensively, with runnig the software and hardware models at the same time.

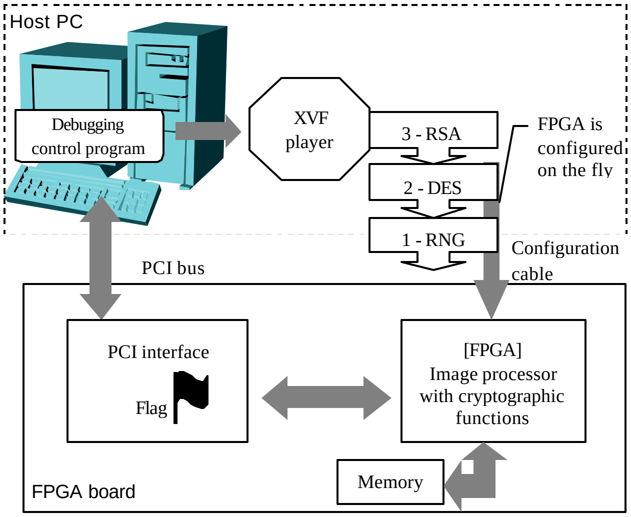
Host PC
Debugging control program
XVF player
3 - RSA
2 - DES
1 - RNG
FPGA is configured on the fly
Configuration cable
PCI bus
PCI interface
Flag
[FPGA] Image processor with cryptographic functions
Memory
FPGA board

Figure 7. Example of dynamic reconfiguration

In conclusion, the key of the successful design was the collaboration of software simulation and hardware emulation. We used the design methodology presented in the previous sections, and, as a result, the design work was shortened to half of the planned time.

5. REFERENCES

[1] Gschwind, M., Salapura, V. and Maurer, D.: "FPGA prototyping of a RISC processor core for embedded applications," IEEE Transactions on Very Large Scale Integration (VLSI) Systems, Volume: 9 Issue: 2, pp. 241-250, April 2001.

[2] J. Gateley, M. Blatt, D. Chen, S. Cooke, P. Desai, M. Doreswamy, M. Elgood, G. Feierbach, T. Goldsbury, D. Greenley, R. Joshi, M. Khosraviani, R. Kwong, M. Motwani, C. Narasimhaiah, S. J. Nicolino Jr., T. Ozeki, G. Peterson, C. Salzmann, N. Shayesteh, J. Whitman, and P. Wong, "UltraSPARC-I emulation," *Proceedings of 32nd Design Automation Conf.* San Francisco, CA: IEEE, June 1995.

[3] Roesler, E. and Nelson, B., "Debug Methods for Hybrid CPU/FPGA Systems," *Proceedings of 2002 IEEE International Conference on Field Programmable Technology (FPT)* Hong Kong, China, pp. 16-18, December 2002.

[4] Pogodalla, F., Hersemeule, R., Coulomb, P., "Fast prototyping: a system design flow for fast design, prototyping and efficient IP reuse," *Proceedings of the Seventh International Workshop on Hardware/Software Codesign, 1999 (CODES '99)*, pp. 69 –73, May 1999.

[5] Xilinx, Inc., "Virtex-II Pro Platform FPGA Data Sheet," January 2003.

[6] IBM, "RISCWatch Debugger for PowerPC Processors," Product brief, April 1996.

[7] Xilinx, Inc., "Xilinx In-System Programming Using an Embedded Microcontroller," June 1999