A Secure Self-Reconfiguring Architecture based on Open-Source Hardware

Javier Castillo, Pablo Huerta, Victor López, José Ignacio Martínez
Universidad Rey Juan Carlos, Móstoles, Spain
{javier.castillo, pablo.huerta,victor.lopez,joseignacio.martinez}@urjc.es

Abstract

With the new and powerful Field Programmable Gate Array (FPGA) families, new possibilities have been opened. One of these features is the possibility of reconfiguring a section of the FPGA while the rest is working. Moreover, this fixed part could be responsible for reprogramming the reconfigurable part, either because a change in functionality is required or because a new version of the hardware needs to be implemented. This paper shows how an FPGA system based on an Open Source OpenRISC 1200 microprocessor takes advantage of this feature to perform the Secure Download of the firmware and the hardware needed to run an application. In this particular case a Reed-Solomon Encoder and a Cryptographic application were used to demonstrate the viability of the scheme.

1. Introduction

The incredible growth of FPGA capabilities in recent years and the new features included on them has opened many new investigation fields. One of the more interesting ones concerns Partial Reconfiguration and its possibilities. This feature allows the device to be partially reconfigured while the rest of the device continues its normal operation.

In a system without Partial Reconfiguration capabilities the responsibility of reprogramming the FPGA with a new configuration is taken by an external device, usually a microprocessor[1]. This device downloads the bitstream which contains the new configuration inside the programmable device through its configuration pins. It is not unusual that the new bitstream has to be transmitted to the board through a communication channel[2] (Internet, RS232, PCI, Wireless,…). This transmission also requires external hardware controllers that have to be physically connected on the board.

With the capacity of the new FPGA families (millions of equivalent gates) it is not difficult to imagine a new scenario where all the external logic is implemented inside the FPGA, including the microprocessor and the hardware controllers, saving area and money.

Using this approach, the reconfigurable area can be seen as an FPGA inside the FPGA, where the fixed part is responsible for controlling and securing the reconfigurable one and reprograms it when it is required [3].

This paper proposes an architecture based on open source cores in order to develop a functional auto-reconfiguring platform. The viability of Open-Source Cores has been discussed in previous works[4][5][6]. The main element of the architecture is a soft-core processor (OpenRISC 1200) which takes a bitstream from the communication channel, a remote TFTP server in this case, and partially reprograms the FPGA in a secure way.

In chapter 2 we will describe the overall system architecture that allows the partial reconfiguration. Chapter 3 exposes the Xilinx Partial Reconfiguration Flow that generates the partial bitstreams needed to partially reprogram the system. This chapter also describes the ICAP port which is physically responsible for reprogramming the FPGA. Chapter 4 shows the system implementation describing all the parts that compose the system. In Chapter 5 the viability of this scheme will be demonstrated by applying it to two examples, a Reed-Solomon Encoder and a Cryptographic application. Finally, the conclusions,
results, and future work are presented.

2. System Architecture

The proposed architecture is composed of the elements shown in Figure 1. The most significant element is the microprocessor. In this particular case an open-source OR1200 core was selected to be the heart of the system.

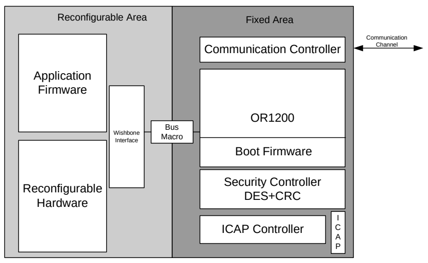

Figure 1. System Architecture

The soft-core is responsible for controlling the communication channel, taking the data from it, decrypting and checking the bitstream using the DES and CRC cores, and reprogramming the reconfigurable area using the ICAP controller. The configuration data is transmitted to the system through the communication channel, this channel could be for example, Ethernet, RS232, 802.11, Bluetooth, …

The partial bitstream contains not only the custom hardware, it also has block rams configured as ROM with the firmware needed to run the application.

When the reconfigurable part is reprogrammed the OR1200 jumps to the new firmware and the execution of the application begins.

There are two ways to exit from the current application. The first one is when the application runs only during a limited time or a limited number of iterations and then returns the control to the Reconfiguration Manager. The second one is the most common and is when the OR1200 receives an interruption from the communication controller with a “stop” command. When this happens, the OR1200 jumps again to the subroutine that controls the reconfiguration process. Then you can download a new partial bitstream or restart the application returning from the exception to the point where the software was before.

3. Self-Reconfiguration

The proposed architecture bases its functionality on the possibility of reconfiguring one part of the device while the other part is working. The fixed part is responsible for reprogramming the reconfigurable part using the ICAP port included by Xilinx in some of their FPGA families. To perform this task some considerations must be followed when designing the system.

3.2. Partial Reconfiguration Flow

In order to develop a functional system with self-reconfiguration capability, Partial Reconfiguration Flow must be followed as described in [7]. This flow allows the generation of partial bitstreams which contains configuration information regarding an FPGA area while the rest is not affected during the programming.

The first step is to create two modules including all the elements which compose the design. One module is the fixed part containing the soft-core processor with the boot firmware, the peripheral controllers, the ICAP port controller and the Security Controller. In the other module the application with its associated firmware is included.

Because two applications are going to be tested, two different modules have to be developed, one containing the Reed-Solomon application and another containing the Cryptographic Application. Both modules must have the same ports in order to keep the interface between the fixed and the reconfigurable part during the partial reconfiguration.

Once the two modules are designed and the interface between them is well defined, the top files are written. The top files only contain modules instantiated as black boxes, that means, only their interface is declared. The connections between these modules have to be done using bus macro elements. The bus macro is a critical element in the Partial Reconfiguration Flow. The bus macro is just a hard macro locked in a fixed position. When the new configuration is loaded, the hard macro keeps the connections between the fixed part and the new reconfigurable module.

Two top files have to be developed, one for each user application. Both have to be equal, the only difference is the user application instantiation. In each different top file the associated user application must be instantiated.

The next step consists of writing the UCF constraint file. In this file the location of the external pins, the positions of the bus macros , and the area constraints must be specified.

The system description files and the constraints file are the input to the Modular Design Flow [8].

The Modular Design Flow is composed of three phases:

- Initial Budgeting Phase
- Active Module Implementation
- Final Assembly

During the Initial Budgeting Phase the positioning of all modules and I/O pins is done. The result is a NGD file with all modules instantiated as unexpanded blocks.

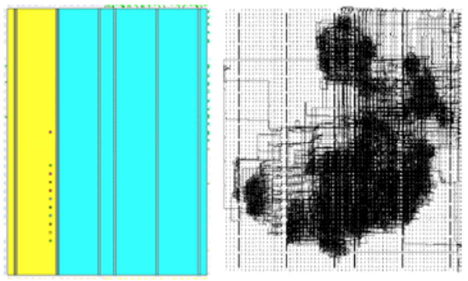

Figure 2. On the left side you can see the area constraints. On the right side, the design after the place and route process. The way in which the area boundary is only crossed by the bus macro primitives can be seen.

In the second phase each module that composes the system is mapped and routed inside the area bounds described in the constraints file. The result is a partial bitstream which can be used to partially reprogram the system. The module implementation is also published in the pims directory to be an input in the final assembly phase.

The final step is the final assembly phase where the top file with the unexpanded blocks and the implementation of the modules, resulting from the second phase, are combined to get an initial bitstream.

After the process, an initial bitstream for each top file, and a partial bitstream for each reconfigurable module are obtained.

These partial bitstreams will be transmitted to the system and used for self reconfiguration through the ICAP Interface.

3.2. ICAP

The fixed part of the FPGA needs a mechanism to reprogram the reconfigurable part. This mechanism is provided by Xilinx in some of their FPGA families and is called “Internal Configuration Access Port” (ICAP). The ICAP interface is a subset of the SelectMap interface, and it allows the internal logic to access the configuration data of the FPGA.

The ICAP interface is located in the lower right hand corner of the FPGA, so this introduces a restriction to our system. The fixed part responsible for reprogramming the FPGA must be located on the right hand side.

The ICAP can accept data up to 50Mhz without handshaking protocol, but by controlling the CCLK input data can be downloaded at lower rates. The ICAP port only accepts partial bitstreams because it cannot stop the FPGA during the reconfiguration process.

4. System Implementation

The system was implemented over a Celoxica RC203 development board with a Virtex-II X2CV3000FG676 FPGA.

The fixed part was implemented as explained before in the right hand side of the FPGA. This part uses an Ethernet link as a communication channel to download the bitstreams from the host PC. The Security Controller is made up of a DES cryptographic core, and a CRC calculator. The OR1200 soft-core takes the data from the communication controller and sends it to the Security Controller to decrypt and check the data. Then the data is applied to the ICAP through the ICAP controller.

To demonstrate the viability of the scheme two applications are shown: one based on a Reed-Solomon Encoder, and the other on a data encryption and hashing accelerator composed by an AES core of 192 bits key length and a MD5 hash algorithm.

4.1. OpenRisc 1200

The chosen microprocessor was an OpenRISC1200 core. This soft-core is freely distributed under an LGPL license at OpenCores website [9]. The OR1200 [10] is a 32-bit scalar RISC with Harvard architecture with a 5-stage integer pipeline intended for embedded, portable and networking applications.

One of its main characteristics is its configurability. Using a configuration file you can add or remove more than ten optional units as data and instruction caches, memory management unit (MMU), power management unit, and many others.

The basic communication channel of the platform is an OpenCores Wishbone Compatible Bus [11]. It has synchronous data and address buses with multiple masters and slaves. An arbiter decides in each moment which master takes the control of the bus.

OR1200 includes a complete SDK based on GNU tools with a GCC compiler, Binutils containing linker and assembler and GDB for debugging purposes. Many operating systems have been ported to OpenRisc Architecture, eCos, uClinux, Linux, RTEMS, microC/OS-II. Also different C libraries have been ported to OpenRISC architecture like uClibc or newlib.

Even though uClinux and eCos over OR1200 have run successfully, for this work no operating system was used, since our embedded applications do not require it. ANSI C code and newlib to support basic C libraries, as well as assembler code to start the execution of the program were used.

4.2. Security Controller

The configuration data transferred from a remote source contains valuable information which has to be secured. The data has to be secured in three ways. First the data has to be encrypted to protect the data from unauthorized readers. If the bitstream is not encrypted, it can be copied or reverse engineered. Second, the data has to be protected from communication errors or modifications during the transmission, this could be achieved by using a CRC and optionally with a digest algorithm. Third, the data source has to be authenticated to ensure that the source of the data is valid. This can be done using certificates based on public key cryptography.

The implemented Security Controller performs the first two operations: it decrypts the data using a 64 bit DES [12] algorithm and checks its integrity using a CRC. The selected CRC method was a 16-bit CRC-CCITT specification. This CRC calculation is applied to each block of 64 bits, obtaining data packets of 80 bits.

These packets are sent to the FPGA through the communication channel and checked and decrypted by the Security Controller. If an error is detected, the process is aborted and the initial configuration is loaded to return to the initial state.

4.3. ICAP Controller

The ICAP port is the element provided by Xilinx that allows the Self- Reconfiguration in Xilinx devices.

In order to manage the ICAP controller from the OR1200 microprocessor, a controller connected to the system bus has been developed. This controller has two registers mapped into the memory of the microprocessor, a data register where the data to be written/read to the ICAP port is contained, and a control register to indicate when the transference has to begin or when it has finished.

5. Application Test

To test the system two applications were run. One is based on a Reed-Solomon Encoder core downloaded from OpenCores. The second one is a Cryptographic Application based on an AES [13] algorithm of 192 bits key length and an MD5 [14] digest algorithm.

At the beginning the FPGA is configured with initial configuration data containing the Reed-Solomon Application. This configuration data is stored in a Smart Media Flash Memory. At the power up, a CPLD reads the initial configuration from the Flash and applies it to the FPGA performing a full reconfiguration.

When the system starts up, a menu is displayed through the RS232 interface.

The menu has three different options. The first one is to run the user application stored in the reconfigurable area. The second is to reconfigure the FPGA with a new partial bitstream. The last option is to download from a remote TFTP server a new program in the external ZBT SRAM and run it.

The partial bitstreams transmitted to the FPGA are 120 KB in length. The transmission time of the data over Internet depends on where the server is located. When the TFTP server is in the same LAN segment, the time is in the order of milliseconds, whilst the server is not in the same LAN segment the transmission time is not predictable, depending on the net status. The self-reconfiguration itself takes 19 milliseconds to be completed.

Selecting the first option, the application stored in the initial bitstream, in this case the Reed-Solomon application, can be executed. By selecting the second option the Cryptographic application could be downloaded to the FPGA using the ICAP port.

Once the system is reprogrammed, the menu appears again through the RS232 link and a new bitstream can be downloaded or a jump to the new application just downloaded can be made. After the user application is finished, the control returns to the boot menu. If the application is an infinite loop, which is very common in an embedded system, sending a “stop” command through the communication channel exits the loop and jumps to the boot menu again.

5.1. Reed-Solomon Encoder

The initial bitstream which is loaded in the FPGA at the start contains a Reed-Solomon Encoder application and its associated software.

Reed-Solomon [15] are block-based error correcting codes with applications in digital communications and storage. There is a wide range of different Reed-Solomon codes depending on the number of bits taken as input and the number of redundant bits added. In this case the core implements (n, k) code where n-k = 16 (8 byte error correction code). The selected code length was (255,239) which is one of the most used ones, for example many popular standards such as G709, DVB1 and DVB2 use it.

The application software receives a block from the communication channel, encodes it using the Reed-Solomon core and sends the result back through the channel.

The application takes up 456 slices and 3 block RAMS containing the associate software needed to run the application. This area is roughly 25% of the reconfigurable area.

5.2. Cryptographic Application

The other selected application is a Cryptographic Application made up of an AES core of 192 bits key length and an MD5 digest algorithm. As in the previous example, the partial bitstream also includes the associated software needed to test the application.

The Cryptographic Application takes a block, calculates the hash of the block and applies the AES algorithm over the concatenation of the block with the hash. Then, it decrypts the block, recalculates the hash and compares it with the original. If they are equal the test is OK, if not an error message is displayed through the communication channel.

The AES algorithm plus the MD5 algorithm take up 954 slices and 3 block RAMS for the application software, approximately 45% of the reconfigurable area.

6. Cost Saving Analysis

The first point that has to be taken into account to compare the two approaches is the amount of area needed to support this scheme in the FPGA.

The number of slices used by the SCP, the Security Controller, the ICAP controller and the Communication Controller is 3090, for a VirtexII FPGA. Assuming the number of slices the system occupies is independent of the FPGA size whilst the family doesn't change, the smallest FPGA needed to store the system is an XC2V1000 with 5120 slices. Therefore, the system has 2030 slices free for the reconfigurable area.

The price Xilinx offers for an XC2V1000 for high volumes is around 100$ depending on the encapsulate type and the speed grade.

The economic cost of the system using a traditional architecture is made up of microprocessor, communication controller and an FPGA, a smaller one in this case.

A 32 bit microprocessor in the market costs about 15$ depending on the performance of the microprocessor. The communication controller (Ethernet, RS232,…) cost is about 2$ each. Finally, the price of the cheapest FPGA required to fit in the design (XC2V500 with 3072 slices) is roughly 80$.

As can be seen, the economic cost of the parts for the design is very similar for both approaches, but there are other considerations to take into account.

Using our approach, the PCB is made up of only one chip, instead of four. That is translated into an economic saving in the PCB design and manufacturing, and also in the size of the PCB, which in this case is smaller than using a traditional scheme.

In this case the FPGA family used was a VirtexII, which is the most expensive one sold by Xilinx. Using for example a Spartan3 or a Virtex4 family the cost of the FPGA will be significantly smaller. Using these FPGA families and this new scheme, the economic cost of the system's parts could be reduced in a significant percentage.

7. Results

The area for the fixed part of the system is presented in Table 1:

Table 1. Synthesis results

	Slices
OR1200	2184
Security Controller	386
ICAP controller	14
RS232 Controller	440
Ethernet Controller	66
Total	3090

This area represents 60% of the total FPGA resources using an XC2V100 FPGA. Therefore, up to 40% of the FPGA could be freely use for the reconfigurable modules depending on the floorplanning. Using a bigger FPGA, like an XC2V3000 FPGA, the area for the reconfigurable part could be up to the 80% of the device.

Another important data to take into account is the partial reconfiguration time. Ignoring the bitstream transmission time that depends on the communication channel and its status, the time is made up of two different times. The first one is the time the system spends decrypting and checking the integrity of the received data and the second one is the proper reconfiguration time of the FPGA.

The system clock frequency is 25 Mhz. The DES algorithm takes 16 cycles decrypting a 64 bits block. The CRC takes only 1 cycle to perform the operation. Although the ICAP can accept data up to 50Mhz, this throughput is reduced because the data has to be taken from the memory and writen to the ICAP controller connected to the microprocessor which uses a shared Wishbone bus for instructions and data. This process is slow and makes the ICAP to receive a data every 20 clock cycles. Other architectures to solve this problem have being studied, but the most suitable would be a bitstream cache directly connected to the ICAP, to avoid accesses to the main memory. The measures gives that the whole reconfiguration takes 115 ms.

8. Challenges and Future Work

One of the greatest problems when facing Self Reconfiguration over Xilinx devices is the lack of tools to simplify the flow. One of the most important things to be done in this field is designing tools that help to design a self reconfigurable system. There are some initiatives trying to solve this problem such as [16]. Another problem is that the commercial boards available on the market don’t help the designer because the pin assignment is not appropriate for a partial reconfiguration flow since the pins driven by each module of the design have to be locked in the area occupied by that module. Future work in Partial Self Reconfiguration has to be focused on solving these two main problems.

Our future work will be aimed at supporting public cryptography capabilities to the system using an RSA core. This will allow making secure key session interchange and authentication of the data source.

Another step will be to create a graphical tool that helps to perform all these tasks in an easy way. Now all the process is automated using batch files, the idea is to make it simpler providing a graphical tool and a set of libraries and APIs that simplify the tasks of developing a complete system on an FPGA making possible to companies to use this architecture in a easiest way.

9. Conclusions

In this paper a secure architecture based on Open-Source cores has been presented. The capacity of new FPGA families (millions of equivalent gates) makes it possible to implement the whole system inside them, not requiring external device controllers or peripherals. The FPGA keeps its reconfiguration capability by using an area as an FPGA inside the FPGA. This reconfigurable area is reprogrammed by using the ICAP port by the FPGA itself.

In this scheme the partial bitstream is made up of the software and hardware needed in order to run the application. An OR1200 soft-core processor takes the bitstream from the communication channel and reprograms the reconfigurable part of the FPGA in a secure way. The viability of this approach was tested with the implementation of a Reed-Solomon encoder and a Cryptographic application.

10. References

[1] K. Brunham, W. Kinsner, “Run-time reconfiguration: towards reducing the density requirements of FPGAs”, CCECE, 2001, Volume 2, pp: 1259-1264 vol.2 B.

[2] R. Fong, S. Harper, P. Athanas, “A Versatile Framework for FPGA Field Updates: An Application of Partial Self-Reconfiguration”, Proceedings of the 14th IEEE International Workshop on Rapid System Prototyping, San Diego, CA, June 2003.

[3] B. Blodget, P.J-Roxby, E. Keller, S. McMillan, P. Sundararajan, “A Self-Reconfiguring Platform”, Proceedings of the 13th International Conference on Field Programmable Logic and Applications (FPL'03), pp. 565-574, Sept 2003

[4] M. Bolado, H. Posadas, J. Castillo, P.Huerta, P. Sanchez, C. Sanchez, H. Fouren, P. Blasco, “Platform based on Open-Source Cores for Industrial Applications”, DATE 2003, February 2003

[5] M. Bolado, J. Castillo, H. Posadas, P. Sanchez, E. Villar, C. Sanchez, H. Fouren, P. Blasco, “Using Open-Source Cores in Real Applications”, DCIS 2003, November 2003

[6] J. Castillo, P. Huerta, J. I. Martinez, “SystemC Design Flow for a DES/AES Cryptoprocessor” , WSEAS 2004.

[7] Xilinx: XAPP290, “Two Flows for Partial Reconfiguration: Module Based or Small Bit Manipulations”, 2002
[8] Xilinx , “Development System Reference Guide, Chapter 4, Modular Design”
[9] OpenCores, http://www.opencores.org

[10] D. Lampret, “OpenRISC 1200 IP Core Specification”, September June 2001.

[11] Silicore Inc., “WISHBONE System-on-Chip (SoC) Interconnection Architecture for Portable IP Cores.”, September 7, 2002

[12] FIPS, “Data Encryption Standard”, January, 1977
FIPS, “Advanced Encryption Standard”, November, 2001

[13] “RFC 1321 - The MD5 Message-Digest Algorithm”, April 1992

[14] B. Sklar, “Reed-Solomon Codes”

[15] P. Butel, G. Habay, A. Rachet, “Managing Partial Dynamic Reconfiguration in Virtex-II Pro FPGAs”, Xcell Journal , Fall, 2004