Implementation of a Reconfigurable Data Protection Module for NoC-based MPSoCs

Leandro Fiorin, Slobodan Lukovic
ALaRI, Faculty of Informatics
University of Lugano
Lugano, Switzerland
{fiorin, lukovics}@alari.ch

Gianluca Palermo
Politecnico di Milano
Dipartimento di Elettronica e Informazione
Milano, Italy
gpalermo@elet.polimi.it

Abstract

Security issues are emerging to be a basic concern in modern SoC development. Since in the field of on-chip interconnections the security problem continues to remain mostly an unexplored topic, this paper proposes a novel technique for data protection that uses the communication subsystem as basis. The proposed architecture works as a firewall managing the memory accesses on the basis of a lookup table containing the access rights. This module, called Data Protection Unit (DPU), has been designed for MPSoC architectures and integrated in the Network Interfaces near the shared memory. We implement the DPU inside an MPSoC architecture on FPGA and we add features to the module to be aware of dynamic reconfiguration of the system software. Starting from a general overview of our design down to components’ structure, we introduce the place and the role of the DPU module inside the system for a reconfigurable secure implementation of a MPSoC on FPGA. The description of the DPU concept, its implementation, and integration into the system are described in detail. Finally, the architecture is fully implemented on FPGA and tested on a Xilinx Virtex-II Pro board.

1. Introduction

System-on-Chip (SoC) development moves towards multiprocessor based and reconfigurable design. At the same time, the increasing complexity that will be required in next generation’s SoCs pushes designers to research new on-chip communication solutions. The Network-on-Chips (NoCs) [6, 8] concept has proven itself as a solid interconnection strategy that brings reliable, efficient and fast intercore communication. However, managing communication in modern embedded systems necessarily carries many security challenges related to the secure transfer and storage of critical data [13]. In fact, enabling a secure communication among the rapidly growing number of integrated IP cores is a task becoming more and more relevant. Therefore, there is an increased need of security-aware design solutions, in particular in the design of reconfigurable SoCs. However, even if the research on NoC related topics has been an emerging area of interest, security issues in this field have been often shadowed by other topics and not been explored to the same extent, being only recently addressed by the research community [13, 11].

In this work we discuss the implementation of a memory protection unit suitable for multiprocessor systems build on top of an on-chip network architecture. In fact, data protection is necessary in order to avoid extraction of sensitive information or the tampering of data and instructions by cores compromised by software-based attacks - such as in the case of those exploiting buffer overflow techniques [4] - or IPs configured maliciously. To show the effectiveness of our concept, we implement a NoC-based multiprocessor system in a Xilinx FPGA and detail the design trade-offs and costs of the discussed data protection module (DPU), in particular focusing on the aspects related to the reconfiguration of the module. The module allows performing secure accesses to memories and/or memory-mapped peripherals. It consists of a hardware solution that enables access to a particular memory space only if the initiator of the request is authorized to perform that particular operation, filtering request packets not satisfying access requirements. The protection module is adaptable to variable system scenarios, with predictable reconfiguration time overhead.

The paper is organized as follows: Section 2, after reviewing some relevant works on NoC-based reconfigurable systems, provides an overview of ongoing research on security problems and solutions implemented in NoC-based systems, focusing in particular on academic and industrial work on data protection in embedded systems. Section 3 gives a description of the NoC-based multiprocessor sys-
em implemented. In Section 4, we detail the characteristics of the data protection module presented, while synthesis results of the implementation of the multiprocessor system in FPGA are discussed in Section 5. Finally, Section 6 presents conclusions and future work.

2. Related Work

In this section we first give an overview of ongoing research in the field of reconfigurable systems adopting a NoC as communication infrastructure. Moreover, we outline and discuss related academic works addressing security aspects related to NoCs implementations, together with academic and industrial proposals to implement memory data protection in systems adopting conventional buses for on-chip communication.

In [16], a FPGA architecture adopting a hardwired NoC as interconnection medium is discussed. The system proposed is implemented on top of the communication infrastructure, providing a cost-efficient statically and dynamically reconfigurable architectural solution. In [5], the communication problem among modules dynamically placed on a reconfigurable device is approached using a dynamic NoC, through which the components placed at run-time on the device can mutually communicate. Nollet at al. [17] developed a run-time resource management scheme that is able to efficiently manage a NoC containing fine grain reconfigurable hardware tiles, while in [2] a dynamically reconfigurable NoC architecture is proposed for reconfigurable multiprocessor System-on-Chip (MPSoC), with the aim to satisfy increased communication needs, low cost of the silicon implementation, Quality of Service and scalability.

With regard to work addressing the specific topic of security in NoC-based architecture, in [14] and [15] a framework to secure the exchange of cryptographic keys in NoC is presented. Secure and not-secure cores can exchange critical messages adopting public key cryptography. Moreover, the proposed methodology ensures that no unencrypted key leaves cores on the NoC-based architecture and additionally supports IP secure cores running only trusted software.

The risks associated to a reconfigurable system based on NoC are presented by Diguet at al. in [10]. A first solution to secure such a system is discussed in the work. Main elements of the system are the Secure Network Interfaces (SNIs) and a Secure Configuration Manager (SCM), which respectively provide filtering of the communication and management of the configuration of system resources.

In [11] and [13], security problems that may affect a NoC-based system are outlined, while providing guidelines to enhance protection from external attackers. Focusing on related work on memory data protection in embedded systems, a specific implementation - aiming to provide protection for data stored in memory in AMBA based system - is described in [7]. Similar approaches to selectively allow access in memory are also provided by ARM in its AXI TrustZone memory adapter [3] and by Sonics in its SMART Interconnect solutions [1].

In this paper we discuss the FPGA implementation of the concept presented in [12], extending the work in order to include the possibility to reconfigure the characteristics of the module. Moreover, we present its implementation within a multiprocessor system realized exploiting the functionality offered by the Xilinx development environment.

3. Overview of the System Implemented

In this Section, we detail the implementation of the NoC-based system on FPGA. We give a short overview about the general system implemented and we provide architectural information about the basic blocks of the NoC, i.e., the Network Interface (NI) and the Router, and we highlight the challenges and design issues related to their implementation on FPGA.

3.1 Platform description

The described NoC has been customized for implementation on a Xilinx Virtex-II Pro FPGA board [19]. The final system implemented is shown in Figure 1. It is composed of two MicroBlazes, shown in the Figure as $\mu B_0$ and $\mu B_1$, and a block of shared memory implemented using part of the BRAM available in the Xilinx board. MicroBlaze is the soft-core RISC processor provided by Xilinx, while BRAM is the on-chip Block RAM synthetisable using board resources. The interconnection infrastructure is composed of a three-port router and the NIs, providing a custom interface to the two type of IP blocks present in the system. As
3.2 Network Interface

The Network Interface is employed to adapt a communicating core to the on-chip network. The module, acting as interface between the core and the communication subsystem, hides to the processing elements all the issues related to a reliable and efficient transmission of the data through the network. The NI is in charge of structuring data as packets, and to manage transmission of necessary control flow information. This module is in charge of providing an interface to the transmission protocol implemented in the core and an efficient packetization of the data to be transmitted. Moreover, among the other tasks, it is also in charge to guarantee the necessary bandwidth and latency for the transmission and to provide additional services, such as security [9].

3.2.1 Interface to the MicroBlaze

Figure 2 shows the internal structure of the NI to the MicroBlaze. As shown in the figure, NI consists of two FIFOs, one for each direction of the data flow, and control logic that implements the basic services previously described. The NI takes data and control signals from the FSL interfaces of the soft-core processor. In fact, the FIFO-like structure of the FSL allows its easy integration in the NI, adapting the control signals of the internal queues to those of the FSL interface. The MicroBlaze drives the Master FSL interface (FSL - M) to transmit information to the NI, where we implemented a Slave interface (FSL - S). Transmission of data from the NI are driven through the Master FSL interface implemented in the NI.

In order to distinguish between control signals and data sent by the MicroBlaze, we define the following protocol. As first step, the processing element sends information on the communication (see Figure 3), such as the destination address (DestAddr), the data length (DLength) and the type of operation to be performed on the destination (i.e. load (L) or store (S)). IDKey is added for future implementation of identification techniques for the IP, while the Opt can be used for some extra tasks.

A control bit is provided by the FSL to indicate whether the transmitted information is a control or data word. The control bit is set to high (1) if the word is a control one (as shown in Figure 3, it contains destination address, length, type of operation and other optional information bits). When transmitting data, this control bit is set to 0. A number of data words, equal to the value specified in DLength, follows the control word. Furthermore, transitions of the control bit indicate that a new packet is to be sent to the network.

In the NI, we implemented a memory-mapped protocol in which the operations are expressed in terms of read and write to memory addresses. The NI translates a range of memories in the related identifier of the node in the network. The transaction-based protocol implemented is shown in

![Figure 2. Internal implementation of the Network Interface to the MicroBlaze](image)

<table>
<thead>
<tr>
<th>DestAddr</th>
<th>DLength</th>
<th>IDKey</th>
<th>L/S</th>
<th>Opt.</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>5</td>
<td>10</td>
<td>15</td>
<td>16</td>
<td>31</td>
</tr>
</tbody>
</table>

![Figure 3. Control and data words sent by the MicroBlaze to the NI through the FSL interface](image)

![Figure 4. Transaction based protocol implemented between Initiators and Targets](image)
Figure 4. A store request from the initiator of the transaction, directed to the desired target, is immediately followed by the data to transfer. The target answers with a positive acknowledgment in case of successful transaction; with a negative acknowledgment in case of unsuccessful ones. A load request from the initiator of the transaction is followed as positive acknowledgment by data transmitted by the target. A negative acknowledgment is sent in case of problems in the transaction. In our discussion we focus on the network level. Therefore, we assume no signal loss in the transmission of the packets through the NoC.

3.2.2 Structure of packets in NoC

The packets’ structure used within the network is shown in Figure 5. We adopted a wormhole control flow in the transmission of the packets. Therefore, our packet is divided in flits, which in our case represent the smallest information logically and physically transmitted through the network. As shown in the Figure 5, there are different types of flits. In order to distinguish which type of flit is transmitted, two control bits are used (Flit Type in Figure 5 - bus width is therefore of 34 bits: 32 of data plus 2 of control).

As shown in Figure 5(a), the first flit of a packet is labeled setting the Flit Type control bits to ‘10’. Last flits, closing the packets, are labeled with ‘01’, while intermediate flits are identified with Flit Type set to ’00’. Packets composed of just one flit are labeled setting Flit Type equal to ’11’ (Figure 5(b)).

The first flit of each packet contains the header, which carries information about the network layer (bit 0 to bit 9) and about the transaction-based protocol implemented. We grouped the two information in the same flit, in order to reduce the overhead associated to the header of the packet. DestID identifies the target node and its value is calculated translating the DestAddr send by the processing element. SourceID univocally identifies the source node of the transaction and its value is given by a hard-wired register in the NI. The register is necessary in order to be able to identify the initiator of the transmission. Length represents, in number of words, the length of the data that follows the header of the packet. Type of access requested, i.e. load or store (L/S), is also sent, as well as the role (Role) assumed by the processing element (super-user or user) and some optional bits. Role has been included for future improvements of the system, allowing an identification of the operative mode of the processing elements.

The structure of the packet used for positive and negative acknowledgments (ACK and NACK) is shown in Figure 6. As previously said, acknowledgments are sent in case of successful/allowed writing (ACK) or unsuccessful/rejected read or write (NACK). Acknowledgments are sent directly back to the NI of the processing element issuing the request. Acknowledgment packets are composed only of the header and therefore of just one flit. The fields containing information related to the routing are equivalent to those present in packets transferring data, while the following bits are set to one to notify an ACK and to zero for a NACK.

3.3 Router

We design a router with variable length of the input and output queues and implementing a table-based routing algorithm. The router can be automatically generated with a variable number of input and output ports (see Figure 7). The needed information for the routing is extracted from the first flit of the packet. The destination address in the header of the packet (DestID) is looked up and the related output port is calculated. A request of utilization is therefore risen to the arbiter associated to the selected port. The arbiter, in case of non utilization of the associated port, as-
signs in Round Robin the use of the queue to the input port requesting it and sets up the switch fabric in order to directly connect the selected input port with the output port. We implemented the switch fabric in FPGA using a combination of multiplexers. When the last flit of the packet is received, the arbiter releases the queue and assigns it to the next input port requesting its use.

4. Data Protection Unit

The Data Protection Unit (DPU) is a hardware module that enforces access control rules specifying the way in which a component connected to the NoC can access the blocks in which a memory can be divided to allow separation between sensitive and non-sensitive data of different processors [12]. The module is embedded in the Network Interface of the target memory (or of the memory-mapped peripheral) to supply services similar to those offered by a classical “firewall” in data networks. The Network Interface receives packets coming from several initiators requesting access to the target memory. While processing the packet, the information contained in the header is passed to the DPU. The protection module looks up the access rights for the requesting packet and checks if the requested operation is allowed, granting or denying the access of the data to the memory block.

The most relevant part of the DPU is represented by the lookup table. In hardware this element is commonly implemented combining a typical Content Addressable Memory (CAM) [18], used in fully associative memory and data networks routers, and a RAM storing the access rights (load, store, both or none). It is important to note that coupling the DPU with the NI guarantee that no additional latency is associated with the access right check since, as we will show better later, the protocol conversion and the DPU access are performed in parallel.

4.1 Packet used for memory transactions

In order to describe the features offered by the DPU and its implementation on the FPGA, we briefly present now the packets structure to implement memory access in target memory in NoC-based architectures. The implementation proposed for the DPU depends on the specific structure of the packet, even if approaches similar to the one explained here can be easily employed for protocols different from the one presented.

We used the Opt. field in the header of the packets described in Figure 5 to transmit the memory address to which the initiator is requesting access. The Length field is used to communicate the length, in number of words, of the information sent or to be retrieved, while the other fields assume the meanings previously described.

4.2 DPU architecture

The architecture proposed for the DPU is shown in detail in Figure 8. Each entry in the lookup table is indexed by the concatenation of the SourceID and the requested memory address MemAddr. The RAM of the lookup table stores the access rights for the two different possible roles of the initiator (Role 0 - load, store; Role 1 - load, store). This means that all the data within the same block have the same rights.

We use a Ternary Content Addressable Memory (TCAM) [18] to compact the table, grouping ranges of keys in one entry. TCAM, in addition to logic '1' or logic '0', allows to store a don't care (X) value in those positions in which either a '1' or a '0' matches the entry key. A ternary symbol is encoded adding to a CAM cell the storage for a mask bit, set to logic '1' to not consider the value stored in the CAM, and at the (approximated) cost of an additional memory cell. In order to minimize the overhead, we use TCAM only for those bits looking up the requested memory address (MemAddr), while CAM cells for the other fields of the entry key. In fact, for efficiency, only allowed accesses are recorded in the table and we believe to be more convenient to add an entry line to specify the access right of an initiator, instead of maintaining the fixed overhead due to a global TCAM.

On the other hand, the use of the TCAM for the MemAddr bits allows to lookup all the requests to the same memory block. Moreover, it allows us to check that the length of the data requested to be loaded or stored does not exceed the block boundaries. In fact, the address space of the memory blocks it is delimited by the starting address stored in the CAM and by the sum of the values in the CAM and those in the mask bits.

As shown in Figure 8, we modified the common TCAM architecture in order to provide as output also the upper bound address of the memory block (upper_bound signal). Once the packet header is received, the length information
4.3 Interface to BRAM

As mentioned above, the DPU is embedded in the NI that interfaces to the interconnection network the BRAM used as shared memory in our system.

As shown in Figure 9, the NI embeds the controller of the BRAM, which is in charge to handle write/read accesses to memory. The BRAM controller works in parallel with the DPU controller. Both controllers are implemented as state machines that take the same input and process it in parallel. The UML activity diagram for an access in memory is shown in Figure 10. Upon the arrival of a new first flit containing the header of the packet, access information are passed by the NI to both controllers. While the DPU controller checks the access rights of the request, the BRAM controller sets the memory block for the read/write access. In case of a request satisfying the access rules specified in the DPU, data are read or written to memory and an appropriate acknowledgment packet is sent to the initiator of the transaction. If the request is not allowed, the packet is discarded and a packet of negative acknowledgment is sent to the initiator.

Figure 9. Internal implementation of the Network Interface to BRAM embedding the DPU

Figure 10. Activity diagram of the NI embedding the DPU: access to memory and reconfiguration

4.4 Reconfiguration of the DPU table

In order to make more flexible and more efficient the proposed solution for data protection, we designed the DPU module to be reconfigurable. In fact, in a general software environment the required memory access rights can change dynamically with the evolution of the applications (and/or the system). This requires an update of the data protection characteristics in order to satisfy the security requirements of the changing applications.

To enable this feature, we added a write port and another memory unit (called shadow memory) to the basic DPU architecture. The additional write port is used to update the DPU table, while the memory module is used to store the new DPU values. In fact, the shadow memory stores the necessary information to reconfigure the DPU that allows to satisfy the security requirements of the following application scenario. When it is necessary to let the system switch to a new scenario, the shadow memory is updated with the new information. Only when the shadow memory has received all the new table values from the processing element that started the table update (the controller of the overall system) and the reconfiguration signal has been issued, the
values on the shadow memory can be committed to the DPU table. This method avoids a transient behaviour of the DPU during the updating since committing from the shadow memory is faster than a remote update. Packets that arrive from the NoC during the reconfiguration wait in the input queue of the NI the end of the process before being analyzed.

Another important issue is that the reconfiguration phase for the DPU can be performed only from selected processing elements (PEs) in secure mode, since otherwise the reconfiguration of the shadow memory can be used as base for attacks. The implemented DPU has been designed to be memory mapped as a normal peripheral and the control of the access rights to the shadow memory is done by adopting a protection strategy equivalent to the one just described for the addresses in the normal memory blocks. Requests of write access to the shadow memory are therefore restricted to selected PEs with the required access rights.

5. Synthesis Results

In this section we present synthesis results of the multiprocessor system previously described and shown in Figure 1. As already discussed, the presented architecture is composed of two microprocessors (MicroBlazes), one shared on-chip memory block (BRAM) of 64 KB, and the interconnection system, in which we implement our module for data protection. The whole architecture was developed on a Xilinx Virtex-II Pro XC2VP30-FF896 board, by using the Xilinx Embedded Development Kit version 8.2. The system was implemented to work at the operative frequency of 100 MHz. For test and debug purposes, an RS232 interface was used for the communication between the board and the host computer.

Table 1 shows FPGA resources utilization of the overall system, in the case in which a DPU with respectively 4 and 8 lines is implemented. Numbers reported in the tables refer to an implementation of the elements of the system as shown in Figure 1. In the first case, the total number of equivalent gates is equal to 6,757,418, while in the second case is 7,057,675. A DPU with 4 entry lines is able to protect a memory divided into 2 protection regions from all the possible types of access request (load/store) issued by the two processing elements in the system. In fact, the number of entry lines is given by the product of the number of PEs and the number of protection regions in which the memory is divided. A DPU with 8 entry lines can protect the same system with a memory with up to 4 protection regions.

Table 2 shows the ratio between the dimension of a DPU with 8 entry lines and those of the other components of the system. As it is possible to notice, the dimension of the DPU is almost equivalent to the dimension of the NoC router and it represents a significant part of the NI to BRAM.

In Figure 11 we show, in number of occupied slices, the area of different configurations of the DPU. As also shown in [12], the number of slices occupied by the implementation of the data protection module is directly proportional

<table>
<thead>
<tr>
<th>System with DPU (4 lines)</th>
<th>Used/Available</th>
<th>% Used</th>
</tr>
</thead>
<tbody>
<tr>
<td>Slice Flip Flops</td>
<td>2,465/27,392</td>
<td>9</td>
</tr>
<tr>
<td>4 input LUTs</td>
<td>5,671/27,392</td>
<td>20</td>
</tr>
<tr>
<td>Occupied Slices</td>
<td>3,371/13,696</td>
<td>24</td>
</tr>
<tr>
<td>Block RAMs</td>
<td>104/136</td>
<td>76</td>
</tr>
<tr>
<td>MULT18X18s</td>
<td>6/136</td>
<td>4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>System with DPU (8 lines)</th>
<th>Used/Available</th>
<th>% Used</th>
</tr>
</thead>
<tbody>
<tr>
<td>Slice Flip Flops</td>
<td>2,610/27,392</td>
<td>10</td>
</tr>
<tr>
<td>4 input LUTs</td>
<td>5,891/27,392</td>
<td>21</td>
</tr>
<tr>
<td>Occupied Slices</td>
<td>3,527/13,696</td>
<td>25</td>
</tr>
<tr>
<td>Block RAMs</td>
<td>104/136</td>
<td>76</td>
</tr>
<tr>
<td>MULT18X18s</td>
<td>6/136</td>
<td>4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>DPU (8 lines)/Component</th>
<th>Consumption ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>DPU/Router</td>
<td>1.06</td>
</tr>
<tr>
<td>DPU/Microblaze</td>
<td>0.25</td>
</tr>
<tr>
<td>DPU/NI (with DPU)</td>
<td>0.83</td>
</tr>
</tbody>
</table>

Figure 11. Area of the DPU for different numbers of entry lines
to the number of entry lines, and it represents the dominant part in the area overhead of the NI to BRAM.

6 Conclusions and Future Work

In this work we presented an FPGA implementation of a NoC-based multiprocessor system, in which we included a hardware module coupled with the network interface that gives the possibility to perform secure accesses to memories and memory-mapped peripherals. FPGA technology was selected because it allows a fast design, implementation and testing of the system. The developed data protection module introduces no additional delay in the system and it is transparent to the processing elements and memory block. We studied the possibility to run-time reprogram the data protection module, in order to provide additional flexibility and better control over the system security. Implementation costs and overhead associated with different FPGA implementations were analyzed and presented.

As future work, we will include in an automatic FPGA design flow the generation of the NoC with the security solution presented. Moreover, we will analyze the interaction of the module proposed with the Operating System and the system software. In fact, our objective is the possibility to secure the memory access not only referring to the initiator but also to the role assumed and, with finest granularity, to the threads running on the processors. This could enable the possibility to perform the migration of the threads across the network together with their associated access rights.

7 Acknowledgments

This work has been carried out under the MEDEA+ LoMoSA+ Project and is partially funded by KTI - The Swiss Innovation Promotion Agency - Project Nr. 7945.1 NMPP-NM.

References