4

Concepts of Communication and Synchronization in FPGA-Based Embedded Multiprocessor Systems

David Antonio-Torres
Tecnologico de Monterrey Campus Puebla
Mexico

1. Introduction

The advent of deep submicron (DSM) technologies has driven the evolution of very large scale of integration (VLSI) chip design to new frontiers. In the early days, digital system design was tackled as an integration of discrete systems in a printed circuit board (PCB), the so-called jelly-bean design (Maxfield, 2004). If the complexity of the task was large, the resulting PCB was expected to be full packed with discrete components, thus pressing the designers with concerns such as area, power consumption, speed, etc. The further shrinking dimensions of the transistors allowed the designers to pack some similar discrete components together into a single one, thus alleviating design concerns about signal integrity, connectivity, etc. However, it was later found that some other aspects of design could also be alleviated if a larger number of the discrete components of the PCB were integrated into a single chip; this design strategy introduced the concept of System on Chip (SoC); an SoC is then defined as an integrated circuit that incorporates on-chip memory, hardware and software components, where the software components are to be run by one or more microprocessors or digital signal processors (DSP) (Chang et al., 1999).

It can be argued that SoC design has caused a great impact on consumer markets such as mobile communications. However, the market, in return, has pushed for convergence of multimedia, communication and signal processing into a single consumer electronic device, the so-called multimedia embedded systems, leading the designers to rethink the design strategy. Due to the massive data processing, the number of hard deadlines that such systems impose and the parallelism that the previous two entail, the presence of more microprocessors was mandatory. As a result, the latest incarnations of SoC are now termed Multiprocessor System on Chip (MPSoC). An MPSoc truly implements parallelism in the form of hardware threads, that is, each microprocessor present in an MPSoC executes a thread of execution in its own right. While the inclusion of more microprocessors provides more computing power to deal with stringent processing conditions, several other design concerns have been brougth about. Indeed, low-power operation, architecture design, performace predictability, on-chip communication, among others, are still open to design and exploration in the MPSoC field (Jerraya & Wolf, 2005).

Design of MPSoCs is a complex task that requires the contribution of several disciplines: computer architecture, embedded microprocessors, low-power design, physical and layout implications of DSM technologies, parallel computing, operating systems, network protocols, etc. In addition, the type of applications that an MPSoC is expected to run is not trivial and a correct tuning between the selected MPSoC architecture and the application is a lengthy process (Jerraya & Wolf, 2005). As a consequence, a few papers report performance comparisons between several architectures for a given application. Furthermore, due to the number of complexities around an MPSoC design, low-level details about synchronization and communication, which play a differentiation factor between architectures, are not fully clarified (Li & Hammami, 2009), (Yu & Schaumont, 2006). This chapter intends to shed some light to internal mechanisms of synchronization and communication in three dominant architectures in MPSoC: shared memory, message passing and Network on Chip (NoC). With the purpose of further clarifying those mechanisms, this chapter also proposes the use of an easy-to-program embedded microprocessor: PicoBlaze (Xilinx, 2011a). PicoBlaze is an 8-bit embedded microprocessor, in the form of soft core, developed by Xilinx and targeted for some families of FPGAs. These important characteristics of PicoBlaze are further reinforced with its programming in assembly language. Likewise, a generic dual core architecture proposed by Xilinx (Asokan, 2007) influences the development of the MPSoc architectures reported in this chapter. Consequently, PicoBlaze lends itself to implement a design flow for embedded systems whose architecture can be very easily simulated and prototyped. It is expected that with the use PicoBlaze and a design environment built around it, intricacies of communication and synchronizarion of those three MPSoC architectures can be unveiled and compared.

This chapter is organized as follows. Section 2 describes the tools of the design environment for the design and verification of the three proposed MPSoC architectures and provides details of the programming and simulation of PicoBlaze. Architectural features and the hardware blocks that provide the mechanisms of synchronization and communication for the multiprocessor architectures are identified and explained in section 3. Section 4 provides the details of implementation of the multiprocessor architectures using PicoBlaze. As the main focus of this chapter is on low-level details, the information provided in section 4 comes in the form of VHDL and assembly-language code. Also, section 4 evaluates the performance of the MPSoC architectures in terms of latency and area. Section 5 draws some conclusions and outlines the future work. Finally, acknowledgements and references are provided in sections 6 and 7 respectively.

2. FPGA-based design platform

One of the proposals of this chapter is the use of FPGAs as a platform for the evaluation of multiprocessor architecture. Due to its reconfigurable fabric and short design times, FPGAs lend themselves quite readily for the exploration of new architectures. Along with the selection of FPGAs for the exploration of new architectures, the proposed design platform includes a set of tools that speeds up the design flow and facilitates the evaluation of the designed architectures. This section explores the three components that make up the proposed platform design: FPGAs, PicoBlaze and Xilinx ISE.

2.1 FPGAs

The use of FPGA platforms in the exploration of new computer architectures and embedded systems has proven successful (Li & Hammami, 2009), (Yu & Schaumont, 2006). Along with the introduction of FPGAs in this field, languages for hardware description (HDL) such as VHDL and Verilog also help promote the use of reconfigurable logic devices in the exploration of new digital systems (Skahill, 1996), (Brown & Vranesic, 2000). An example of a design environment for the design of hardware and software components of embedded systems is EDK of Xilinx (Xilinx, 2011b). EDK relies on a library of reusable software components, a library of reusable hardware blocks and a set of tools that enable the programming and evaluation of embedded systems in an FPGA. Relying on the use of EDK for the design of embedded systems, Xilinx has also proposed a multiprocessor architecture based on two 32-bit microprocessors. Features of this proposed architecture are given later in this chapter.

2.2 PicoBlaze

PicoBlaze is an 8-bit embedded microprocessor developed by Xilinx and targeted for some families of FPGAs. The rationale for the selection of PicoBlaze for the development of MPSoC architectures are listed below.

- PicoBlaze has a simple and very flexible architecture in terms of the configurability and inclusion of peripherals
- Reduced instruction set and easy programming and simulation
- Because of the port-mapped peripherals approach that it implements, communication with peripherals is simplified and coding effort is reduced
- Its simple architecture and easy programming is in line with one of the objectives of this chapter, which is to highlight the characteristics of the syncronization and communication aspects of three multiprocessor architectures.

  While complex applications call for the use of more robust embedded microprocessors, it is the belief of the author that by reducing the complexity of the programming of the microprocessors and its necessities for communication the low-level exploration can be more focused on the characteristics of synchronization and communication required by the multiprocessing architecture.

2.3 Xilinx ISE

ISE is the flagship tool of Xilinx in the market of digital design (Xilinx, 2011c). ISE plays a key role in the proposed design environment, as it provides a set of entry tools for the capture of functionality of the design, a gate synthesizer and an FPGA programmer. In particular, this chapter makes use of a well proven approach: synthesizable VHDL descriptions for the hardware components of the MPSoC architectures and development of assembly-language code for the applications that the PicoBlaze instances are to run (Antonio-Torres et al., 2009). As ISE is focused on the development of hardware blocks, its use is complemented with the use of a couple of tools for assembly-language programming: the assembler KCPSM3 (Xilinx, 2011a) and the instruction-set simulator for PicoBlaze pBlazeIDE (Mediatronix, 2001). The three tools give shape to the design flow outlined below.

- The assembly language applications are developed for all the PicoBlaze instances of the architecture
- The applications are then assembled with KCPSM3 and translated into VHDL descriptions of read-only memories that hold the applications. The assembly-language applications translated into VHDL code allow the integration of the software development stages with the hardware design phases
- The design of the MPSoC architecture is further developed within ISE under a hardware design approach. If needed, the simulation of the software applications interacting with the hardware components can be conducted in the form of waveform analysis. As a matter of fact, this is the approach that is used to validate the proposed MPSoC architectures
- Upon completion of hardware design and simulation, the MPSoC architectures can be synthesized in an FPGA and concerns, such as area, can be evaluated

Once the proposed design flow and its supporting tools have been presented, the next section introduces the three multiprocessor architectures proposed for exploration.

3. Multiprocessor architectures

Due to its complexity and the huge design space that multiprocessor architectures call for, it can be argued that FPGA is a good candidate as a platform for evaluation and prototyping. In line with this idea, Xilinx has proposed a Generic Dual Processor Architecture (Asokan, 2007), (Kowalczyk, 2003), which is shown in Figure 1. This architecture is to be developed with two Xilinx tools: the Embedded Design Kit (EDK) and Software Design Kit (SDK), both of which are components of the tool Xilinx Platform Studio.

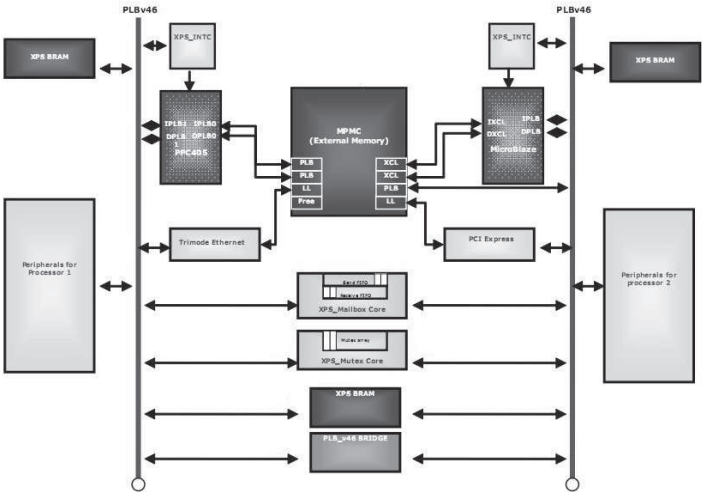


Fig. 1. Generic dual processor architecture proposed by Xilinx

The Generic Dual Processor Architecture is composed of two microprocessors. Depending upon the selected family of FPGAs, the microprocessor instances can be MicroBlaze or PPC405; MicroBlaze is a 32-bit soft core, while PPC405 is a 32-bit hard core available in some

families. Additionally, each microprocessor can be deployed with a set of local peripherals, optionally supported by an interrupt controller. Playing the role of shared resources, an external memory with a multiport memory controller (MPMC) and an XPS BRAM are considered in the architecture; a bus bridge is also included, allowing the access of the two microprocessors to the two sets of, no longer, private peripherals. Implementing the mechanisms for communication and synchronization of the microprocessors are a mutex, XPS_Mutex (Xilinx, 2009b), and a mailbox, XPS_Mailbox, (Xilinx, 2009a); note that both are supplied by EDK in the form of VHDL descriptions and complemented with low-level software drivers. The mutex is in charge of arbitrating the accesses of the microprocessors to the shared resources, thus avoiding contention in case of simultaneous accesses. On the other hand, the mailbox embodies a channel of bidirectional communication between the microprocessors. As the mutex and the mailbox supplied by EDK are targeted to MicroBlaze and PPC405, the following sections provide details of how they can be tailored to the communication needs of PicoBlaze.

3.1 Shared memory

Figure 2 shows the concept of the shared memory architecture, where two microprocessors are attached to a bus along with a set of memory blocks and controllers of input/output devices. Before a microprocessor accesses one of the shared resources, it must request control to the mutex; in high-level language, this request comes in the form of a LOCK command (Culler et al., 1998). Once the control over the resource is granted by the mutex, the microprocessor is free to access the shared resource. When the task is completed, the microprocessor must release the control over the resource; in high-level language, the release operation comes in the form of an UNLOCK command (Culler et al., 1998).

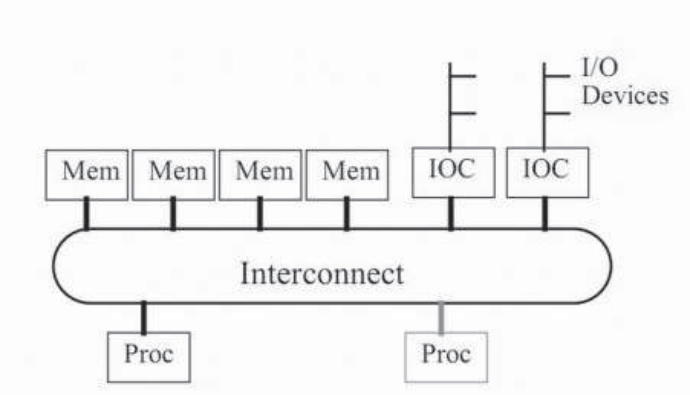


Fig. 2. Traditional organization of a shared memory architecture

At the core of a mutex, there is a finite-state machine (FSM) playing the role of an arbiter (Brown & Vranesic, 2000). Figure 3 shows an arbiter for three microprocessors that can lock/unlock one shared resource. The FSM must have as many *Gnt* states as microprocessors present in the system. Furthermore, the mutex, as a component, must provide an access port to each microprocessor in the architecture.

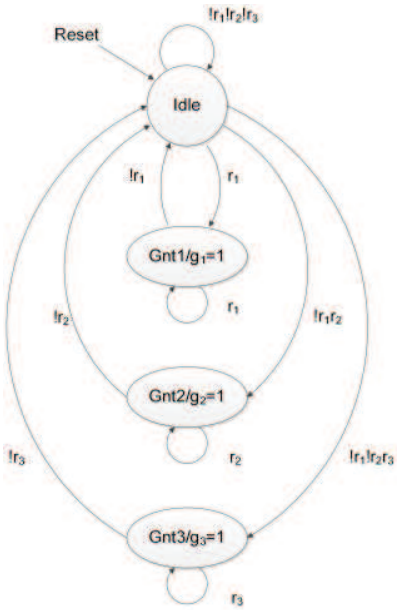


Fig. 3. A finite-state machines as a mutex for the arbitration in a shared memory architecture

As can be seen in Figure 3, the arbiter resolves contention by assigning priorities to all the microprocessors present in the system. The interaction between a requesting microprocessor and the arbiter is described as follows

- The microprocessor issues the command LOCK to the arbiter along with its identifier (this identifier is usually termed CPUID)
- If more than one microprocessor simultaneously try to LOCK the arbiter, the contention is resolved according to priorities
- The microprocessor reads back an internal register of the arbiter to retrieve the CPUID of the microprocessor that currently holds control over the shared resource
- If the CPUID read back from the arbiter coincides with that of the requesting microprocessor, the microprocessor can then take control over the shared resource; else the microprocessor issues again the LOCK command until it gains control over the resource
- Once a microprocessor has gained control over the shared resource, it proceeds with the required task and, upon completion, it must relinquish the control by issuing the command UNLOCK along with its CPUID

3.2 Message passing

Instead of relying on shared resources and requesting their access to a mutex, the message passing architecture identifies a mailbox as a bidirectional channel of communication between microprocessors. A mailbox allows two microprocessors to synchronize, share data and exchange status information. Figure 4 shows a typical message passing architecture that relies on mailboxes for interprocessor communication.

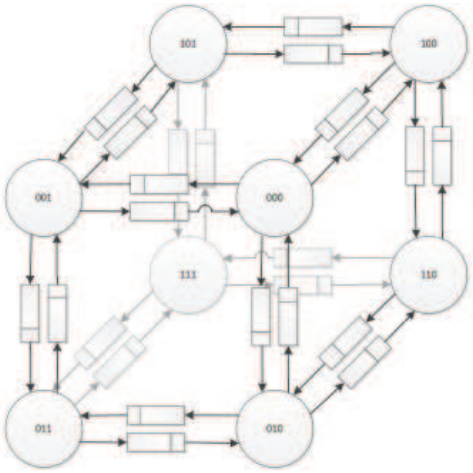


Fig. 4. Message passing architecture relying on mailboxes for interprocessor communication

As will be discussed later, a mailbox is composed of two first-in first-out (FIFO) memory buffers. A FIFO features two access ports, a read port and write port, see Figure 5. The important characteristic of a FIFO is that it keeps the data stored according to the order they were pushed into. A FIFO also features two status flags: an EMPTY flag that, when asserted, indicates that no data is available to be read and a FULL flag that, when asserted, indicates that there is no room for a new data to be written.

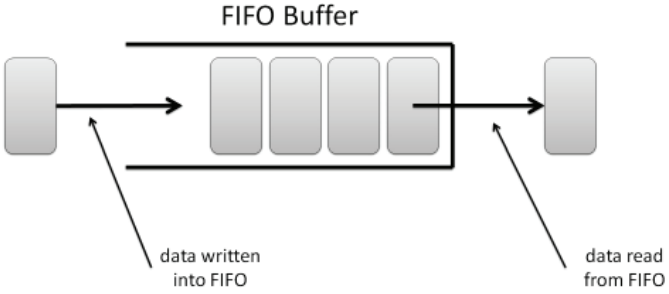


Fig. 5. Concept of a FIFO memory buffer with two access ports

The implementation of a mailbox with two FIFO buffers allows the communication between two microprocessors to be bidirectional, that is, one microprocessor is assigned the read port of FIFO 0 and the write port of FIFO 1, while the other microprocessor is assigned the read port of FIFO 1 and the write port of FIFO 0. The communication between the microprocessors initiates when microprocessor 0 writes a data to a write port, in which case it is said that the microprocessor 0 has issued a SEND command, and completes when microprocessor 1 reads that same data at the corresponding read port, in which case it is said that microprocessor 1 has issued a RECEIVE command (Culler et al., 1998). The interaction of the two microprocessors with the mailbox follows the procedure described below.

- Microprocessor 0 inspects the FULL flag of its corresponding write port and if the flag is zero it issues a SEND command
- Microprocessor 1 inspects the EMPTY flag of its corresponding read port and if the flag is zero it issues a RECEIVE command
- To implemente the communication in the opposite direction, microprocessor 0 and microprocessor 1 shift places with respect to the conditions given above

3.3 Network on chip

In addition to the well-known architectures described above, a recent exploration in the design of MPSoCs is Network on Chip (NoC). An NoC builds up interprocessor communication by borrowing some concepts from macronetworks theory (De Micheli & Benini, 2006), (Bjerregaard & Mahadevan, 2006). Instead of allowing microprocessors, or, in general, any processing element (PE), to be attached to a bus, an NoC provides each PE with a multipoint access to the rest of PEs in the system. Figure 6 shows the concept of an NoC. Note that the small circles represent routers that define the path of the data.

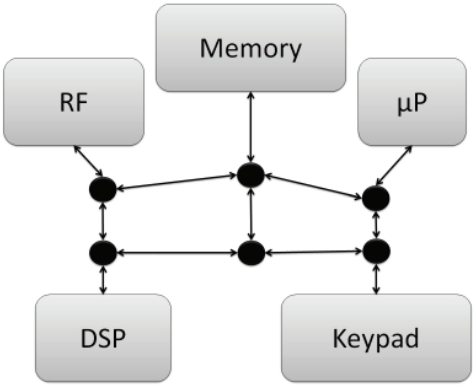


Fig. 6. Concept of a network on chip

From the perspective of an NoC, each PE implements a piece of functionality in the architecture and their interaction gives shape to the functionality of the whole system. In order to provide each PE with a channel of communication with the rest of the system, the concept of a network is introduced. Among the main tasks of the network are to identify the receiver of the message and establish a link between the source and the destination point; at each end of the link is a PE. Figure 7 shows a simple 4-by-4 grid NoC with global on-chip communication for the existing PEs.

Figure 7 is also helpful for the identification of the fundamental components of the network, which are described next.

- A network adapter interfaces between the core and the NoC, thus separating aspects of computation and communication in the system
- The routing nodes establish the path of the data according to the protocol of the network
- The links provide the channels of communication for the routing nodes

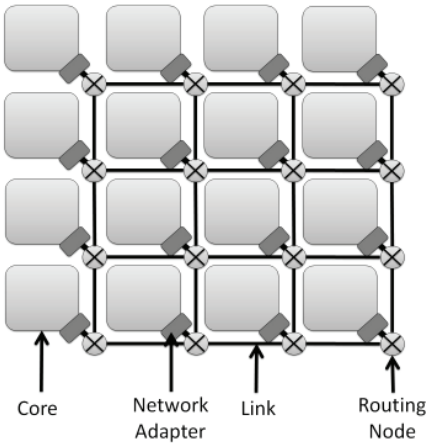


Fig. 7. 4-by-4 grid NoC with the main components of a network

The next section provides the hardware and software details of the implementation of the three multiprocessor architectures just presented. The hardware details are given in the form of VHDL descriptions, while software details come in the form of assembly-language routines. Additionally, aspects of evaluation are also discussed in order to put into perspective two important aspects of an MPSoC: latency and scalability.

4. Programming and evaluation

This section provides the low-level details of the comunication between a number of PicoBlaze instances and the components of the on-chip communication for the three types of multiprocessor architectures described in the previous section. In addition, the performance of the architectures will be evaluated in terms of latency and scalability. The latency of each architecture will be reported in terms of clock cycles or instruction cycles; as PicoBlaze features a fixed instruction cycle for the whole instruction set, the relation between clock and instruction cycles is quite straightforward. The scalability will be reported in terms of further modifications of the hardware components currently present in each architecture.

The three architectures are going to run a simple application that consists in adding eight 8-bit numbers that are stored locally. All the microprocessors run an exact copy of the application. Each microprocessor is going to be identified with a unique CPUID. As part of the evaluation and to facilitate debugging, microprocessor 1 is assigned CPUID 0x01 and adds numbers from 0x10 to 0x17; microprocessor 2 is assigned CPUID 0x02 and adds numbers from 0x20 to 0x27 and so on. The benefits of defining CPUID and a simple summation are that the effort during the visual inspection of waveforms is reduced and the exchange of data, for the sake of communication and synchronization, can be very easily identified. The final result of the application is the addition of all the individual additions conducted locally by each microprocessor. From a high-level language programming environment, this application can be seen as a long summation that is partitioned into a number of equal parts and that the number of partitions are allocated to an equal number of microprocessors. The

mechanisms of synchronization or communication will allow the obtention of the total result of the summation (Culler et al., 1998).

Depending on the type of the architecture, a specific interprocessor communication mechanism is to allow the microprocessors to share their local results and a global variable is to account for the addition of the local results. The global variable may come in the form of a shared resource or an internal register of one of the microprocessors.

4.1 Mutex

As has been previously discussed, in a shared-memory architecture the mutex must provide a port of communication with each microprocessor present in the system. For the demonstration of this architecture, a system with eight PicoBlaze instances is evaluated. Figure 8 shows the organization between the eight PicoBlaze microprocessors, the shared resource and the mutex. Note that, due to the type application to be deployed in the architecture, the communication between the shared resource and the microprocessors is bidirectional. The shared resource is a multiport register that will hold the final result of the additions.

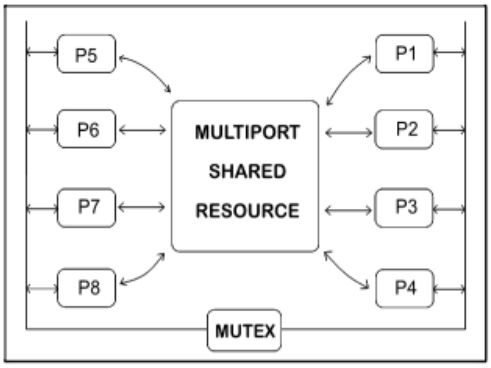


Fig. 8. MPSoC with a mutex and eight PicoBlazes

The VHDL entity of a mutex with ports for the eight PicoBlazes is shown below. Note that the ports and flags of PicoBlaze are encapsulated into a bus with the purpose of reducing the effort of the structural design of the whole system.

```
entity mutex_8 is
    Port ( CLK : in  STD_LOGIC;
           RST : in  STD_LOGIC;
           P1_BUS : in  STD_LOGIC_VECTOR (17 downto 0);
           P2_BUS : in  STD_LOGIC_VECTOR (17 downto 0);
           P3_BUS : in  STD_LOGIC_VECTOR (17 downto 0);
           P4_BUS : in  STD_LOGIC_VECTOR (17 downto 0);
           P5_BUS : in  STD_LOGIC_VECTOR (17 downto 0);
           P6_BUS : in  STD_LOGIC_VECTOR (17 downto 0);
           P7_BUS : in  STD_LOGIC_VECTOR (17 downto 0);
```

```
            P8_BUS : in  STD_LOGIC_VECTOR (17 downto 0);
            CPUID : out  STD_LOGIC_VECTOR (7 downto 0));
end mutex_8;
```

Each Px_BUS, where x stands for 1, 2, etc., is decomposed into the following signals

- Px_BUS(17) carries the request of the microprocessor: the LOCK or the UNLOCK command
- Px_BUS(16:10) carry the CPU ID of the requesting microprocessor
- Px_BUS(9:2) carry the address of the mutex in the bus system
- Px_BUS(1) and Px_BUS(0) carry the strobes that enable the communication

As can be seen in its entity, the mutex also provides an output port for all the microprocessors to read back the CPUID of the microprocessor that currently owns the shared resource; this reading back usually takes place after a microprocessor has tried to lock the mutex. The assembly-language code shown below is the routine that every microprocessor must execute to lock the mutex; the execution of the application will not continue until the microprocessor succeeds in locking the mutex. Note that PicoBlaze uses the instruction `OUTPUT` to write a data to an output port and the instruction `INPUT` to read a data from an input port. On the other hand, the combination of instructions `COMPARE` and `JUMP NZ` allows PicoBlaze to test a data (Xilinx, 2011a).

```
;requests access to the shared resource
LOAD    OutReg, CPUID
OR      OutReg, Lock ;CPU tries to lock the mutex
locked:
OUTPUT  OutReg, MutexAddress
INPUT   InReg, MutexAddress
COMPARE InReg, CPUID
JUMP    NZ, locked ;if not locked, tries it again
```

Upon succeeding locking the mutex, the microprocessor reads the multiport register, which for this example implements the global variable, executes another addition between the local and the global variable and saves the result into the global variable by writing back to the register. The VHDL entity of the multiport register is shown below.

```
entity mp_register_16 is
    Port ( CLK : in  STD_LOGIC;
           RST : in  STD_LOGIC;
           P1_BUS : in  STD_LOGIC_VECTOR (17 downto 0);
           P2_BUS : in  STD_LOGIC_VECTOR (17 downto 0);
           P3_BUS : in  STD_LOGIC_VECTOR (17 downto 0);
           P4_BUS : in  STD_LOGIC_VECTOR (17 downto 0);
           P5_BUS : in  STD_LOGIC_VECTOR (17 downto 0);
           P6_BUS : in  STD_LOGIC_VECTOR (17 downto 0);
           P7_BUS : in  STD_LOGIC_VECTOR (17 downto 0);
```

```
            P8_BUS : in  STD_LOGIC_VECTOR (17 downto 0);
            REG_OUT : out  STD_LOGIC_VECTOR (7 downto 0));
end mp_register_16;
```

The assembly-language routine for accessing the multiport register is shown below. Note that the local variables are `LowByte` and `HighByte`, while `ReadRegLow` and `ReadRegHi` are the addresses for a reading access to the register and `WriteRegLow` and `WriteRegHi` are the addresses for a writing access to the register.

```
INPUT   Acc, ReadRegLow          ;reads the global low byte and
ADD     LowByte, Acc             ;adds it to the local low byte
INPUT   Acc, ReadRegHi           ;reads the global high byte and
ADDCY   HighByte, Acc            ;adds it to the local high byte
OUTPUT  LowByte, WriteRegLow     ;the shared resource is updated
OUTPUT  HighByte, WriteRegHi     ;accordingly
LOAD    OutReg, CPUID
OR      OutReg, Unlock
OUTPUT  OutReg, MutexAddress     ;CPU unlocks the shared resource
```

Figure 9 shows a typical situation when all the microprocessors contend for the shared resource. Note that each microprocessor is represented by their lock command and CPUID. As the microprocessor with CPUID 0x01, which is that with the highest priority, is contending, the mutex resolves the contention by granting the access to this microprocessor. The signal *mutex_debug*, which is provided for debugging purposes, accounts for this situation. It can be verified that the contention is resolved by the mutex in four clock cycles.

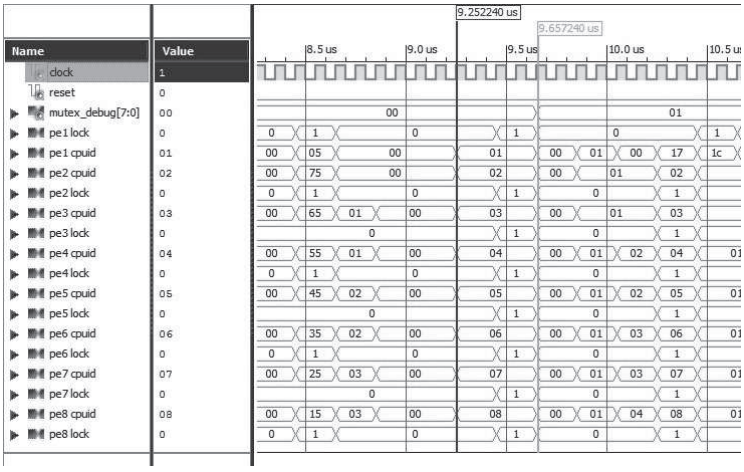


Fig. 9. The eight microprocessors contend for the shared resource and the mutex resolves the contention and grants access to microprocessor with CPUID 0x01

Figure 10 shows that the interval of time that the microprocessor with CPUID 01 owns the mutex is 24 clock cycles. The signal *mutex_debug* provides this information. Knowing that

each instruction of PicoBlaze takes two clock cycles, each microprocessor own the mutex for 12 intruction cycles. This number of instruction cycles was expected, as can be verified in the code excerpt that was shown previously. As the application that every microprocessor is running is exactly the same, all the microprocessors own the mutex for the same period of time.

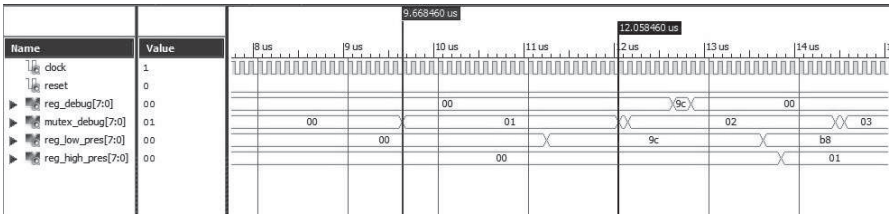


Fig. 10. Microprocessor with CPUID 0x01 owns the shared resource for 24 clock cycles

Figure 11 shows the moment when the PicoBlaze instance with the lowest priority, CPUID 0x08, accesses the multiport register, signals *reg_low_pres* and *reg_high_pres*, and updates its current contents. Note also that the completion of the work is indicated once this microprocessor unlocks the mutex, which happens at the time 28850 ns, approximately. Given a simulated clock period of 100 ns, it can be concluded that the task of the system is completed in an equivalent time to 289 instruction cycles.

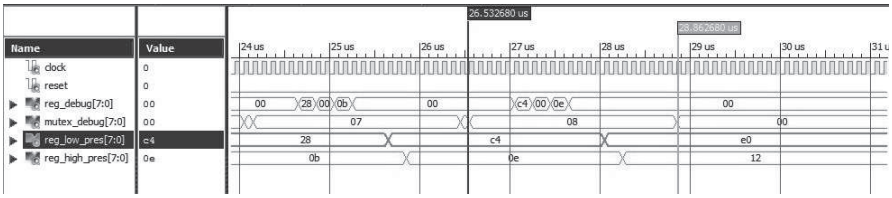


Fig. 11. The application is completed once the microprocessor with CPUID 0x08 unlocks the mutex

The main disadvantage of the use of a mutex in an MPSoC architecture is that the microprocessor with the lowest priority will have a latency to access the shared resource directly proportional to the number of microprocessors present in the architecture. The main advantage of this architecture is that the final result of the application is kept in just one memory location.

In terms of scalability, the number of states of the arbiter is dependant of the number of microprocessors of the architecture. If more microprocessors are to be added, more states are to be added to the arbiter. Additionally, the structure of the arbiter must be replicated as many times as shared resources present in the MPSoC. It can be concluded that the area of the communication components is linearly proportional to the number of microprocessors and the number of shared resources.

4.2 Mailbox

The structure of a mailbox composed by two FIFOs is shown in Figure 12. The mailbox features two ports: Port 0 and Port 1. The microprocessor attached to Port 0 must make use

of the addresses 0xB0 through 0xB4 for the communication with the mailbox, for a writing access the data must be written to FIFO 0 and for a reading access the data must be read from FIFO 1. The microprocessor attached to Port 1 must make use of the addresses 0xC0 through 0xC4 for the communication with the mailbox; for a writing access the data must be written to FIFO 1 and for a reading access the data must be read from FIFO 0.

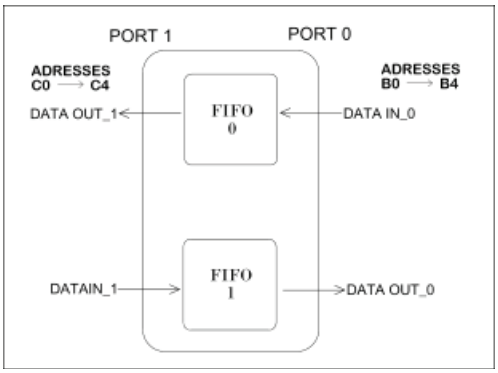


Fig. 12. Structure of a mailbox composed by two FIFOs

The labels given to the two ports of each FIFO are provided for easy identification when two PicoBlazes are attached. The definition of the entity of the mailbox is shown below. Note that the design of the mailbox is generic in terms of the depth of the two FIFOs and the number of bits of the memory locations. In addition, the port that was previously defined as Port 0 now is identified as the right port, while the port that was previously defined as Port 1 is now identified as the left port. Finally, the definition of the elements of each port are defined in accordance to the ports of PicoBlaze.

```
entity mailbox is
   generic(
        NBIT : integer := 8; --number of bits
        WIDTH : integer := 3 --addressing lines
        );
   Port( CLK, RST : in  STD_LOGIC;
        --right port
        DATAIN_0 : in  STD_LOGIC_VECTOR(7 downto 0);
        PORTID_0 : in  STD_LOGIC_VECTOR(7 downto 0);
        RD_0, WR_0 : in  STD_LOGIC;
        DATAOUT_0 : out  STD_LOGIC_VECTOR(7 downto 0);
        --left port
        DATAIN_1 : in  STD_LOGIC_VECTOR(7 downto 0);
        PORTID_1 : in  STD_LOGIC_VECTOR(7 downto 0);
        RD_1, WR_1 : in  STD_LOGIC;
        DATAOUT_1 : out  STD_LOGIC_VECTOR(7 downto 0));
end mailbox;
```

Figure 13 shows a multiprocessor system with nine PicoBlaze instances. The PicoBlaze instances are numbered from P1 to P9 in a typical tree organization (Bjerregaard & Mahadevan, 2006). The arrowheads between the microprocessors represent the mailboxes establishing the interprocessor communication channel. The mailboxes are numbered from 1 to 8.

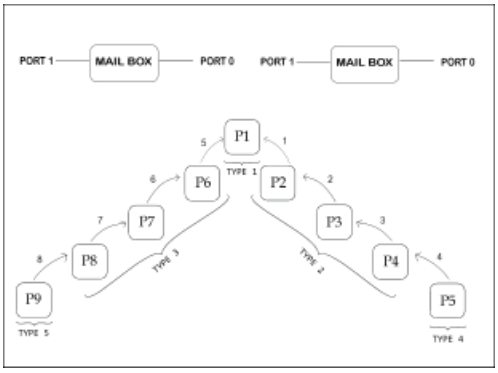


Fig. 13. Message passing architecture with nine PicoBlazes and eigth mailboxes in a tree structure

While the application is exactly the same for all the microprocessors, they do not implement the same functionality in the three organizations; this is the reason why a TYPE feature is also defined for each microprocessor. As a consequence, while all the PicoBlaze instances run the same application, they differ in the CPUID and TYPE that they are assigned. The execution of the application according to the definition of TYPE is explained next. Note that the TYPE-1 microprocessor has the longest execution.

- TYPE-1 microprocessor is in charge of holding the total result of all the individual additions. Once it concludes its local addition it starts to read the mailboxes and then update their local addition accordingly. The application is complete once both mailboxes indicate that are empty.
- TYPE-2 microprocessors write the result of their local addition to the mailbox to the left (Port 0 of mailbox) and then start to read data from the mailbox to the right (Port 1 of mailbox) and writing it to the mailbox to the left (Port 1 of mailbox) until the mailbox to the right indicates that it is empty, thus reaching an end to their execution.
- TYPE-3 microprocessors follow the same execution as TYPE-2 microprocessor, except that in the opposite direction. TYPE-3 microprocessors complete their local addition and write their result to the mailbox to the right (Port 1 of mailbox) and then start to read data from the mailbox to the left (Port 0 of mailbox) and writing it to the mailbox to the right (Port 1 of mailbox) until the mailbox to the left indicates that it is empty, thus reaching an end to their execution.
- TYPE-4 microprocessor completes its local addition and then it writes it to the mailbox to the left (Port 0 of mailbox), thus completing its job
- TYPE-5 microprocessor completes its local addition and then it writes it to the mailbox to the right (Port 1 of mailbox), thus completing its job

The assembly-language directives for PicoBlaze shown below provide the necessary functionality for the definition of the CPUID and the TYPE. Note that the CPUID definition is no longer necessary for the organization of the architecture.

```
;constants and registers
CONSTANT CPUID, 01       ;unique identifier of the CPU
CONSTANT TYPE, 01        ;classification in the architecture
```

As part of the assembly-language application of the system, a decoding subroutine is provided for each microprocessor to branch to their corresponding thread of execution according to the TYPE they were assigned. The code excerpt below shows this subroutine. Again, the combination of the instructions `COMPARE` and `JUMP Z` facilitates the required branching.

```
;the type of communication is determined according
;to the TYPE assigned to each CPU
;type 1 is the last to initiate in order to allow
;the fifos to be loaded at its both sides
LOAD     InReg, TYPE
COMPARE  InReg, 04
JUMP     Z, commType4
COMPARE  InReg, 05
JUMP     Z, commType5
COMPARE  InReg, 02
JUMP     Z, commType2
COMPARE  InReg, 03
JUMP     Z, commType3
JUMP     commType1
```

Subroutines for the TYPE-2 and TYPE-4 microprocessors are shown below. Both subroutines make use of a set of routines that implement even lower-level communication between PicoBlaze and the mailboxes. Note that in both subroutines a `JUMP` to the address `loop` represents the completion of the job.

```
commType2:
    ;this type writes to port 0 and reads from port 1
    LOAD    FirstData, LowByte
    LOAD    SecondData, HighByte
    CALL    writeAccessPort0       ;shares local variables
askIfEmptyType2:
    INPUT   InReg, ReadEmptyPort1  ;asks if fifo is empty
    COMPARE InReg, isFifoEmpty
    JUMP    Z, loop             ;if fifo is empty job is done
    CALL    readAccessPort1     ;if fifo is not empty it passes
    CALL    writeAccessPort0    ;a new pair of data
    JUMP    askIfEmptyType2
```

```
commType4:
    LOAD    FirstData, LowByte
    LOAD    SecondData, HighByte
    CALL    writeAccessPort0     ;shares local variables
    JUMP    loop
```

Figure 14 shows how the TYPE-4 microprocessor writes its local variables, two 8-bit variables that hold the result of its local addition, to a mailbox. Note the use of the address 0xB0 by the port portid_pe5 and the two short pulses of the port writest_pe5 to indicate a writing access to the Port 0 of the mailbox.

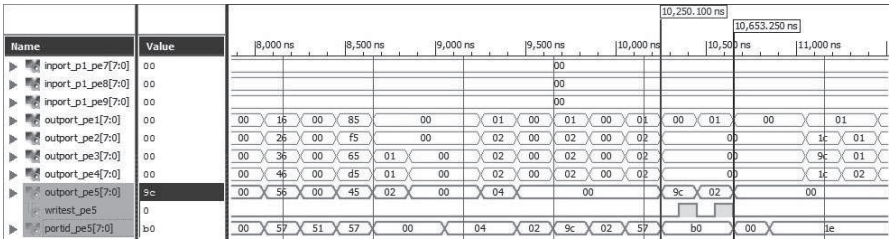


Fig. 14. Communication of the TYPE-4 microprocessor with a mailbox

Figure 15 shows the communication of the microprocessor P2, which is a TYPE-2, with its both mailboxes. The cursor indicates the beginning of such communication. This microprocessor starts by writing its local addition variables to the mailbox to the left, note the use of the address 0xB0 and the two pulses in writest_pe2. Next, the microprocessor reads the empty flag of the mailbox to the right, note the single pulse of readst_pe2; the flag is reflected in inport_p1_pe2 (Port 1 of the mailbox to the right). As the mailbox indicates that it is not empty, the microprocessor proceeds to read the mailbox in four read accesses (note that the addresses for these accesses are 0xC1, 0xC4, 0xC1 and 0xC4); the data read is verified in inport_p1_pe2, which is then written to Port 0 of the mailbox to the left, the port output_pe2 transports the data.

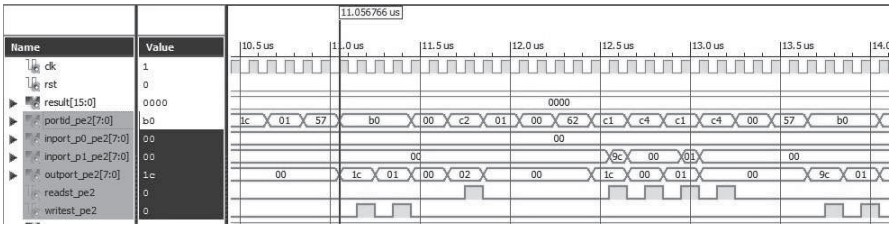


Fig. 15. Communication of a TYPE-2 microprocessor with its mailboxes

Figure 16 shows the operation of the microprocessor P1, the only TYPE 1 in the architecture, before completing its job. The cursor coincides with moment the microprocessor reads the empty flag of the mailbox attached to the right. As can be seen, the port inport_p1_pe1 carries the value 0x01 indicating that the mailbox is empty. This also indicates that the microprocessor has completed the processing of all the data generated by the microprocessors to the right (P2, P3, P4 and P5). Next, microprocessor P1 reads a new data from the mailbox attached to

left through the port inport_p0_pe1 (the data is 0x040C) issuing the address sequence: 0xB1, 0xB4, 0xB1 and 0xB4. Around the time 32 microseconds, microprocessor P1 reads the empty flag of the mailbox to the left; as the value in port inport_p0_pe1 is 0x01, microprocessor P1 is indicated that the mailbox to the left is empty. This leads microprocessor P1 to write the final result of all the additions to an external register for debugging purposes; this takes place around the time 32.5 microseconds and the data written is 0x177C. Note that the register is mapped to the addresses 0xE1 and 0xE2.

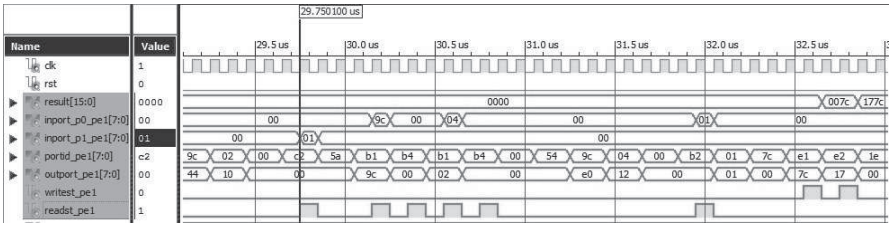


Fig. 16. Communication of the TYPE-1 microprocessor with its mailboxes and the completion of the application

Due to the tree structure of the MPSoC and the point-to-point communication implemented by the mailboxes, the messages take a long path, and, as a result, a long latency, to reach the microprocessor at the top of the tree for them to be processed. It can be demonstrated that the time it takes for a message to be transferred from one mailbox to the other equals eigth instruction cycles. This number can be used for the calculation of the total latency of the system given the number of mailboxes present. The time a message takes from its source to its destination can be computed as the number of mailboxes in the path times eight instruction cycles. In order to avoid that the latency of the messages adds up, the messages can be parallelized through the branches of the tree.

In terms of scalability, the message-passing architecture is simple. For each point-to-point communication between two microprocessors, one mailbox is needed. In addition, for one more microprocessor that is added to the architecture, at least one more mailbox needs to be introduced. While the communication mechanism in a message-passing architecture is not complex, its main disadvantage is the area required by each mailbox. As each mailbox is made up of two FIFOs and a FIFO is made up of registers, the requirement of memory for a large number of mailboxes in an architecture can be high.

The latency and area penalties in a message-passing architectures can be reduced if a more flexible communication approach is devised. The next section explores the use of a router for the implementation of a multipoint communication network.

4.3 NoC

Several NoC architectures have been explored and successfully implemented (De Micheli & Benini, 2006), (Bjerregaard & Mahadevan, 2006); however, in line with the proposal of this chapter, the design of a specific router has been chosen. This router is adequate for the FPGA fabric and its functionality has already been reported in VHDL (Bakr, 2008), see Figure 17. It is worth noting that the VHDL description has been modified to support the communication needs of PicoBlaze.

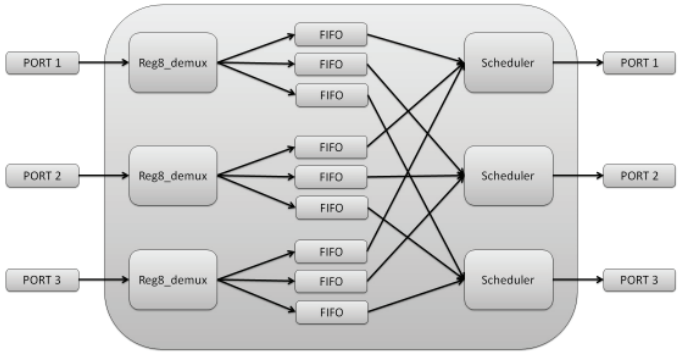


Fig. 17. Structure of an NoC router adequate for FPGA implementation

The router shown in Figure 17 allows the communication of up to three PEs. The router requires that each PE features separate input and output ports. The operation of the router is as follows. The incoming messages, at the left-hand side ports, are inspected and their header extracted in order to determine their destination. This inspection is conducted by the joint operation, at each port, of a register and a demultiplexor. Once the destination is identified, the messages are stored in the corresponding FIFO; the number of the internal FIFOs is determined by the number of ports of the router. From the FIFOs, the messages are finally routed to their destination based on the operation of a round-robin scheduler.

In order to take advantage of the 8-bit communication data of PicoBlaze, each packet of communication between two microprocessors is composed of two 8-bit data. The first 8-bit data, the address, identifies the destination of the message; this address data is coded as one-hot code, thus allowing up to eight microprocessors to be addressed in the architecture. The second 8-bit data is the message itself. For purposes of evaluation, eigth PicoBlaze instances are connected to the router; as a result, the router is composed of 8 register-demultiplexer modules, 64 4-level FIFOs and 8 schedulers. Similar to the implementation of the other two multiprocessor architectures evaluated in the chapter, each microprocessor is to implement the addition of eight 8-bit data and the goal is to exploit parallelism to produce the addition of all the individual results. Figure 18 shows the

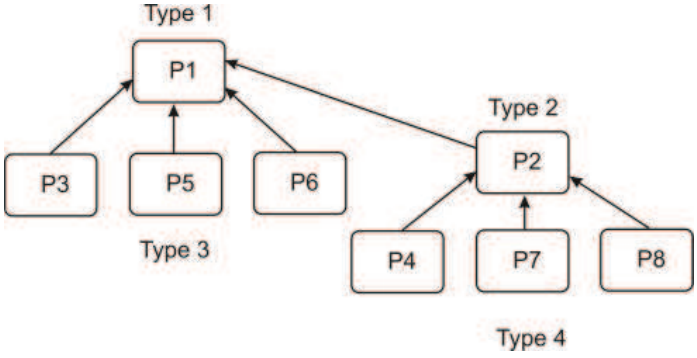


Fig. 18. Communication of eight microprocessors making use of an NoC

communication between eight microprocessors. Note that the eight microprocessors have been classified into four types in order to define the data flow that will allow the computation of the final result.

Along with the two bytes that hold the result of their local additions, the microprocessors issue the address of the destination microprocessors. The router extracts the address from the communication packet and stores the data in internal FIFOs. The arrival of new data to the FIFOs is indicated to the destination microprocessor in the form of an interrupt. The code excerpt below shows the entity of the router for the eight microprocessors. Note that only the ports of two microprocessors are shown for the sake of brevity.

```
entity noc_router is
    Port ( CLK : in  STD_LOGIC;
           RST : in  STD_LOGIC;
           INPORT1    : in  STD_LOGIC_VECTOR (7 downto 0);
           PORTID1    : in  STD_LOGIC_VECTOR (7 downto 0);
           WRITE1     : in  STD_LOGIC;
           INPORT2    : in  STD_LOGIC_VECTOR (7 downto 0);
           PORTID2    : in  STD_LOGIC_VECTOR (7 downto 0);
           WRITE2     : in  STD_LOGIC;
     ...
           OUTPORT1   : out  STD_LOGIC_VECTOR (7 downto 0);
           INTERRUPT1 : out  STD_LOGIC;
           OUTPORT2   : out  STD_LOGIC_VECTOR (7 downto 0);
           INTERRUPT2 : out  STD_LOGIC;
     ...
     );
end noc_router;
```

The differences between the four types of microprocessors in the NoC architecture are explained next. Once they complete their local additions, type-3 and type-4 microprocessors share their results with their destination microprocessor. As a consequence, these two types of microprocessors do not support interrupts. Upon the occurrence of an interrupt, type-1 microprocessor updates its local result with the new data. Type-2 microprocessor updates its local result upon the arrival of new data and waits for the occurrence of all the interrupts to share its updated local result with type-1 microprocessor. The assembly-language application that the microprocessors execute include the CPUID and the type of each; this is demonstrated in the code excerpt shown below.

```
;constants and registers
CONSTANT CPUID, 01      ;unique identifier of the CPU
CONSTANT TYPE, 01       ;classification in the architecture
```

The assembly-language routine that allows the microprocessors to share their local result is shown below.

```
;type_2 retransmits local values to cpu1,
;data first, address second
;until interrupt counter counts six interrupts
;(low and high byte from three cpus)
type_2:
    COMPARE  InterruptCounter, 06
    JUMP     NZ, type_2
    LOAD     DataReg, LowByte
    OUTPUT   DataReg, RouterWriteData
    LOAD     AddressReg, CPU1
    OUTPUT   AddressReg, RouterWriteAddress
    LOAD     DataReg, HighByte
    OUTPUT   DataReg, RouterWriteData
    OUTPUT   AddressReg, RouterWriteAddress
    JUMP     loop
    ;
    ;type_3 retransmits local values to cpu1,
    ;data first, address second
type_3:
    LOAD     DataReg, LowByte
    OUTPUT   DataReg, RouterWriteData
    LOAD     AddressReg, CPU1
    OUTPUT   AddressReg, RouterWriteAddress
    LOAD     DataReg, HighByte
    OUTPUT   DataReg, RouterWriteData
    OUTPUT   AddressReg, RouterWriteAddress
    JUMP     loop
```

As has been discussed before, interrupts play a key role for the communication of messages in the architecture. A simple routine that handles interrupts and allows the microprocessors to update their local results is shown below.

```
;depending on the CPU the data are retransmitted or not
;the interrupt flag of router is cleared after data is read
isr:
    ;data in router is read
    INPUT   Temporal, RouterReadData
    ;verifies the flag to proceed with the addition
    COMPARE OrderByteFlag, 01
    JUMP    Z, add_high_byte
    ADD     LowByte, Temporal
    ;next interrupt addition of high byte will take place
    LOAD    OrderByteFlag, 01
isr_continuation:
    ;this counter only applies to cpu type 2
    ADD     InterruptCounter, 01
```

```
    LOAD    Temporal, 00
    ;interrupt flag of router is cleared
    OUTPUT  Temporal, RouterClearFlag
    RETURNI ENABLE
    ;
add_high_byte:
    ADDCY   HighByte, Temporal
    ;next interrupt addition of low byte will take place
    LOAD    OrderByteFlag, 00
    JUMP    isr_continuation
;
```

Figure 19 shows low-level details of the communication between type-3 and type-1 microprocessors. The signals *out_port*, *port_id* and *writest_strobe* of the PicoBlaze instances P3, P5 and P6 allow sending the two bytes of their local additions to PicoBlaze P1; note that the three messages are composed of four write accesses: the number 0xB0 in port_id identifies the data, while the number 0xB1 identifies the CPUID of the receiver. It can be seen that the three writes accesses complete around 12.05 microseconds of simulation time. The bottom half of Figure 19 shows the arrival of the data shared by P3 and P5 (first the low byte first and then the high byte); the completion of the two messages occur at 14.86 and 20.06 microseconds.

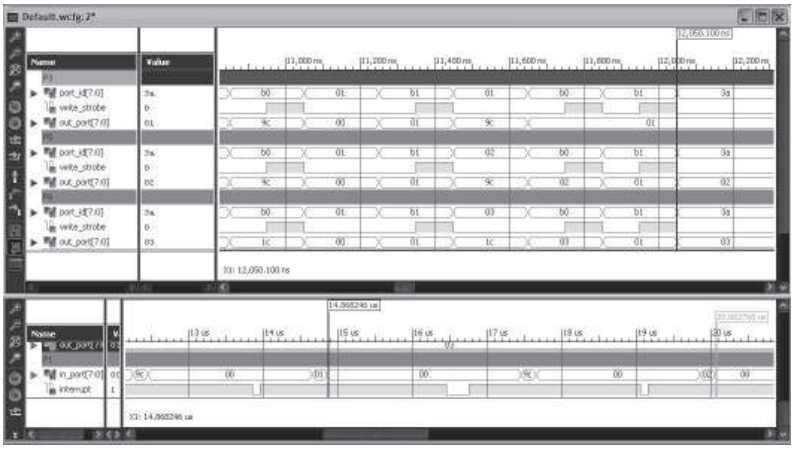


Fig. 19. Communication between type-3 and type-1 microprocessors

Details about the communication between a type-4 and a type-2 microprocessor are shown in Figure 20. For this communication scenario, P4, P7 and P8 are the senders and P2 is the receiver. The three messages are completed around 11.65 microseconds, while the messages from P7 and P8 are received at 14.44 and 19.45 microseconds.

Communication between P2, the sender, and P1, the receiver, is shown in Figure 21. The number sent by P2 is 0x0AF0, which accounts for the local additions of P2, P4, P7 and P8. As indicated previously, P1 is to hold the result of the eight local additions. Note that the latency in this communication is of around 3.39 microseconds.

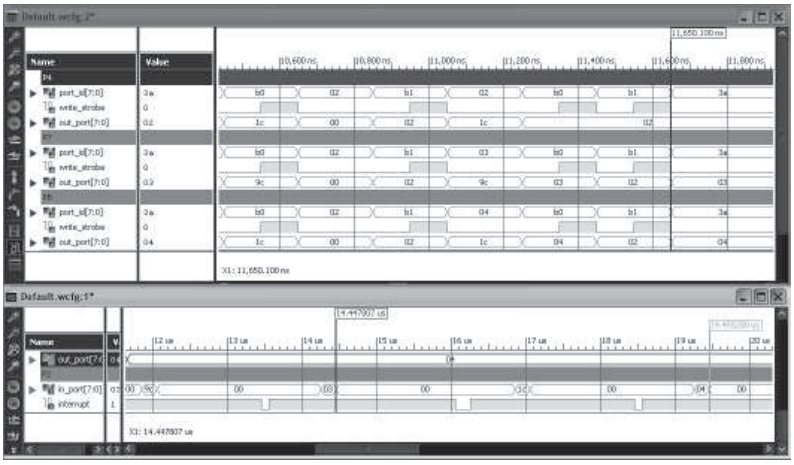


Fig. 20. Communication between type-4 and type-2 microprocessors

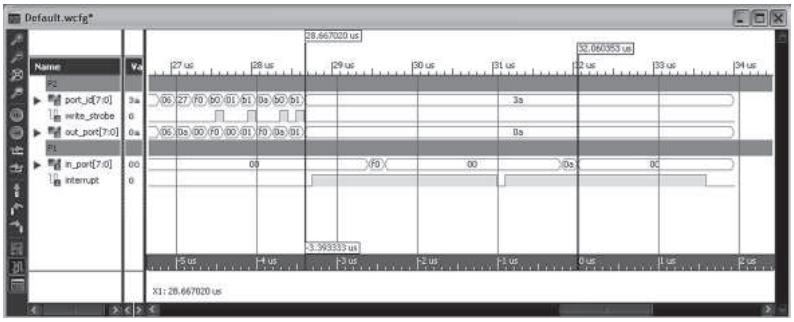


Fig. 21. Communication between type-2 and type-1 microprocessors

It can be demonstrated that the latency in the communication between a sender and a receiver in the NoC has two sources: the inherent operation of the router and the ISR latency of PicoBlaze. Inside the router, the joint operation of the register and demultiplexer incurs in a delay that is mostly combinational; however, the scheduler consists in a 32-state FSM (4 states for each microprocessor operation), whose operation is influenced by the time PicoBlaze takes to service an interrupt. In a worst-case scenario, the ISR takes 20 instruction cycles (10 for each byte of the local addition that is sent), which equal 40 clock cycles; adding 4 clock cycles for the operation of the FSM for each microprocessor and multiplying by the 8 microprocessors in the system, it results in a 352-clock-cycles delay. The simulation that is being reported is based on a period of 100 nanoseconds.

In terms of the hardware provisions of the NoC, the depth of the FIFOs should reflect the needs of the application, as the number of FIFOs present in the router is exponentially dependant of the number of microprocessors in the architecture. Despite the high demands for FIFOs, an NoC offers the most flexible approach for the communication and synchronization of multiple microprocessors. Indeed, the sender has control over the selection of the microprocessor that is to receive the message and, depending on the existing traffic conditions of the system, the

sender may intelligently define new routes of communication to comply with hard deadlines of communication.

5. Conclusions and future work

Multiprocessor system-on-chip (MPSoC) is the last incarnation of complex system-on-chip. The market push for digital convergence and multimedia systems has put a lot pressure on the capabilities of integrated circuits and the only possible way designers can take up the challenge is to deploy a number of microprocessors in a single chip. This new era in the VLSI design is possible thanks to the latest deep submicron (DSM) technology, as DSM technologies allow a larger number of transistors and more processing available per area. As the design and programming of microprocessors have currently reached a level of maturity, the design focus of current SoCs is on the on-chip communication. On-chip communication defines the structure of communication and synchronization between all the microprocessors present in the architecture.

This chapter has intended to shed some light into the low-level details of communication and synchronization between a number of microprocessors in the architecture. For that purpose, a design platform has been proposed; this platform is composed of the tool ISE and the microprocessor PicoBlaze, both from Xilinx, and FPGAs for prototyping and evaluation. Likewise, three multiprocessor architectures have been proposed for evaluation: memory shared, message passing and Network-on-Chip (NoC). The communication and synchronization components necessary in each architecture have been described in VHDL and the software applications have been developed in assembly language. Programming of the applications at low level has allowed a deep inspection into the mechanisms surronding on-chip communication. Additionally, the low-level approach has served to identify aspects of latency and scalibility in the three architectures.

From the exploration and evaluation of the three architectures, it can now be concluded that NoC has the most promising results. This is not surprising, as NoC design has received a great deal of attention from several research groups around the globe. Among the most important characteristics of NoC design are: parallel architectures are naturally implemented; multipoint communication between microprocessors increases the design space, leading to well-explored architectures and high-performance applications and the on-chip communication structure is now separated from the computation structure implemented by the microprocessor instances.

NoC design is an open research area that still requires a great deal of exploration and evaluation. With the aid of FPGAs, NoC designs can be prototyped and evaluated early in the design flow. However, in order to get the most out of NoCs, more aspects need to be considered in this design area. A variety of microprocessors, parallel programming, networking standards, operating systems, high-level programming languages, exploration of data-intensive applications, among others, need to be considered as part of the design space that NoC brings about. In particular, automatic deployment of an application on a given number of microprocessor instances and that the process of deployment be aware of the network of communication in the architecture are still under research. It is the author's belief that the use of high-level design languages, such as SystemC, can be of great help to NoC design when it comes to predicting performance, modelling behaviour and functionality, predicting network trafic and, finally, mapping the application efficiently on the architecture.

Additionally, by making use of the support of SystemC for higher levels of abstraction, the design time and the exploration of the design space of NoCs can be drastically reduced.

6. Acknowledgements

The author would like to thank the following institutions and people: Xilinx Inc. for the donation of their tools for the preparation of this work, IMEC and the Microelectronics Training Center for sharing the design of the NoC router and the students Mr. D. Sosa-Ceron, Mr. R. Ramirez-Mata, Mr. P. Galicia-Rojas, Mr. R. Garcia-Baez and Mr. L. Cortez-Eliosa at the Tecnologico de Monterrey Campus Puebla for the elaboration of the images of this chapter.

7. References

Antonio-Torres, D.; Villanueva-Perez, D.; Sanchez-Canepa, E.; Segura-Meraz, N.; Garcia-Garcia, D.; Conchouso-Gonzalez, D.; Miranda-Vergara, J.; Gonzalez-Herrera, J.; Riquer-Martinez de Ita, A.; Hernandez-Rodriguez, B.; Castaneda-Espinosa de los Monteros, R.; Garcia-Chavez, F.; Tellez-Rojas, V. & Bautista-Hernandez, A. (2009). A PicoBlaze-Based Embedded System for Monitoring Applications, *CONIELECOMP '09 Proceedings of the 2009 International Conference on Electrical, Communications, and Computers*, 978-0-7695-3587-6, Cholula, Puebla, Mexico, 2009

Asokan, V. (2007). Designing Multiprocessor Systems in Platform Studio, White Paper 262 (WP262), Xilinx Inc., USA, 2007

Bakr, S. (2008). Network-on-Chip Router, *Lab Exercise no. 29*, IMEC, Microelectronics Training Center, Belgium, 2008

Bjerregaard, T. & Mahadevan, S. (2006). A Survey of Research and Practices on Network-on-Chip. *ACM Computing Surveys*, Vol. 38, March 2006, ACM

Brown, S. & Vranesic, Z. (2000). *Fundamentals of Digital Logic with VHDL Design*, McGraw-Hill, 0070125910, USA

Chang, H.; Cooke, L.; Hunt, M.; Martin, G.; McNelly, A. & Todd, L. (1999). *Surviving the SoC Revolution: A Guide to Platform-Based Design*, Kluwer Academic Publisher, The Netherlands

Culler, D.; Singh, J. & Gupta, A. (1998). *Parallel Computer Architecture: A Hardware/Software Approach*, Morgan Kaufmann Publisher, 1558603433, USA

De Micheli, G. & Benini, L. (2006). *Networks on Chips*, Morgan Kaufmann Publisher, 0123705211, USA

Jerraya, A. & Wolf, W. (2005).*Multiprocessor Systems-On-Chips*, Ahmed Amin Jerraya & Wayne Wolf, (Ed.), Morgan Kaufmann Publisher, 012385251X, USA

Kowalczyk, J. Multiprocessor Systems. White Paper 162 (WP162), Xilinx Inc., USA

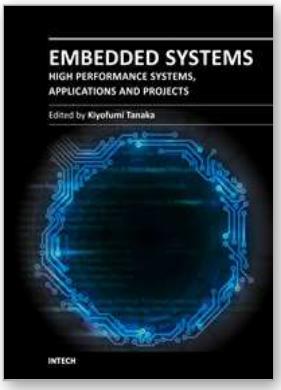
Li, X. & Hammami, O. (2009). An Automatic Design Flow for Data Parallel and Pipelined Signal Processing Applications on EmbeddedMultiprocessor with NoC: Application to Cryptography. *International Journal of Reconfigurable Computing*, Vol. 2009, Article ID 631490, September 2009

Maxfield, C. (2004). *The Design Warrior's Guide to FPGAs*, Newnes, 0-7506-7604-3, USA

pBlazeIDE Instruction Set Simulator, available at: http://www.mediatronix.com /pBlazeIDE.htm

Skahill, K. (1996). *VHDL for Programmable Logic*, Addison-Wesley Publishinh Company, Inc., 0201895862, USA

XPS Mailbox (v2.00a) Datasheet, Xilinx Inc., USA, 2009
XPS Mutex (v1.00c) Datasheet, Xilinx Inc., USA, 2009
PicoBlaze User Resources, available at: http://www.xilinx.com/ipcenter/processor_central/picoblaze/
Platform Studio and the Embedded Development Kit (EDK), available at: http://www. xilinx.com/tools/platform.htm
Xilinx ISE Design Suite, available at: http://www.xilinx.com/products/design-tools/ise-design-suite/
Yu, P. & Schaumont, P. (2006). Executing Hardware as Parallel Software for Picoblaze Networks, *International Conference on Field Programmable Logic and Applications, 2006. FPL '06*, 1-4244-0312-X, Madrid, August, 2006



Embedded Systems - High Performance Systems, Applications and Projects

Edited by Dr. Kiyofumi Tanaka

ISBN 978-953-51-0350-9
Hard cover, 278 pages
Publisher InTech
Published online 16, March, 2012
Published in print edition March, 2012

Nowadays, embedded systems - computer systems that are embedded in various kinds of devices and play an important role of specific control functions, have permeated various scenes of industry. Therefore, we can hardly discuss our life or society from now onwards without referring to embedded systems. For wide-ranging embedded systems to continue their growth, a number of high-quality fundamental and applied researches are indispensable. This book contains 13 excellent chapters and addresses a wide spectrum of research topics of embedded systems, including parallel computing, communication architecture, application-specific systems, and embedded systems projects. Embedded systems can be made only after fusing miscellaneous technologies together. Various technologies condensed in this book as well as in the complementary book "Embedded Systems - Theory and Design Methodology", will be helpful to researchers and engineers around the world.

InTech Europe
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

InTech China
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821