Multiprocessor System in an FPGA

Wilson José Mário P. Véstias Horácio C. Neto
INESC-ID/IST/UTL INESC-ID/ISEL/IPL INESC-ID/IST/UTL

Abstract

New applications are emerging more complex and demanding than ever, leading technology forward. In the embedded systems area, the MPSoC solution is being increasingly adopted by industry, as it enables systems to reach their real time deadlines and at the same time overcome area and power restrictions. In this respect, FPGAs emerge as a promising platform to develop these kinds of systems. FPGA-based multiprocessors present a cheaper and faster solution compared to ASIC-based multiprocessors.

In this project we designed, implemented and tested a homogenous soft-core open source based multiprocessing system in an FPGA considering two different communication architectures. A matrix multiplication algorithm has been parallelized and used to evaluate the system acceleration and efficiency.

1. Introduction

For many years, hardware engineers have relied in increasing the systems clock frequency as a mean to get more performance. However, this approach ceases to be viable as problems like heat sink and heat removal began to be extremely difficult to overcome. In the search for simpler ways of getting more performance, multiprocessing systems appear as an increasingly popular solution. As technology advanced, it began to be possible to integrate in a chip a complete multiprocessing system. These systems are called MPSoC (Multi Processor Systems on Chip).

Nowadays, MPSoC is an extremely attractive solution in the embedded systems area, as it allows embedded systems to meet their real time deadlines and at the same time overcome power consumption and area critical constrictions [1].

In this respect, FPGAs (Field Programmable Gate Array) emerge as a new and promising platform to implement multiprocessing systems. FPGAs enable fast prototyping and research of new architectures without ASIC (Application Specific Integrated Circuit) related problems. However, designing in HDL is very time consuming. An alternative to HDL design is the use of soft-core processors on an FPGA to build a multiprocessor system. Soft-core processors are configurable processors designed to fit well on an FPGA design. In today’s FPGAs it is possible to integrate dozens of processors and therefore to provide a significant parallel computation capacity.

This paper presents a multiprocessor architecture based on an open source soft-core processor and proposes two alternative communication architectures.

In section 2, the multiprocessing systems discussion is particularized for FPGAs. First, alternative soft-cores architectures are evaluated and the most adequate to this project is chosen. Then, some FPGA-based multiprocessor systems implementations are presented together with a brief architectural background discussion. Finally, communication architectures are introduced and alternative implementations are discussed. Section 3 describes the processor MB-LITE architecture. Section 4 presents the multiprocessing system architecture design and details the two communications architectures developed. Section 5 describes the test application for the system and the test methodology adopted. It also presents the synthesis and simulation results and discusses area/performance related issues. In section 6, a reflection is made about the work done and some suggestions to improve the work further are presented in section 7.

2. FPGA-based Multiprocessing Systems

Choosing the processing element of a multiprocessor system is a crucial step in the project as it is rather likely that it will limit the system in some way. Several soft-core were compared, including open source and commercial. In table 1 we present the main attributes of the soft-cores studied as potential options. In the end we chose the soft-core MB-LITE because its performance and resource consumption are close to the commercial soft-cores, such as the MicroBlaze, and it provides the same flexibility of his open source counter-parts.

2.1 Architecture Background

Normally the target application of the FPGA-based multiprocessing system determines the architecture.
Three main types of architectures can be defined [1]: 1) Master-Slave, where one or more processors act as the master processor, controlling the behavior of the slave processors; 2) Pipeline, where the architecture is composed by a chain of processors and each processor acts as pipeline stage; 3) Net architecture, where there is no hierarchy between processors and all processors are able to communicate with each other when necessary.

Another important issue is the way communications are established physically between the system elements [1]: 1) Point to point. The processors are directly connected. Provides a large bandwidth but is area inefficient; 2) Shared bus. It is the best known communication mechanism but is ineffective in terms of performance because the bus can only be used by one processor at a time; 3) NoC (Network-on-Chip). Uses small routers inside the chip to enable communications between all cores of the system with low latencies. For many on-chip cores, it is the solution that best combines area and performance.

Finally, we can define three methods to trade information between cores [1]: 1) Shared Memory is the most frequently used method. One of the mains reasons for this fact is the FPGAs limited amount of on-chip memory. 2) Message passing is mainly used in distributed memory systems and consists in sending messages with information between cores. Streaming consists in unidirectional communication from the sender to the receiver. One advantage is non-blocking communication.

2.2 Related Work

Multiprocessing systems are generally divided in two categories: heterogeneous systems and homogeneous systems. The first type involves systems where different processors or accelerators are part of the same system. In homogeneous systems all cores are identical.

Heterogeneous systems are by nature associated with application specific systems. The flexibility to have different cores in one system enables them to adapt better to the target application. These systems are used across several areas, like, bioinformatics [5], controllers [6], communication networks [7] and multimedia [8].

Although most of today’s MPSoC systems are heterogeneous, in order to meet the targeted application requirements, in the near future, homogeneous multiprocessor systems may become a viable alternative, bringing other benefits such as run-time load balancing and task migration [16]. The homogeneous architectural style is used generally for data-parallel systems. Wireless base stations, in which the same algorithm is applied to several independent data streams, are one example; motion estimation, in which different parts of the image can be treated separately, is another. Normally, homogeneous multiprocessor systems are general purpose, examples of these are in [9, 10].

2.3 Communication Architectures

Nowadays, communication architectures have to support an increasing number of processors connected. So, it is very important if an architecture scales well, providing a good performance/area ratio and the flexibility to integrate different cores with ease. In this respect, NoC architectures appear as a promising solution. NoC present a highly modular character and potentially a higher bandwidth depending on the adopted topology [9]. There are several articles exploring the implementation of NoC architectures [12, 13]. Systems extremely configurable and flexible are presented in [14] and [15]. Some articles compare NoC with other architectures. In [16], the authors perform a comparison between NoC based architectures with mesh topology and shared bus/memory architectures. Finally, in [17], a methodology to design NoCs is presented.

Clearly, a NoC based design will not always be the preferred solution for all kinds of applications. It is expected that NoC based designs will provide good solutions for flexible products that should be reconfigurable and programmable; for designs which are the basis for several product variants; for applications with a heterogeneous task mix; for applications with stringent time to market requirements; for products where reuse both at the block and the function and feature level is considered valuable [17].
3. Processor MB-LITE

The MB-LITE is a 32 bit RISC soft-core processor based on the MIPS organization with the same classic five pipeline stages architecture, IF (Instruction Fetch), ID (Instruction Decode), EX (Execute), MEM (Memory) e WB (Write-back), and compatible with the MicroBlaze ISA (Instruction Set Architecture). The IF stage feeds the pipeline with the required instruction and stores the current PC (Program Counter) [18]. The ID stage decodes the instructions in control signals which travel along with the instruction. The execution stage determines the ALU operands and the operation which needs to be executed. In addition to all the basic functionalities like shifts, additions and logical operations, the ALU can also optionally include a multiplier and a barrel shifter. The MEM module controls the interaction with data memory and the register bank is written in the WB stage.

The instructions in MB-LITE present the same latency specified for the MicroBlaze architecture. Most instructions - except for branches - have a latency of one cycle. Also the architecture provides a single interrupt like the MicroBlaze implementation.

MB-LITE implements distributed control in order to eliminate the need for a centralized and complex pipeline controller. All dependencies like stalls, hazards and forwards are solved locally. These dependencies can generate conflicts which are resolved using different techniques. Data conflicts are resolved by forwarding so that stalls are reduced to a minimum. The structural hazard which occurs when the same register is read and written concurrently is also solved using operand forwarding. When the result of a load instruction is immediately used, it is possible to forward the memory result either using stalls or additional logic. Finally, control hazards are solved using a pipeline flush.

To simplify connecting components like memories, co-processors, bridges and adapters, the MB-LITE authors provide an easily modifiable address decoder, which is used in our project design. The address decoder decodes the incoming addresses based on a generic memory map and forwards the control, address and data signals to the proper output.

4. Multiprocessing Architecture

The multiprocessing system designed is a homogenous system with distributed memory and communication through streaming. Various configurations of the same system were designed to study the best architectural options and to evaluate how the system scales with an increasing number of processing elements. The system is formed by two or more cores and a communication infrastructure which interconnects the cores. As connecting two processing elements is very simple, we have opted to implement a point to point communication to connect the cores in the two processor architecture. For the larger multiprocessors, two communication architectures have been designed. The first was a shared-media based communication architecture with a crossbar topology. The second was a NoC based communication architecture with a 2D-mesh topology. As a base to the system developed we designed a block which basically provides the processor with the means to communicate with other processors, see figure 1.

4.1 Base Block

The base block is composed by the following elements: processor MB-LITE, Data Instruction Memory (DIMEM), mailbox, address decoder and finally the data received controller. Figure 1 shows how these elements interconnect. The MB-LITE is the core processor and therefore the base block central module. All the components are memory mapped which means they can be accessed by the core through load and store instructions. This communication relies on the master/slave paradigm. The MB-LITE is the master and the other elements are the slaves. The address decoder is responsible for routing data and control signals between the MB-LITE and the other components. The DIMEM module represented in figure 1 is an instruction and data memory which can only be accessed by the processor MB-LITE. It presents itself as a unified memory, Von Neumann architecture. The Mailbox is a small memory where the data received from the other cores is stored. It can be read by the processor and written by the network. The received data controller is responsible for alerting the processor of data arrival. This module is basically a counter which increments each time a block of memory is written in the mailbox. This module can be reset by the processor. Finally, a port is mapped in memory to enable the sending of data by the processor to the network.

4.2 Switching Crossbar

The first decision made was to design a distributed memory system. This way, the traffic imposed on the network is reduced as each one of the processors communicates directly with his own memory. The communication is done by data streaming. This method avoids for one hand the memory conflicts originated from a memory shared approach, and on the other hand the latency associated to a message passing approach (e.g. blocking the master processor waiting for the slave response). The processors can then send data through the network to other processor by specifying the destiny and mailbox address where to write.
A processor can verify if data has been completely received by checking the associated counter. As said before, two communication architectures have been designed.

The switch crossbar network was chosen as the first architecture to design because it is simple to implement and enables simultaneous communication between the cores. Furthermore, for a low number of processors, the area overhead is small which favors FPGA mapping.

As can be seen in figure 2, the network is composed by two types of blocks: a router and a network adapter (NA). The router is responsible for routing the packages to their destiny. The network adapter controls the mailbox access and warns the processor when data is ready. As the network is composed by only one node the path taken by the packet equals its destiny. In the event of two packets pretending the same destiny the router chooses one based on an arbitration algorithm. We chose the round robin algorithm as it is very simple and easy to implement which means smaller and faster components. Basically the round robin algorithm follows an established order giving access alternately to a packet from a different source. To allocate a path for the packets a circuit switch approach was adopted, mainly because the target applications for the designed multiprocessor system involve matrix computations which benefit from this approach. Sending data from one processor to another can be divided in two steps. In the first step a header is sent. This packet has the necessary information to reserve a path for the forthcoming packets. Also, it has the base address and the block data size for writing all the data in the mailbox. Then, the following data which belongs to that header is sent in sequence and stored in that order in the Mailbox. Broadcast is also supported in this architecture. To implement broadcast an extra bit was added to the packets to identify a broadcast request. This request is then identified in the arbiter and has a priority above all the other requests in all arbiters.

4.3 NoC

The NoC architecture is composed by several nodes interconnected in a way according to a certain topology. For the NoC created, a mesh topology was chosen because it is simpler to implement and routing in a 2D architecture is easier resulting in potentially smaller and faster routers. Figure 3 shows the NoC with eight (eight processor cores + standard output controller core) cores. Each node in the network is formed by a network adapter and a router. The network adapter provides an interface between the processor and the network. It is responsible for data packing and retrieving the correct signals from the arriving packets to write in the mailbox. The router is responsible for relaying the packets through the network until they reach their destiny. The path a packet takes to reach its destination is defined in the source of the packet, more precisely in the network adapter. The network adapter contains a routing table with the path defined for each possible destination. The packet arbitration is again based on the round robin algorithm. As for the switching function we adopted the circuit switching approach again for the same reasons explained before. However, now we have more nodes so circuit switching is not done the same way.

The communication across the network is divided in three steps. First, the header is sent to reserve the resources necessary for the following packets to arrive at the correct destination. Then, data is sent sequentially and written in that way in the mailbox. The last packet has a different type which is identified by the circuit and allows freeing the resources reserved. The path resources across the crossbar, when reserved, stay blocked for packets from any other sources different from that of the header which reserved the path.

5. Results

In this section we describe the algorithm used to parallelize the test application and the methodology
5.1 Application

The application chosen to test the multiprocessing system was matrix multiplication. Matrix operations, like matrix multiplication, are very common appearing in almost all scientific research areas, as graph theory, numeric algorithms, signal processing and digital control [19]. For that reason, matrix multiplication presents a good generalization degree, and hence ideal to test the design system.

Given two matrices A and B, with n×m and m×q dimension respectively, the product matrix C with n×q dimension can be defined the following way:

\[ C_{ij} = \sum_{k=1}^{m} A_{ik}B_{kj} \quad \text{where, } 0 \leq i < n, 0 \leq j < q \]  

(1)

The matrix multiplication requires m×n×q multiplication operations and n×q×(m-1) add operations. A simple algorithm was adopted to design and test the system. The following assumptions were taken in the algorithm development. Matrices A and B are always square with dimension n×n, in which n is always power of 2. The number of processors, p, is also always power of 2. Finally, only the root processor or processor 0 has access to matrices A and B.

Let n = r * p, where r ≥ 1 is an integer. The matrix A is partitioned in p regions where each region contains r lines which are assigned to each processor. Matrix B is available to all the processors. The root processor acts as host processor responsible for distributing all the necessary data to the other processors and waiting for the results from the other processors. The distribution/collection of data depends on the implemented architecture. The generalization of the algorithm for p processors and n dimension matrices is simple. The original matrix B is divided in n×p dimension sub-matrices. Each processor is then responsible for n/p rows of matrix A and the submatrix n×p of B. The steps previously described when p=n are applied to these sub-matrices, and a n/p×p block is obtained from each processor which is part of matrix C. This process is repeated n/p times till the full matrix C is obtained.

5.2 Test Methodology

The systems were all synthesized using XST tool (Xilinx ISE 13.1) targeting a Virtex-6 device (XC6VLX130t-3ff1156). The MB-LITE cores were used without any optional functional unit or interrupt support. Each processor has 4 Kbytes of local memory and 512 bytes of mailbox.

To verify the system proper functioning, a C program was written and compiled with MB-GCC, MicroBlaze compiler (EDK 10.1). The program multiplies two 8×8 integer matrices. The program is loaded into the data instruction memory and a test bench is run, using ISim (Xilinx ISE 13.1) to observe the results. The input data is read from a RAM memory from which only the root processor has access.

The root processor is located in the central node in the NoC architecture, as can be seen looking at figure 4. The central node presents a globally shorter distance to reach all the other nodes, thus implying faster communication. The results were verified using the standard output interface (see node 8 in figure 4), which reads data from a bus and writes them to the ISim console. The results were checked by inspection.

Performance was measured by comparing execution times and by analyzing the speedup and efficiency reached by the system.

The Amdahl law relates the sequential and parallel part of an algorithm and gives us the maximum speedup possible for a target application. In (2) is presented the total execution time equation. The speedup and efficiency can be computed by equation (3) and (4), respectively. N refers to the parallelism degree (number of processors). S is the time taken to execute the code sequential part and P the time taken by a single processor system to execute the parallel part of the code.

\[ T(N) = S + P/N \]  

(2)

\[ Sp(N) = T(1)/T(N) \]  

(3)

\[ E(N) = Sp(N)/N \]  

(4)

These metrics allow evaluating the gain from parallelizing a certain application and running it on a multiprocessing system.

5.3 Crossbar Scalability

Looking at table 2, it can be seen that the area occupied by the systems grows almost linearly with the number of cores. However, the eight core system router has more than three times the size of the four...
cores system router (see number of LUTs, in table 3). The main reason for this discrepancy is the crossbar, whose occupied area grows quadratically with the number of ports.

The increase in number of processors does not significantly affect the maximum frequency achievable by systems, except for the eight core system. This is because the processor is, in general, the limiting factor of the system while the network allows higher clock frequencies. However, in the eight core system, the maximum system frequency suffers a significant drop. Looking at table 3 it can be seen this is due to the router having a lower maximum frequency. The router suffers from increasing complexity of the arbiter due to the lower number of possible destinations to choose from. Finally, the network adapter is independent of the number of processors in the system, as can be seen in table 3.

### 5.4 Application Aceleration

In this section the results presented concern the crossbar based system. The system speedup achieved and the theoretical ideal speedup for the target algorithm can be observed in the graph sketched in figure 4. The theoretical ideal speedup represents the maximum speedup the system can achieve given the parallel algorithm used. The broadcast optimization plays a big role in improving the speedup results, as it can be seen in figure 4. The system without broadcast presents small improvements with the increasing in number of processors and even drops when the eight core system is used. The reason for this is that more processors lead to the need to send more matrices and therefore the communication delay grows against computation delay. With broadcast, matrix B can be sent to all the processors simultaneously, which allows to save much time in communications. Although the speedup increases with the number of cores, as supposed, even with broadcast support the system is far from the ideal speedup, except for the two core system (which has always one latency cycle to send data do the other core).

These results are due to the communication overhead. Basically the communication latency is not uniform. Ideally the packet would take one cycle to cross the network, from being read from the FIFO until the data it travels with it being stored in the Mailbox (we are only taking into account the data packets, not the header packet). However this depends on two main factors: if there is more than one element in the FIFO and the arbiter state. If more than one element is in the FIFO, a packet is immediately chosen or being forced to wait till N-2 clock cycles, N being the number of processors in the system. The efficiency, represented in figure 5, indicates if the processors of the system spend too much time idle. A small efficiency, like the one observed for the eight core system, implies there is barely any gain from increasing the number of processors.

### 5.5 Crossbar versus NoC

In this section we use the crossbar switch based system without broadcast, so that a fair comparison can be made with the NoC based system. As both systems have eight cores the main difference lies in the communication architecture. Table 4 presents the synthesis results for both systems. Table 5 shows the individual results for the router modules. The area utilization of the NoC (see table 4) is vastly superior to that of the Crossbar Switch. This is mainly because the NoC uses nine router modules, compared to only one router module, with 8 entries, for the Crossbar Switch system. However, the NoC grows linearly with the number of cores (because each router has always the same size which implies the size of a individual node is fixed) while the crossbar switch grows faster, as can be seen in table 2 (because, as shown in table 3, the router grows almost quadratically with the increase in number of cores). This is the main reason why the NoC is better scalable than the crossbar switch. The NoC larger maximum clock frequency, is due to the less complex arbiter (which reduces the critical path).

The performance of the two systems was compared by analyzing the execution time of the parallel portion of the program, because this corresponds to the matrix multiplication parallelization. The two systems present equivalent performance with a short advantage for the NoC system. This is probably because of the NoC higher clock frequency, even though almost negligible (see table 6).

<table>
<thead>
<tr>
<th></th>
<th>4CoresB</th>
<th>8CoresB</th>
</tr>
</thead>
<tbody>
<tr>
<td>LUTS</td>
<td>336</td>
<td>41</td>
</tr>
<tr>
<td>BRAMS</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>$F_{\text{Max}}$ (MHz)</td>
<td>272</td>
<td>547</td>
</tr>
</tbody>
</table>

Table 2. Synthesis results for four core router and eight core router.

<table>
<thead>
<tr>
<th></th>
<th>1  %</th>
<th>2  %</th>
<th>4  %</th>
<th>8  %</th>
</tr>
</thead>
<tbody>
<tr>
<td>LUTS</td>
<td>833</td>
<td>1  %</td>
<td>1736</td>
<td>2</td>
</tr>
<tr>
<td>BRAMS</td>
<td>3</td>
<td>1  %</td>
<td>7</td>
<td>3</td>
</tr>
<tr>
<td>$F_{\text{Max}}$ (MHz)</td>
<td>206</td>
<td>197</td>
<td>194</td>
<td>174</td>
</tr>
<tr>
<td>$T_{\text{Min}}$ (ns)</td>
<td>4.9</td>
<td>5.1</td>
<td>5.2</td>
<td>5.8</td>
</tr>
</tbody>
</table>

Table 3. Synthesis results for system with one, two, four, and eight cores.
5.6 FPGA Implementation

The crossbar switch based multiprocessing system with four cores was implemented in the device Spartan-3E (XC3S500E-5FG320). To verify the results the system sends the data to the PC through the RS232 interface available on the development board.

The circuit can execute at a maximum clock frequency of 57 MHz. The implementation used a 50 MHz clock. The system consumes 7717 LUTs and 20 BRAMs, presenting a device occupation of 82% and 100%, respectively. The BRAMs are the clearly most critical resource, with even some RAM being mapped to LUTs due to the BRAM shortage.

The system was tested with an 8×8 matrix multiplication application and the results presented in the HyperTerminal on the PC coincided with the expected, proving the correct system functioning.

The system could be probably optimized for resource utilization, but this was not our main concern when designing the system. To multiply bigger matrices, an external memory would be probably necessary.

6. Conclusions

The main objective of this work consisted in the design and implementation of a multiprocessor system in a FPGA. The main contributions involving this work can be summarized as:

- Design of a homogeneous multiprocessor system with distributed memory and streaming communication.
- Development and comparison of two different communication architectures, a crossbar switch based architecture and NoC based with mesh topology architecture.
- Evaluation of scalability and performance of the Crossbar Switch based architecture.
- Acceleration of a matrix multiplication test application through mapping on a multiprocessor system with increasing number of cores.
- Implementation of a four cores multiprocessor demonstration system in a Spartan-3E device.

We concluded that being able to broadcast data is extremely important in parallel applications, because it reduces significantly the communication delay for certain algorithms and applications.

The evaluation of the two communication architectures studied concluded that the NoC based architecture scales better for an increasing number of processors in the system. This has to do with the router always having the same size contrarily to the crossbar switch router which increases quadratically in complexity with the number of cores supported. This also holds true for performance, as the crossbar switch architecture has an arbiter which increases in complexity with the number of processors, generating a greater delay across the network (is part of the critical path).

The crossbar switch based system has shown a speedup far from the ideal, even with broadcast. This fact is justified by the dynamic nature of the
communication delay. The two main factors are: FIFO read latency and round robin related latency.

Finally, it was concluded that software and hardware must be developed together to achieve the best results possible (performance and area related). From a software point of view it is also important to better explore the applications parallelism and conceive new models.

7. Future Work

For the future we intend to improve the NoC architecture, which has been shown to be the most promising one. Broadcasting support is the first requirement to be fulfilled, as we have seen it is a crucial factor to achieve better performance.

A better algorithm must be adopted to parallelize the matrix multiplication as well as other similar applications.

It is also of great importance to make the NoC more generic for easier scaling of the architecture. To improve usability the system must be revised to facilitate the integration of different modules (maybe through the Wishbone interface which would make our system compatible with several IP components which already support it).

Also, it would be interesting to study how our system scales with bigger matrices as input.

References


