Multi MicroBlaze System for Parallel Computing

P.HUERTA, J.CASTILLO, J.I.MÁRTINEZ, V.LÓPEZ
HW/SW Codesign Group
Universidad Rey Juan Carlos
28933 Móstoles, Madrid
SPAIN

Abstract: - Embedded systems need more computational power to satisfy today's applications' needs, like audio/video encoding/decoding, image processing, etc. An option for increasing the computational power of a system is to include various microprocessors and make them work in parallel. This paper presents a study of the viability of making a multiprocessor system on a chip (MPSoC) using the MicroBlaze soft-processor core from Xilinx. Performance of data communication is studied, and also a parallel application is used for testing speedup and efficiency of the system. [1]

Key-Words: - MultiProcessor, FPGA, System on Chip, MicroBlaze

1 Introduction

The increasing computing power needed for some complex applications has meant that parallel programming techniques and systems for solving complex problems in an optimal time, like clusters, multiprocessor systems, grid sytems, etc. have appeared.
Embedded systems need more computational power to satisfy today's applications' needs, like audio/video encoding/decoding, image processing, etc. and multiprocessor systems on a chip (MPSoC) are an option to deal with this increasing computational needs [1] and [2].
A parallel computing system is made up of various processing elements (PE) that work cooperatively to solve a problem. There are two main paradigms for communicating the PEs and allowing data exchange between them: message passing and shared memory. Each of them has its own pros and cons, but usually message passing is more popular in clusters and grid systems, and shared memory is popular in multiprocessor systems [3],[4] and [5].
In this paper a study of the use of various MicroBlaze processors in an embedded system and different architectures for communication between them are proposed. The message passing paradigm for data exchange between processors will be used.
In section 2, the main characteristics of the MibroBlaze soft-core processor used in our parallel system will be presented. In section 3 some communication alternatives for data exchanging between processors will be discussed. Some of its characteristics will be shown, as well as some tests about data communication performance. In section 4 a complete system using the chosen data communications architecture and some results showing the performance of the complete system will be presented. Section 5 will show conclusions about this work and future work lines about parallel architectures for embedded systems.

2 MicroBlaze soft-processor core

The MicroBlaze embedded soft core is a reduced instruction set computer, optimized for Xilinx FPGA implementations. Figure 1 shows a block diagram of the MicroBlaze core [6].
The MicroBlaze embedded soft-core includes the following features:

- Thirty-two 32-bit general purpose registers
- 32-bit instruction word with three operands and two addressing modes
- Separate 32-bit instruction and data buses that conform to IBM's OPB (On-chip Peripheral Bus) specification
- Separate 32-bit instruction and data buses with direct connection to on-chip block RAM through a LMB (Local Memory Bus)
- 32-bit address bus
- Single issue pipeline
- Instruction and data cache
- Hardware debug logic

[1] This work has been supported by the Spanish PROFIT and Medea+ program Ander FIT-0700000-2003-930 contract.

• FSL (Fast Simplex Link) support
• Hardware multiplier (in Virtex-II and subsequent devices)

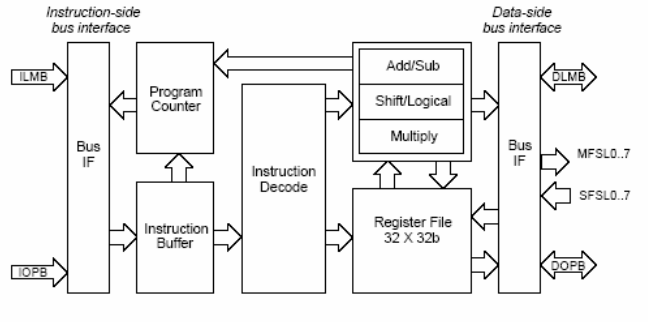

Fig.1. MicroBlaze block diagram[6]

The MicroBlaze core implements a Harvard architecture. It means that it has separate bus interface units for data and instruction access. Each bus interface unit is further split into a Local Memory Bus (LMB) and IBM's On-chip Peripheral Bus (OPB). The LMB provides single-cycle access to on-chip dual-port block RAM. The OPB interface provides a connection to both on-and off-chip peripherals and memory. The MicroBlaze core also provides 8 input and 8 output interfaces to Fast Simplex Link (FSL) buses. The FSL buses are uni-directional non-arbitrated dedicated communication channels.

3 Communication Architecture

As seen in the previous section, MicroBlaze contains eight input and eight output FSL interfaces. The FSL channels are dedicated unidirectional point-to-point data streaming interfaces. The FSL interfaces on MicroBlaze are 32 bits wide. Further, the same FSL channels can be used to transmit or receive either control or data words. The performance of the FSL interface can reach up to 300 MB/sec. This throughput depends on the target device itself. The FSL bus system is ideal for MicroBlaze-to-MicroBlaze or streaming I/O communications [7].

The main features of the FSL interface are:

• Unidirectional point-to-point communication
• Unshared non-arbitrated communication mechanism
• Control and Data communication support
• FIFO-based communication
• Configurable data size
• 600 MHz standalone operation

The FSL bus is driven by one Master and drives one Slave. The next figure shows the principle of the FSL bus system and the available signals.

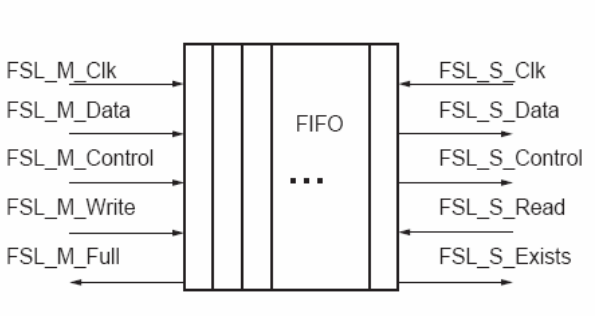

Fig.2. Fsl bus signals[7]

Xilinx EDK provides a set of macros for reading and writing to or from an FSL link. There are two ways of reading/writing on an FSL link: blocking or not blocking, and also there are different instructions for reading/writing data words or control words.
For testing and measuring the capabilities of the FSL links, a simple system has been built. This system consists of 2 MBlaze cores interconnected via 2 FSL links, BlockRAM blocks for data and instruction memory for each processor connected to the local memory buses, an uartlite core for debugging an I/O purposes connected to MB_0, and a timer for measuring data transfer times connected also to MB_0. A diagram of this system can be seen in the next figure:

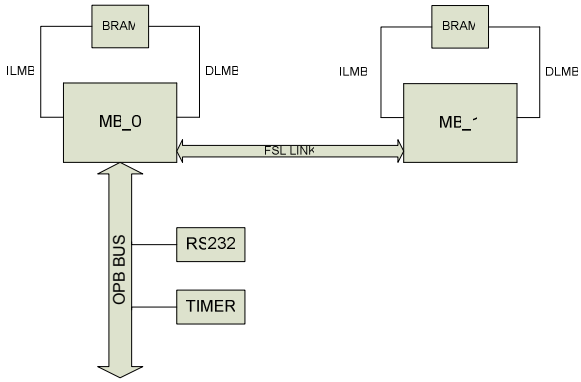

Fig.3. 2 MicroBlaze system for testing communications

This system was used to test the viability of transmitting data over the FSL links and for measuring the time consumed in this task.
For testing the speed of the links, a program for data transfers has been developed. The program was used for transferring different sizes of data, from a simple 32 bit word to a matrix of 32x32 unsigned integers (4 Kbytes).
In the next graph we can see the different speeds obtained with these tests.

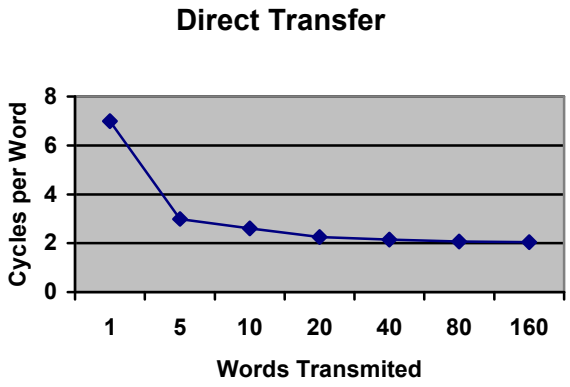

Fig.4. Direct transfer speeds over FSL link

In this graph the cycles per word consumed for different sizes of data transmitted are represented. With direct transfer we mean that the data is transferred using the macros provided by Xilinx for managing the FSL links invoking the macro for each word to be transferred, not using any kind of control loops. The time measured only includes the time to invoke Xilinx macro to send data in one side, and to receive data on the other side, it does not includes the time to read/write data from/to memory. With this kind of transmission and large data sizes, speeds of 2 cycles per word transmitted are reached. This configuration is good if a low volume of data is transferred, but for high sizes of data it is not feasible to call the FSL transfer macro for each word, because it will make the program size too big. For example, for transferring matrices a double for must be used to control the loop to access each element of the matrix and call the FSL transfer instruction only once in each iteration of the loop. With this kind of transmission, a lot of cycles will be spent in the control loop code. The results of the matrix transfers are shown in the next figure.

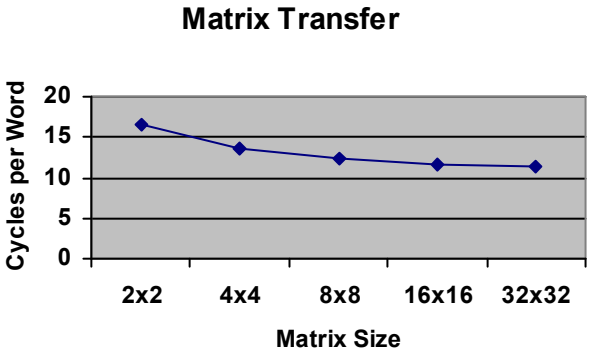

Fig.5. Matrix transfer speeds over FSL link

As can be seen for matrix transfer, speeds of nearly 11 cycles per word can be reached. This is more than 5 times slower than direct transfer, but still quick enough to obtain good results in parallel algorithms with large computing times per data word processed. In this case, the time measured includes the time for reading words from memory, invoking Xilinx macro, and time spent in control instructions from the control loop (incrementing loop variable, testing range, etc).

After testing the capabilities of the FSL links for point to point data transfers, it was necessary to decide which "network" architecture to use for connecting various processors in a cluster. There are many network topologies that can be materialized with point to point links. The pros and cons of three of them will now be discussed and the viability of using them in a MultiMicroBlaze design using FSL links for point to point data transfer:

-Completely Meshed: a completely meshed network is a network in which each node is connected to every other node in the network. It is a good way to reduce the travelling time of packets over a network, because data goes directly from sender to receiver, but its main disadvantage is that the number of links grows extremely quickly when the number of nodes is increased. A completely meshed network topology with MBlaze and FSL links will only be possible for just 9 MBlazes, because of the limitation of 8 FSL links for each MicroBlaze processor.

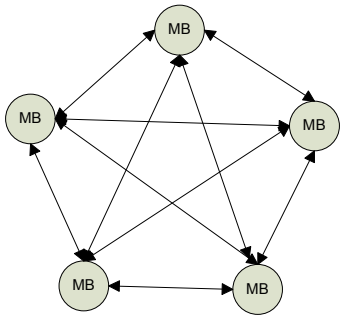

Fig.6. Completely meshed network

-Ring Network: a ring network is a network in which each node in the network is connected to the following and the preceding node in the network, forming a ring. Data is passed from node to node until it reaches the destination node. With MBlaze and FSL links there will not be a size limit for the network, because each MBlaze would just use 2 FSL links. The main problem of this topology is that data transfers from two nodes that are far from each other are very time consuming.

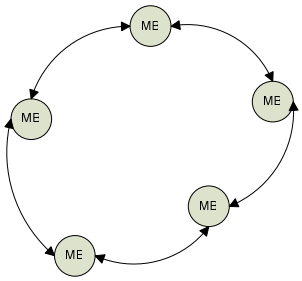

Fig.7. Ring Network

-Star Network: a star network is a network in which each node is connected to a central node. The weak point of this topology is that if the central node fails, the whole system fails. This weakness is not very important in an embedded system, where all the nodes are in the same chip. Using this topology it is possible to build systems with 1 MBlaze as a central node, and up to 8 MBlazes as general nodes. Also bigger systems can be built by linking various subsystems together. This is the choice taken for developing our Multi MicroBlaze System.

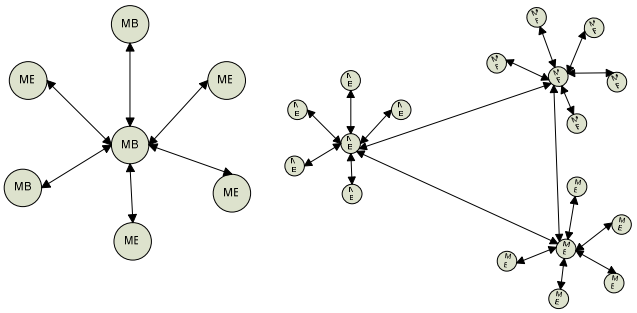

Fig.8. Star network (left), and linked star networks (right)

With this architecture, the central node will be the one that decides which fragments of the work are assigned to each general node, and will also be responsible for grouping the results given by the general nodes.

4 Complete System

Once the communication method (FSL links) and the network topology (star network) were decided, 4 systems were built with: 1, 2, 4 and 8 MicroBlazes each. The system with 1 MBlaze consisted of just the central node, and was built with 1 MBlaze, 16 KBytes of Block RAM for instruction and data memory, an uartlite and a timer attached to the OPB bus. The general nodes were built with one MBlaze and 16 KBytes of Block RAM for instruction and data memory.

With this system some parallelizable applications for testing the speedup obtained due to the use of many processors instead of one were used.

The first application used to test the systems was an application that performed matrix multiplications. The parallelization of the matrix multiplication algorithm implemented consists of sending one or more rows of the first matrix, and the whole second matrix to each processor. So each processor obtains one or more rows of the resulting matrix.

The speedup of a parallel algorithm is defined as the time needed to solve the problem using just one processor divided by the time needed to solve the problem with p processors. In an ideal system and with an ideal parallel algorithm, the speedup would be equal to the number of processors p.

$$speedup = \frac{t_s}{t_p}$$

Another measure of the performance of a parallel system is efficiency. Efficiency is defined as the speedup per processor. In an ideal system with an ideal parallel algorithm, the efficiency would be equal to 1.

$$efficiency = \frac{speedup}{p}$$

With this matrix multiplication program various tests were performed with a different number of processors, and with different matrix sizes. The time measured includes the time for: reading the matrices, distributing data to processors, performing the multiplication, and reading back the results. The results obtained for speedup and performance are shown in the following graphics.

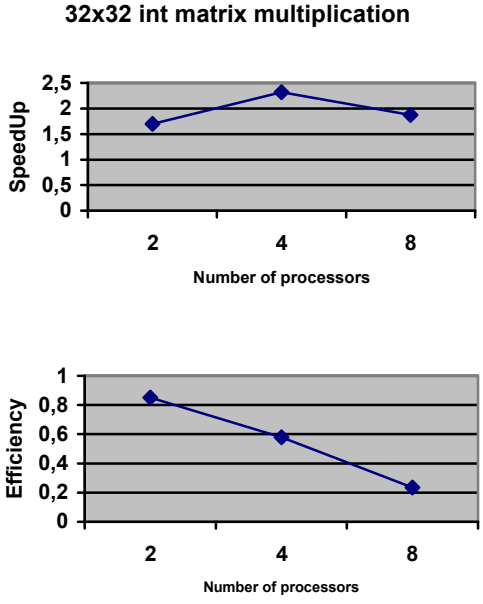

Fig.9. Speedup and efficiency for 32x32 integer matrix multiplication parallel program

As it can be seen in this graph, results are not as good as it could be expected. This is because when the number of processors is increased, more time is spent in communications, consequently the time saved with parallelism is spent in data communications.
The second test consisted of the same matrix multiplication algorithm, but using floating point matrices. Floating point multiplication requires more time than integer multiplication because floating point operations must be emulated due to the lack of a floating point unit in the MicroBlaze processor. In this test, the time consumed in communication will be the same as if the matrices were integer, but as the processing time is bigger, better results in terms of speedup and efficiency are expected.
The results obtained from this test are shown in the next figure.

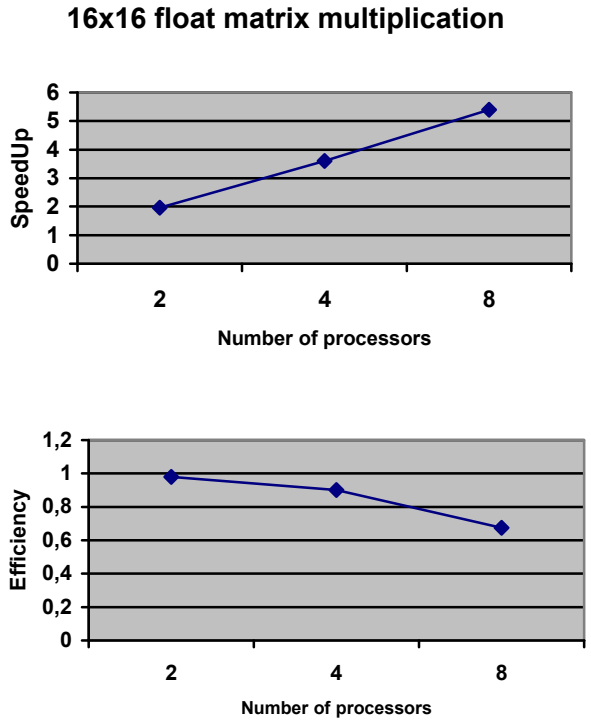

Fig.10. Speedup and efficiency for 16x16 float matrix multiplication parallel program

As it can be seen in this graph, better results are obtained. But efficiency also decreases when the number of processors increases, due to the overhead in communications. A possible solution for saving time in communications and improving the system's efficiency would be to find a way for broadcasting messages that are common to all processors instead of sending one message to each processor. In this test, broadcasting the second matrix to all processor would save a lot of time.
To have an idea of how much FPGA space is needed for multiprocessor systems using Microblaze, some synthesis results will be presented. The system developed has been tested on an RC300 board from Celoxica, equipped with a XC2V6000 -4 Virtex 2 FPGA. The whole system's clock speed was 50 MHz. The synthesis results obtained for the final system (8 MBlazes) are presented in the following table.

Logic Utilization:		
Total Number Slice Registers:	3,017 out of 67,584	4%
Number of 4 input LUTs:	5,760 out of 67,584	8%
Number of occupied Slices:	5,098 out of 33,792	15%
Total Number 4 input LUTs:	8,940 out of 67,584	13%
Number of Block RAMs:	64 out of 144	44%
Number of MULT18X18s:	24 out of 144	16%
Number of GCLKs:	1 out of 16	6%

Fig.11. Synthesis results in a XC2V6000-4 FPGA

From this table, it is clear that a multiprocessor system with only 8 processors spends 15% of the area of the FPGA, so bigger systems can be fitted into this FPGA. The main problem for multiprocessor systems on FPGAs is that there is not enough BlockRAM: a system with 8 processors with 16 KBytes each, spends almost half the BlockRAMs (44 %) of a big FPGA even when 16 Kbytes are not as many RAM as it would be desirable. A possible option for the memory problem would be to use external RAM, but FPGA boards usually have 2 or 4 memory banks, so they should be shared between different processors slowing down memory access.

5 Conclusions

A system using MicroBlaze and FSL links to communicate between them has been developed, and with the experiments tested it can be concluded that the FSL links are an ideal choice for exchanging data between processors due to their high speed data transfer rates. A system with 8 processors connected with FSL links has been used to test some parallel algorithms. With the results of these tests, it can be seen that the system is very appropriate for parallel algorithms in which the data transfer time is substantially lower than the computing time. Analysing the synthesis results, it is clear that what limits the number of processors in a multiprocessor design on FPGA is the amount of BlockRAM available, not the total size of the FPGA.

Future work includes the study of other buses for data communication, alternatives to the reduced amount of

BlockRAM in FPGAs, and the development of a software library (probably based on the MPI library) to make the writing of parallel programs easier for the software developer based on the underlying hardware architecture and the communication engine of the whole system.

References:

[1]. Wolf, W: The Future of Multiprocessor Systems-on-Chip. *Proceedings of the Design Automation Conference (DAC'04)* 2004. pp. 681-685

[2]. Wolf, W: Multimedia Applications of Multiprocessor Systems-on-Chip. *Proceedings of the Design, Automation and Test in Europe Conference (DATE'05)*. 2005. pp. 86-89

[3]. Grama, A; Gupta, A; Karypis; G, Kumar, V: *Introduction to Parallel Computing*. Addison Wesley 2003.

[4]. Culler, D; Sing, J.P; Gupta, A: *Parallel Computing Architecture: A Hardware Software Approach*. Morgan Kauffman Publishers 1999.

[5]. Dongarra, J; Foster, I; Fox, G; Gropp, W; Kennedy, K; Torczon, L; White, A: *SourceBook of Parallel Computing*. Morgan Kauffman Publishers. 2003

[6]. Xilinx: *MicroBlaze Processor Reference Guide.(v 4.0)*. 2004

[7]. Xilinx: *XAPP529: Connecting Customized IP to the MicroBlaze Soft Processor Core Using the Fast Simplex Link (FSL) Channel*. *(v 1.3)*. 2004