HW-SW CO-DESIGN OF MPSOC USING FGPA IP CORES

Iulian NIȚĂ[1], Gabriel ZDRU[2]

Noile tehnologii de proiectare a sistemelor cu chip multiprocesor bazate pe FPGA-uri şi blocuri IP fac posibilă dezvoltarea unor dispozitive optimizate din punct de vedere al performanţelor, consumului de energie şi al costului. Marea flexibilitate oferită de aceste noi instrumente de proiectare permite explorarea spaţiului de proiectare pentru a căuta cele mai eficiente implementări. Astfel, în această lucrare, am realizat o cercetare în domeniul acestor tehnologii şi am propus un model de proiectare concurentă hardware/software (hw/sw co-design) pentru dezvoltarea aplicaţiilor pe sisteme cu chip multiprocesor bazate pe FPGA IP cores. Rezultatele experimentale au fost validate pe o aplicaţie de filtrare a imaginilor, implementată pe un sistem multiprocesor, folosind kitul de dezvoltare Xilinx XUP Virtex 5 şi pachetul software Xilinx EDK.

The new design technologies of multiprocessor systems on chip based on FPGAs and IP blocks, make possible the development of optimized devices in terms of performance, power consumption and cost. Flexibility offered by these new design tools allow design space exploration to search for the most effective implementations. Thus, in this paper, we performed a research on these technologies and we have proposed a hardware / software co-design model for developing applications on multiprocessor systems on chip based on FPGA IP cores. The experimental results were validated with an application for filtering images, implemented on a multiprocessor system using the development kit Xilinx XUP Virtex 5 and Xilinx EDK application software

Keywords: Multiprocessor System on Chip, FPGA, MicroBlaze, image filtering

1. Introduction

In terms of hardware, designing systems on chip using FPGAs, offers a greater flexibility, due to the large number of IP cores available on the market and endless possibilities of configuration, customization and interconnection between them [1].

[1] Assist., Depart. of Applied Electronics and Information Technologies, University POLITEHNICA of Bucharest, Romania, e-mail: iulian.florin@gmail.com

[2] Student, Depart. of Applied Electronics and Information Technologies, University POLITEHNICA of Bucharest, Romania

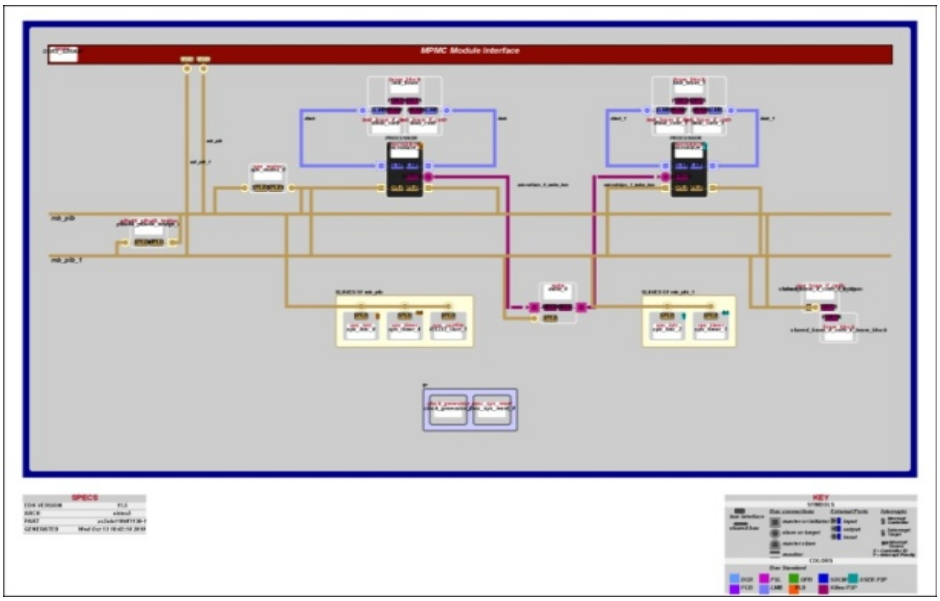


Fig 1. Design example for dual Microblaze core using FPGA IP cores

Among the main advantages of designing using FPGAs, we can mention:

Flexibility: the number of IP cores which can be integrated is limited only by the FPGA's capacity. For example, the Virtex-5 FPGA from Xilinx has 330,000 logic cells[2], which allow implementing from 80 to 100 MicroBlaze processors.

Configurability: each IP block is configurable depending on application requirements. For example, for the MicroBlaze processor there are over 300 possible configurations, in which you can add or remove optional modules such as: FPU (Floating Point Unit), BS (barrel shifter), MUL (Multiplier hardware), DIV (divider hardware) [3]. Likewise you can select the operation frequencies and set the pipeline's depth. Also, depending on the needs of the application, the cache memory can be set at various capacities [4].

Reduced time to market: in the design process it is no longer required to manufacture the integrated circuit, this being implemented on FPGA just in a couple of minutes. By using the existing IP blocks and being able to reprogram the FPGA whenever needed, the design and testing times are considerably reduced.

Reduced cost: it is cheaper to buy an FPGA which can be reused in multiple projects rather than buying special chips that can be used only in specific projects (ASIC). Thus, in the FPGA's case, detecting an error in the designed architecture does not involve buying a new module, but only reconfiguring and reprogramming the design [4][5].

Xilinx and Altera are the leading solution providers for designing embedded systems using FPGAs, IP cores and specific software packages. With these software applications, users can modify the systems on chip by integrating and interconnecting IP cores, and finally a review of the system can be made by generating reports that give details about power consumption, the size of the circuit and technical performances. These programs enable architectural analysis to optimize chip design and this way, users can reduce the chip size, the power consumption and the cost without sacrificing the performance. [5]

Xilinx Embedded Development Kit (EDK) represents a set of tools used for designing systems on chip with one or more processors. The main components of EDK are[6]:

Xilinx Platform Studio (XPS) – is a developing environment used for designing system hardware and contains a library of IP blocks which can be configured and interconnected to fulfill the designing requirements of the application [6][7].

Software Development Kit (SDK) – is a developing software tool used for the designing of projects made with XPS, used for making and verifying dedicated software applications written in C/C++. SDK is build on the Eclipse open source platform, which is familiar to many software developers [6][7].

The MicroBlaze processor is the main component in the multiprocessor dedicated systems made with XPS.

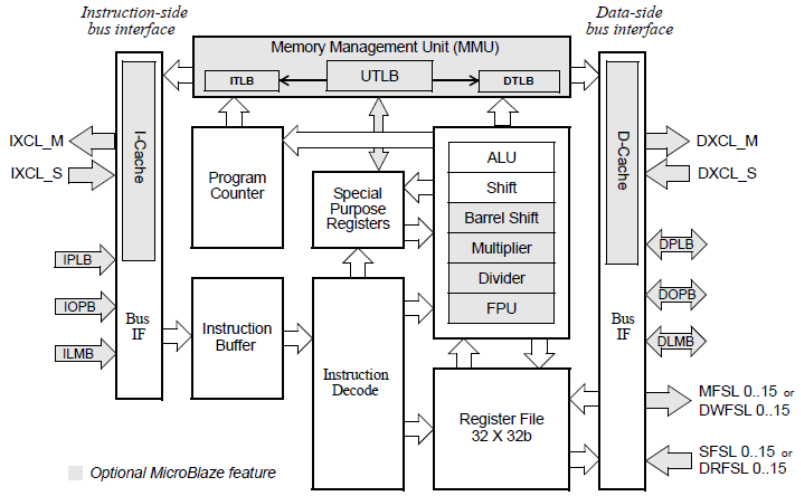


Fig 2. – Architecture of Xilinx MicroBlaze processor [3]

The MicroBlaze processor – is a processor based on RISC type architecture and it's presented in a hardware description language(VHDL). It has a series of basic specifications such as: 32 general purpose registers on 32 bits, 32

bits instructions with 3 operators and 2 addressing ways, a 32 bits address bus, a pipeline architecture based on 3 or 5 levels [3].

To obtain a more powerful processor, the optional modules must be activated and functional frequency must be set to maximum. Of course, these settings will lead to higher power consumption and higher cost expressed in the number of logic cells. Therefore, in order to obtain more efficient implementations, we must experiment every configuration option and choose the optimum solution.

2. Hardware-Software co-design method for MPSoC

Exploring the design space offered by the FPGA logic, software tools and existing IP blocks offer the required flexibility for the designing of multiprocessor systems on chip. In most of the cases, system architecture can be implemented by meeting the imposed performance constraints. In some cases, there can be more implementing solutions, each with its **advantages and disadvantages[8].** For example, an implementation with low power consumption can have high execution times, or an implementation with low execution times can use a high number of logic cells [9].

Architecture is created by following a series of steps, as shown in figure 3. The designers examine the applications requirements, apply the constraints and make first hardware architecture as well as software architecture. Hardware wise, the architecture is designed by interconnecting and configuring the various IP blocks. The communication and synchronizing mechanisms are then established. Software wise, the application is divided in tasks and the dependencies between tasks and data are then analyzed. The communication procedures and the tasks execution order are then defined. The space and temporal mapping of the tasks on the available processing elements is then accomplished, by having in mind the obtainment of the lowest execution time. An operating system is then choose, and the memory locations for the storage of instructions and data are selected. A prototype is created and executed on the FPGA board by making many iterations to observe the possible problems which can appear or other ways of improvement. If the results are not satisfactory, the hardware architecture can be adjusted and the software application and mapping mode can be optimized or redesigned. This process repeats itself until, after a series of tests and verifications, the initial designing requirements are matched. Then the final implementation can begin [10][11].

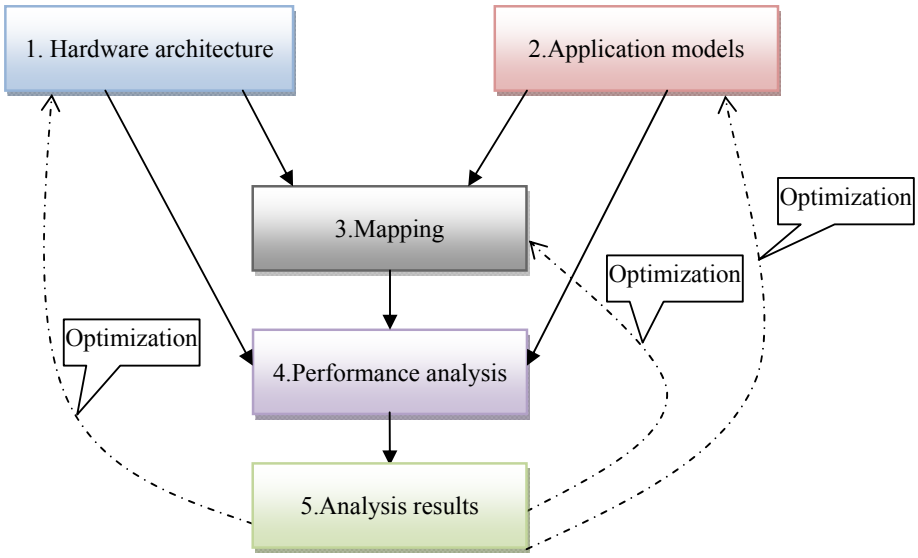


Fig.3. Hw/Sw co-design for MPSoC using FPGA IP cores

2.1. Hardware design

The architecture of multiprocessor systems on chip based on FPGA using IP blocks, can be implemented using a common bus to all processors, or can be made from a hierarchy of buses corresponding to each processor, interconnected by linking elements[12].

Although the shared bus architecture has the advantage of lowering the power consumption and the cost of the chip by using a single bus, it has a major disadvantage: the arbitration protocols that allow the access of the processors to the bus are slowing down the execution of the application. Due to the fact that at a certain point the access to the bus is allowed to a single processor, the arbitration protocols such as Round Robin and Priority Based, force the other processors to wait until the bus is ready [13].

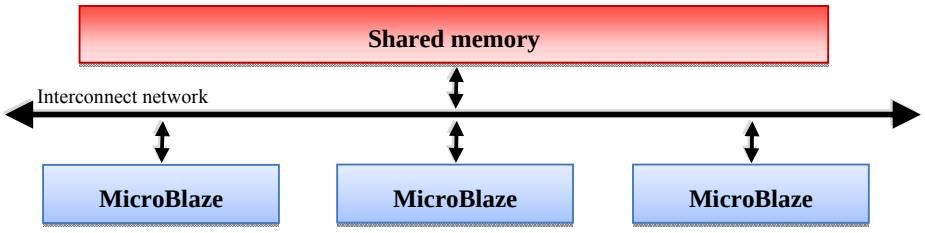


Fig.4. Shared memory multiprocessor system arhitecture

Although the chip cost and the power consumption is higher because new components have been added (buses) to the system, the architecture with multiple buses allows a faster execution of the applications because the usage of protocols is no longer needed. The intercommunication between subsystems based on processor bus groups is made using dedicated components such as: internal or external shared memory, Mailbox, Mutex, FSL or Bus Bridge [8][13].

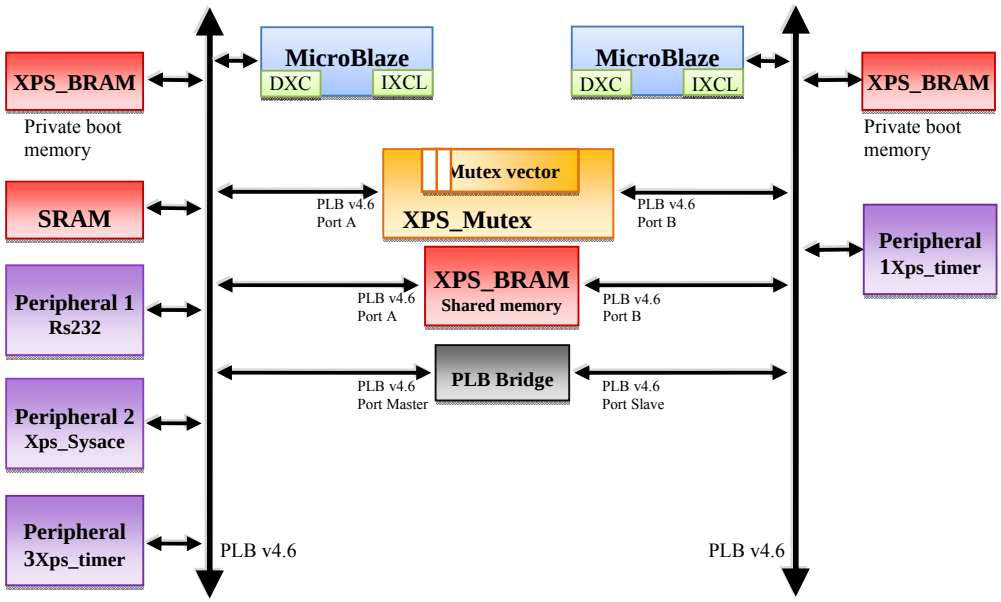


Fig.5. System architecture with two processors that can communicate with each other

Common components can have 2 or more interconnection ports. By their nature, these multiport connection components allow the interconnecting of multiple buses independent one to another. By isolating each subsystem, a subsystem's bus will be ready when the other subsystems use its own bus to send data. These subsystems can have a set of peripherals corresponding to each subsystem but also a set of common peripherals.

In this paper we have used a multiprocessor system on chip implemented by 2 subsystems interconnected through the IP cores XPS_Mutex, Shared XPS_Bram and XPS_PLB_Bus_Bridge, having as common shared resource the IP core peripheral XPS_RS232. Every subsystem contains one MicroBlaze processor, a PLB bus, an XPS_timer and a private memory called XPS_Bram. Moreover, the first subsystem contains the external SRAM memory controller and the RS232 and XPS_Sysace peripherals. The role of each functional block of this architecture is described as follows[8][13]:

XPS_RS232 is the controller used for serial communication to the computer through HyperTerminal, required for the display of messages during the execution.

XPS_Mutex is used to ensure the exclusive access to the shared peripheral RS232.

Shared XPS_Bram is the memory zone shared by the two processors of the system, that is required for synchronization and exchange of data.

XPS_PLB_Bus_Bridge is used to unite the two buses so that the second processor can access the RS232 controller which is connected to the first processor bus.

PLB is the local processor dedicated bus, through which all the functional blocks of every subsystems are interconnected.

Private XPS_BRAM is the processor's private cache memory for data and instruction.

TheXPS_Timer is used to obtain the execution time of every stage of the program.

The SRAM memory is used to store the image processed by the application, because the image is too big and it doesn't fit in the internal memory.

XPS_Sysace is a controller used for accessing the external memory card, where the images that need to be processed are read[8][10][13][14].

2.2 Software Design

In order to test the performance of the architectures with one or two processors we've implemented an image filtering algorithm. The incoming data for this algorithm consists in an image with a salt and pepper type noise, and we try to remove it by applying 3 filters: a medium filter, a smoothing effect filter and a sharpening effect filter.

A black and white image, measuring 210x280 pixels has been used during the tests. The format of the image is bmp and the information of a pixel is stored on 8 bits, therefore a pixel can have a decimal value ranging from 0 for black to 255 for white[15].

After the image is extracted from the memory card and converted in the pixel values matrix, the following steps must be followed for each filtering process: a 3x3 size window containing the current pixel and his 8 neighbors are extracted from the image a convolution is performed between the window and the filter matrix, the value of the output pixel is stored and the window sweeps over the entire image resulting the filtered image [16].Each filter has its own method of calculating the output pixel, thus the median filter performs an ascending sorting and chooses the average value, the smoothing filter performs a convolution of those matrices and then the result is divided by 9 and the sharpen filter makes only their convolution[17]. These steps are illustrated in figure 6.

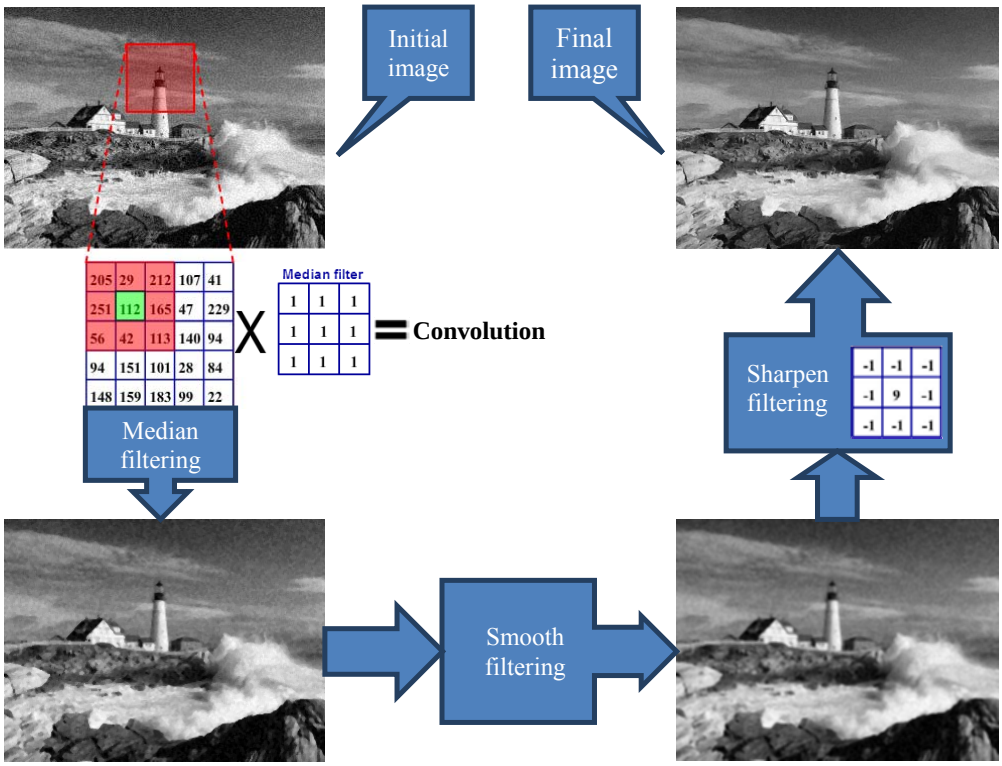


Fig. 6. Image filtering for reducing the noise

Given the fact that three filters have been applied (in a cascading way) for the removal of image noise, one processor can apply a new filter to another previously filtered image (by another processor).In this way we obtain a pipeline type image filtering which is useful when we have a large number of images that need to be filtered or when the processed image is divided into several blocks. Using this type of parallelization we have proposed two algorithms:

1. An algorithm where the first processor applies a median filter then the second processor waits for the completion of the previous filtering and then applies a smoothing filter, and after that, both processors apply(one line/ each processor) a sharpen filter.
2. An algorithm where the second processor applies a median filter, then the 1st processor waits the completion of the previous filtering and applies a smoothing filter and after that both processors apply (one line/ each processor) a sharpen filter.

Another way to parallelize this algorithm is to use the data parallelism, which allows two processors to work at the same time, on different areas of the image. Thus we have proposed two other algorithms:

3. An algorithm in which each processor applies all three filters sequentially (one processor applies filters to even lines, and the other processor applies filters to odd lines) on different lines. In this algorithm the two processors work on the image in a symmetrical way.

4. An algorithm where the processors act similar to the previous algorithm except this time the data is divided in a lop sided manner, thus the 1st processor applies the filters on every 3rd line, and the 2nd processor applies the filters on the 2 lines skipped by the1st processor .

In these algorithms that use data parallelism, the task mapping was initially made by dividing the tasks equally on each processor (this happens in the symmetric algorithm), thus the 1stprocessorworks on the even lines and the 2nd processor works on the odd lines. Analyzing the performance of the two processors obtained using the symmetrical algorithm, we adopted an unequal division of the tasks on each processor. In this way, in the lop-sided algorithm the 1stprocessorreceives one task while the 2nd processor has two tasks assigned to it[14][18].

Given the fact that the maximum size of internal data memory is 256 KB and that the program used for reading an image, line by line, requires more than 500KB memory, we choose to add an external SRAM memory to the 1stsubsystem (1st processor) which contains the xps_sysace peripheral used to access the memory card. In this way, the 1stprocessorreads the entire image and stores it in the shared memory and therefore, the 2nd processor does not require an external memory due to the fact that the image filtering program code is small enough to fit within the internal memory of 128 KB.

Choosing the memories that will be used in the system is very important because using an external memory affects the whole system performance. Therefore the 1stprocessorwill achieve a smaller performance because of the delays on the PLB bus, unlike the 2nd processor which has both data and instruction zones mapped in the local memory. The mapping of such areas in the external memory leads to a decrease of system bus frequency (this is because connecting both the data and the instruction interface to the system bus, they behave as two master components related to the access bus arbitration)[13][19]. In other terms, the 1st processor bus runs at a frequency of 303MHz while the 2ndprocessorbus runs at a frequency of 324MHz. These influence the frequency of shared memory controllers, thusthe1stprocessor shared memory controller runs at a frequency of 280MHz while the 2ndprocessorshared memory controller runs at a frequency of 430MHz. Therefore, dividing the filtration tasks in an asymmetrical mode, the 2nd processor (which is faster) is being used more efficiently.

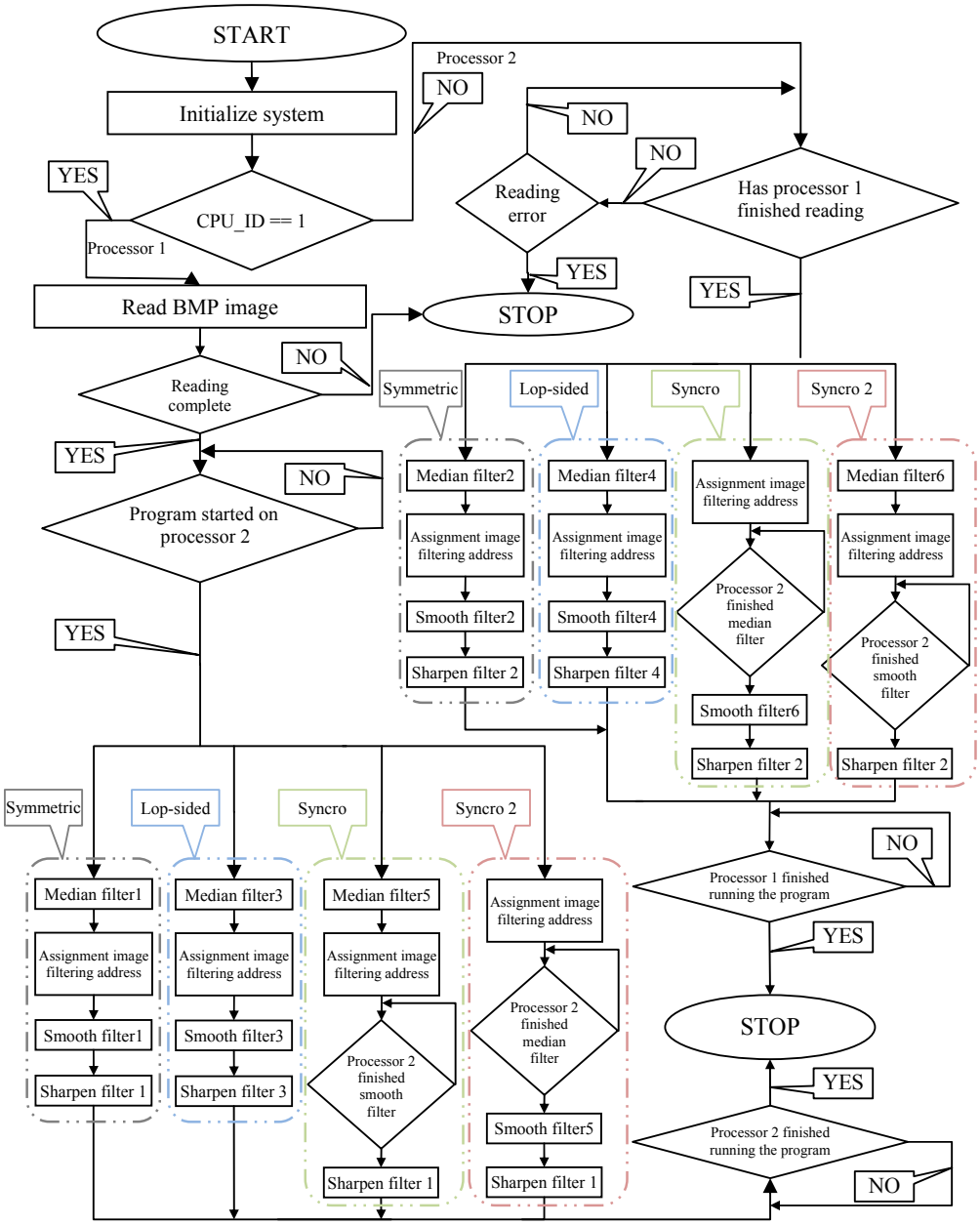


Fig.7. Block diagram of parallelized algorithms

3. Synthesis Results

In order to perform the experiments, we used the Xilinx XUP V5development kit [1] and Xilinx Embedded Development Kit software package [6][7]. With these tools we have designed a system on chip with one Microblaze processor and a multiprocessor system on chip with two Microblaze processors. We also analyzed the performance criteria in a comparative way (regarding execution time, power consumption and cost expressed in logic cells) for running an image filtering algorithm described in subparagraph 2.2. Having as reference the results obtained on the architecture with one processor, the goal is to achieve an efficient implementation on the two processors architecture, with lower execution times and also a reduced power consumption and cost.

In order to achieve this, a performance analysis is made on various configurations of the MicroBlaze processor (by enabling or disabling optional modules). Thus, the optional modules used are: Barrel Shifter (wrote B) 32-bit multiplier (wrote M), hardware divider (wrote D) pipeline depth (wrote O, 5 levels for O deactivated and three levels for O activated). In terms of software, the compiler allows three optimization levels: l = low level software optimization, m = medium level software optimization, h = high level software optimization.

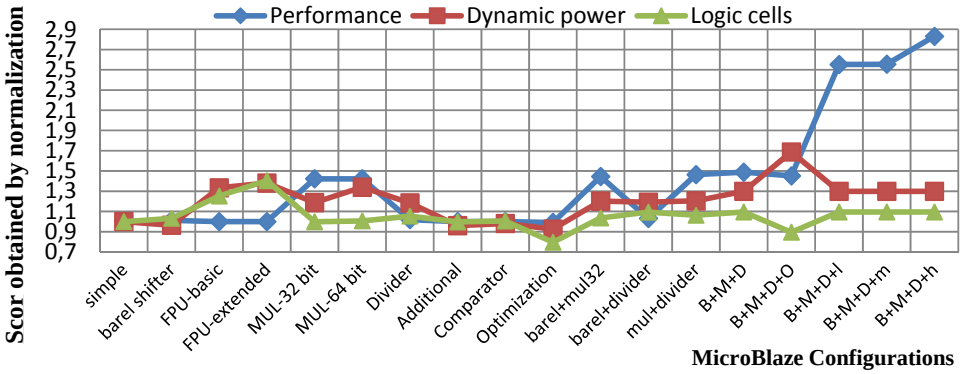
For the image filtering application, as we see in chart 1, the only significant improvement on both the execution time (represented graphically as a value inversely related to performance) and the logic resources used (lower score is better) is obtained at the activation of 32-bit multiplier and other configurations that use this option. Therefore, the best performance is achieved using the configuration B + M + D with a score of 1.48 for performance, 1.09 for logic resources and 1.29 for dynamic power, while the optimal configuration regarding the consumption of logic resources is B + M + D + O with a performance score of 1.45 (corresponding to a total filtering time of 3251.67 ms) 0.89 for logic resources and 1.68 for dynamic power.

A series of software optimizations can be added to these configurations. The Xilinx Platform Studio application (XPS) provides the ability to make improvements in the compiler, so the designer can use four optimization levels: 0, low, medium and high:

- low level - is done by tweaking the jump and pop instructions
- medium level - performs almost all the optimizations available that do not involve a significant increase in the size of the memory consumption. The compiler does not perform optimizations to the waiting loops or to the data accessing memory.
- high level - in addition to the medium optimization level, improvements that will increase the size of the executable file are added [6][7].

In order to obtain the best performance without affecting the logic resources used, with software optimization we can reach a maximum score of 2.83 corresponding to a filter time of 1668.215 ms for the BMDh configuration.

If the software optimizations are applied to the BMDO configuration, similar greatly improved performance will be obtained but also, the performance difference between BMD and BMDO of 0.035 which corresponds to a total filtration time increased by 77ms (for BMDO) will be kept.



Graph 1 Performance, dynamic power and logic cells normed to simple configuration

B=Barelshifter M=Multiplicator D=Divizor O=Optimizare l=low m=medium h=high

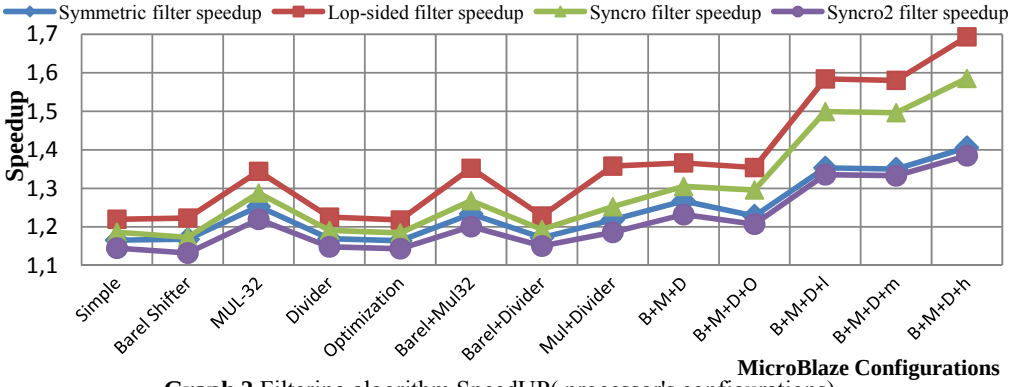
Although we have found an optimum ratio between performance and logic resources used, we also intend to optimize the dynamic power consumption. As expected, there is no perfect configuration, because of the high power consumption of the B+M+D+O configuration (the best one). In this case we have noticed that this configuration led to a significant increase of the dynamic power consumption reaching the value of 0.764W (score 1.68) compared to the minimum consumption of 0.419W (score 0.99) obtained in the configuration where the only enabled option was the optimization. Thus in terms of system power consumption the configurations B+M+D and M+D are compared, yet having a moderate power consumption. These two patterns have increased power consumption up to 9.3% (0.18 W) however they bring a better performance of up to 50% (1.58 seconds).

The optimal configuration in terms of dynamic power consumption is B+M+D+h. The aim of this paper is to find the Speed Up of the multiprocessor system, and that is why we choose the configuration with the highest performance at the expense of power consumption.

Along with the transition to the 2 processor architecture, a filtering algorithm transformation has to be made in order for the 2 processors to work in parallel (as can be seen in figure 7 above). Thus, the filtering algorithm was parallelized resulting two types of algorithms: one based on data parallelism

(Symmetric and Lop-sided) and one based on a pipeline structure**.** (Syncro and Syncro2). All these algorithms were summarized in section 2.2

The following graph presents the speedup of each algorithm and as a result of the performance obtained by running these algorithms. The Speed Up actually represents the performance improvements brought by the architecture with two processors comparing to the performance obtained on single processor architecture. By analyzing this graph we can easily see that enabling the 32-bit multiplier has a positive impact on Speed Up's filtering algorithms. Also, from this graph we can see that the best performance is obtained in the asymmetrical algorithm since the new added processor, which uses only internal memory is more efficiently exploited. Similar to the architecture with one processor, in this case the most capable configuration is B+M+D+h with a maximum Speed Up of 1.69.

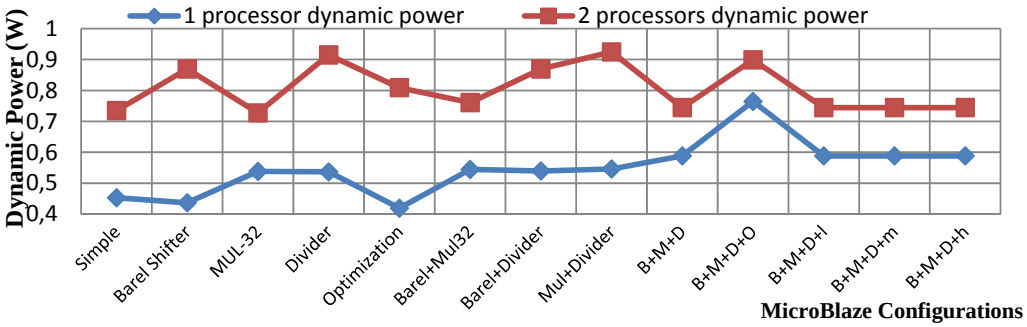


Graph 2 Filtering algorithm SpeedUP(processor's configurations)

In terms of dynamic power consumption it can be seen that the enabling of an option which brings significant performance benefits, will not necessarily lead to increased power consumption. This is observed for the activation of the 32-bit multiplier where the architecture with two processors has obtained a consumption with 0.007 W less than the consumption obtained on the same architecture but with the basic configuration. On the other hand, the architecture with two processors has a slightly higher consumption than the one obtained on single processor architecture because the dynamic power depends on the interconnection node capacitance, circuit voltage and switching frequency [20].Therefore, if you add new components to architecture, they require additional interconnection nodes that will lead to an increase of the dynamic power consumption for example: the architecture with 2 processors consumes with 0.189W more than single processor architecture for a configuration in which the 32 bits multiplier is activated. The dynamic power consumption difference between the two architectures varies depending on the processors configuration, so that the maximum difference is reached in the configuration with the barrel shifter activated (it consumes with

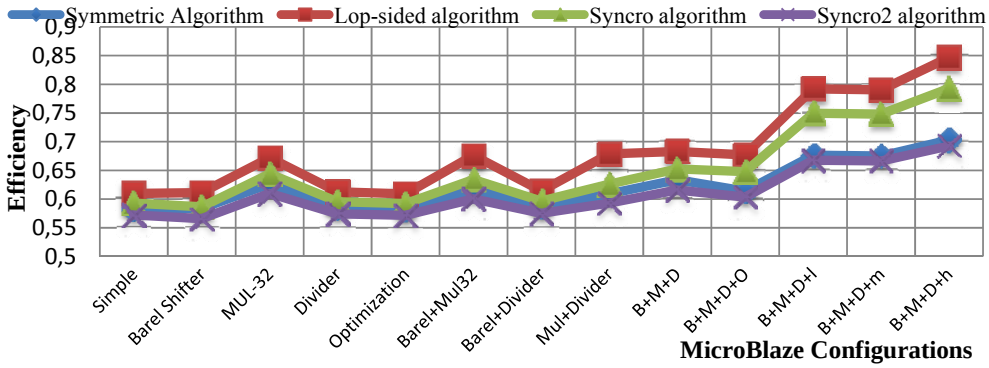
0.43W more than the single processor architecture in the same configuration), and respectively the minimum of this difference (0.134W) is achieved in the BMDO configuration. Thus, this configuration consumes with 17.6% more power and with 58.9% more logic resources than the single processor architecture, however it gains a performance boost of up to 35.4%.The BMDh configuration that was considered optimal, presents a 26.5% power consumption increase, a 57.7% increase in logic resources used and also the best performance increase (by 69.3%) comparing to the single processor architecture.

Therefore we obtained two optimal configurations, an optimal configuration in terms of power consumption (BMDO) and an optimal configuration in terms of performance (BMDh). Depending on the designer or application purpose, one of these two configurations is chosen. In our case the configuration considered optimal is the one with the best performance (BMDh) because our goal was to obtain better performance using two processors architecture.



Graph 3 Comparison of the dynamic power consumption of the two architectures

Ideally, the efficiency (efficiency = Speed Up / number of processors) of a system with two processors should be 1, which means that by adding a processor to the system should double the system performance, but unfortunately this is not achieved in this case. As expected, each parallelization method lends itself better or worse to a particular type of filter, however the lop-sided algorithm provides a balanced efficiency for all three algorithms, resulting a maximum efficiency of 0.85 for the B+M+D+h configuration. After a brief analysis of the effectiveness of each algorithms graph (shown in graph 4) we can easily see that the efficiency increases significantly with the enabling of the software optimizations, so we can conclude that these filtering algorithms were not fully parallelized.



Graph 4. Filtering algorithms efficiency (processor's configurations)

4. Conclusions

The best performance for the single processor architecture has been obtained for the configuration in which the hardware multiplier, the barrel shifter, the hardware divider and the high level software optimization have been activated. Therefore we have obtained a minimum filtering time of 1.66 s.

In order to achieve a smaller filtering time, we have chosen the architecture with two processors. Once created the multiprocessor architecture, we started the parallelization of the filtering algorithm resulting four versions, each with its advantages and disadvantages. After examining the data obtained for each version, we have concluded that the best parallelized filtering algorithm was the one called lop-sided (asymmetrical), which has reached a minimum filtration time of 0.98 s, thus achieving a speed-up rate of 1.69 and an efficiency rate of 0.84. We then analyzed how the power consumption and the number of logic cells were affected. Comment: for a 40.95% decrease in the execution time, the power consumption increased by 26.5% and the number of logic cells increased by 57.75% (sum of Register Slice and Slice LUT).Considering that the HW-SW design of multiprocessor systems based on FPGA IP Cores is new, it is necessary to examine very closely the available possibilities. By exploiting the advantages of this technology, we can implement new MPSoC architectures that offer an optimal performance/power consumption ratio.

Given the fact that a standard for these multiprocessor systems on chip does not yet exist, in addition to the solutions used in this paper, there are other improvements that could be added to this system. These improvements can be hardware related, consisting in the processors interconnection using FSL [8], the transmission of the processed data from one processor to another using xps_mailbox [8], or adding an external DDR memory which will allow the processing of larger color images. Other improvements can be software related, so these algorithms could be optimized by editing the part of the code that allows the

image to be read from the memory card in a way that the program code will fit entirely in the internal memory. Another improvement could be the usage of different methods of parallelizing the filtering algorithm.

REFERENCES

[1] *Charmaine Cooper Hussain*, "Xcell Journal Solutions for a programmable world", Forrest Couch,Vol 64, 2008

[2] "Virtex-5 Family Overview", Product Specification , DS100(v5.0), 6 February 2009

[3] "MicroBlaze Processor v7.20d", Reference Guide, UG081 v10.3, 26 October 2009

[4] *M. Thompson,* "FPGAs accelerate time to market for industrial designs", EE Times,July2004

[5] *Patrick Longa* , "IP core design", SYSC5603 (ELG6163) Digital Signal Processing Microprocessors, Software and Applications, 2006

[6] "EDK Concepts, Tools and Techniques", Guide to Effective Embedded System Design, UG683, 2 December 2009

[7] "Platform Specification Format Reference Manual", Embedded Development Kit, UG642, 24 June 2009

[8] *Vasanth Asokan* , "Designing Multiprocessor Systems in Platform Studio", Xilinx, WP262(v2.0), 21 November 2007

[9] *Mandeep Kaurand Vikas Sharma*, "Analysis of Various Algorithms for Low Power Consumption in Embedded System using Different architecture", International journal of Electronics Engineering, Vol 2,No.1, 2010, pp. 213-217

[10] *Chang, Chen, Wawrzynek, John and Brodersen, "*A High-End Reconfigurable Computing System" IEEE Computer Society, in IEEE Design and Test of Computers, Vol. 22, March 2005,pp. 114-125.

[11] *Hauck, Scott, Dehonand André, "*Reconfigurable computing", Morgan Kauffman Publishers, Burlington, Massachusetts, 2008

[12] *Taho Dorta, Jaime Jimenez, Jose Luis Martin* and *Armando Astarloa*, "Reconfigurable Multiprocessor Systems: A review", International Journal of Reconfigurable Computing 2010, article ID: 570279

[13] *Shaily Mittaland Niti,*" A Resolution for Shared Memory Conflict in Multiprocessor System-on-a-Chip", in International Journal of Computer Science Issues, Vol. 8, Issue 4, No 1, July 2011

[14] *C. Ferri, R.I. Bahar, T. Moreshet, A. Viescasand M. Herlihy*, "Energy Efficient Synchronization Techniques for Embedded Architectures", GLSVLSI'08, USA, 2008.

[15] *Paulina T. Nguyen*, "Bitmap File Format and Manipulation", OLED Display Engineering, San Jose, USA, April 2005

[16] *Peter Mattisand Spencer Kimball,* "GNU Image Manipulation Program: Convolution Matrix",Vol 1, section 2.8, England, July2007

[17] *Gonzalez and Woods* , "Digital image processing", 2nd edition, PrenticeHall, 2002. Chap 4 Sec 4.3, 4.4; Chap 5 Sec 5.1 – 5.3, pp 167-184 and 220-243

[18] *David Pellerin and Milan Saini*, "FPGAs Provide Acceleration for Software Algorithms", FPGA Journal, 2004

[19] *Bryan H. Fletcher*, "FPGA Embedded Processors – Revealing True Systems Performance", Embedded Systems Conference, Memec, San Francisco, California 2005

[20] *Derek Curd*, „Power Consumption in 65 nm FPGAs", White Paper: Virtex-5 FPGAs,WP246 February2007