ISSN (Online) : 2319 - 8753
ISSN (Print) : 2347 - 6710

International Journal of Innovative Research in Science, Engineering and Technology

Volume 3, Special Issue 3, March 2014

2014 International Conference on Innovations in Engineering and Technology (ICIET'14)

On 21st & 22nd March Organized by

K.L.N. College of Engineering, Madurai, Tamil Nadu, India

A Low Power and High Speed MPSOC Architecture for Reconfigurable Application

N.Janakiraman[#1],S.J.Udaya Kumar[#2],K.J.Sabarish[#3],S.Vimal Kandan[#4],P.Nirmal Kumar[#5],S.Selvakumar[#6]

[#1]Department of ECE, K.L.N. College of Engineering, Pottapalayam, Sivagangai dt., Tamilnadu, India

[#2] Department of ECE, K.L.N. College of Engineering, Pottapalayam, Sivagangai dt., Tamilnadu, India

[#3]Department of ECE, K.L.N. College of Engineering, Pottapalayam, Sivagangai dt., Tamilnadu, India

[#4]Department of ECE, K.L.N. College of Engineering, Pottapalayam, Sivagangai dt., Tamilnadu, India

[#5]Department of ECE, College of Engineering - Guindy, Anna University, Chennai, Tamilnadu, India.

[#6]Department of ECE, K.L.N. College of Engineering, Pottapalayam, Sivagangai dt., Tamilnadu, India

ABSTRACT - A multiprocessor system on chip that incorporates all the components inside the chip and used for multiprocessing application. MOLEN is a reconfigurable architecture based on co-operation between a general purpose processor and custom computing unit. MOLEN incorporates an arbitrary number of programmable units exposes the hardware to the programmers and Designers allows them to modify and extend the processor functionality at will. It solves a number of limitations of existing approaches, such as the opcode space explosion, and it requires only a one time extension of the instruction set to incorporate an almost unlimited number of reconfiguration functions per single programming space. For our paper we have done multiple surveys for the development of MOLEN architecture.

I. INTRODUCTION

Our survey is based on MOLEN course gain reconfigurable architecture. And MOLEN architecture consists of following blocks. They are Instruction fetch, Memory Multiplexer, Arbiter, Core processor and Reconfigurable unit. Reconfigurable unit consists of Microcode control unit and custom computing unit. The following explanations are given below.

INSTRUCTION FETCH: The program counter is a 32bit register that contains the address of a location in the Instruction Memory. PC always points to the next instruction in the memory that is to be fetched. The instruction memory is byte addressable. So every instruction takes up 4 bytes of memory.

Depending on the last instruction, the new program counter can be incremented by 4 or an address specified by a jump. PC incremented by 4 every time Because of the manner in which the Instruction memory is addressed. The output will produce the ALU unit. Pipelining register IF/ID Register It stores the (32bit) PC and the (32bit) instruction, and therefore it has to be 64 bit wide.

ARBITER**:** The arbiter controls the proper co-processing of the core processor and the reconfigurable processor by directing instructions to either of these processors. It arbitrates the data memory access of the reconfigurable and core processors. And it distributes control signals and the starting microcode address to the reconfigurable microcode unit.

RECONFIGURABLE DEVICE**:** It bridges the gap between the ASIC and Software programmed microprocessors by providing flexibility greater than ASIC and performance greater than software programmed microprocessor. Using reconfigurable hardware's (like FPGA) for computational purposes where the computation is done through functional elements called logical blocks that are configured through configuration bits.

Reconfigurable hardware coexisting with a core processor can be considered as a good candidate for speeding up processor performance.

Both FPGA and reconfigurable are used to speed-up the performance of various applications. Usually a reconfigurable block is combined with a general purpose microprocessor. In this way the processor maps the instructions that can be done efficiently to reconfigurable block and the microprocessor performs instructions that cannot be done efficiently in reconfigurable block. This reconfigurable block may be a commercial FPGA. The applications are, data encryption, video processing, network security and image generation.

MICROCODE**:** The introduction of microcode in the field of computer engineering has been a major push towards instruction set architectures and thereby introduced the notion of compatibility between current and future processor implementations. It allowed instructions that could be implemented directly in hardware to be included in architecture through emulation, i.e., substituting the instruction in question by a sequence of micro-operations.

While a vast majority of instructions in current architectures are hardwired, microcode is still being employed in those cases when it is not economical (chip area-wise) to implement an instruction directly into hardware, like I/O operations. In this chapter, we familiarize the reader with microcode by re-introducing microcode concepts and introducing 4 design principles. Furthermore, we Show the relationship between CISC, RISC, and VLIW architectures and horizontal and vertical microcode. Microcode was introduced as a collection of micro operations that can be arranged into any sequence.

In the remainder of this dissertation, we refer to microcode as a sequence of microinstructions that in their turn specify which micro-operation(s) to perform. In this section, we discuss the concepts associated with the microcode structure and some organizational techniques employed in microcode.

Programmability is provided by the inclusion of a programmable processor core that is intended to perform non-time-critical functions and that can be reused in successive embedded processor designs. Reconfigurability is provided by the inclusion of a reconfigurable hardware structure that is able to be configured as specialized hardware units intended to perform the time-critical functions.

This integration will allow the embedded processor to achieve both flexibility and adequate high-performance computing. The utilization of any reconfigurable hardware structure, e.g., field-programmable gate arrays (FPGAs), is bound to be less efficient performance-wise when compared to application-specific integrated circuits (ASICs). Still, the performance will be much higher than the programmable processor core.

It has several issues (long reconfiguration times, limited opcode space, and complex decoder hardware) that must be addressed in order to successfully build a truly extendable/flexible embedded processor. The main enabling technology is to reintroduce microcode to emulate both the execution and reconfiguration processes of the reconfigurable hardware. We have termed the resulting embedded processor as the reconfigurable microcoded embedded processor.

GENERAL DESCRIPTION**:** In its most general form, the proposed machine organization, which is augmented with a reconfigurable unit. In this organization, instructions are fetched from the main memory and are temporarily stored in the 'Instruction Fetch' unit. Subsequently, these instructions are fetched by the 'Arbiter' which decodes them before issuing them to their corresponding execution units. Instructions that have been implemented in fixed hardware are issued to the 'Core Processing Units', i.e., the regular functional units such as ALUs, multipliers, and dividers. The *set* and *execute* instructions relate to the reconfigurable unit and are issued to it accordingly. More specifically in our case, they are issued to the reconfigurable microcode unit or '½¹-code unit'. At this moment, it is only important to recognize that it provides fixed and pageable storage for reconfiguration and execution microcode that control the reconfigurable array. The proposed machine organization. Configuration and execution processes on the 'Custom Configured Unit' respectively. The loading of microcode to the 'reconfigurable microcode unit' is performed via the 'Arbiter' which accesses the main memory through the 'Data Fetch/Store' unit.

Similar to other load/store architectures, the proposed machine organization executes on data that is stored in the register file and prohibits direct memory data accesses by hardware units other than the load/store unit(s). However, there is one exception to this rule; the CCU is also allowed direct memory data access via the 'Data Fetch/Store' unit (represented by a dashed two-ended arrow). This enables the CCU to perform much better when streaming data accesses are required, e.g., in multimedia processing.

Finally, we introduce the exchange registers (XREGS) which are utilized to accommodate the mentioned input and output interface that is needed to communicate arguments and results between the implemented function(s) and the remainder of the application code. When only a small amount of data needs to be communicated, the register file suffices. However, by architecturally including the exchange registers, a more general communication framework can be provided in order to communicate an arbitrary number of arguments and results.

This paper is organised as follow. The section II discuss about block diagram. Section III discuss about survey of molen result. Section IV discuss about conclusion. Section V discuss about Scope. Section VI about reference.

II. BLOCK DIAGRAM:

The two main components in the MOLEN machine organization are the "Core Processor," which is a general-purpose processor (GPP), and the "Reconfigurable Processor" (RP).

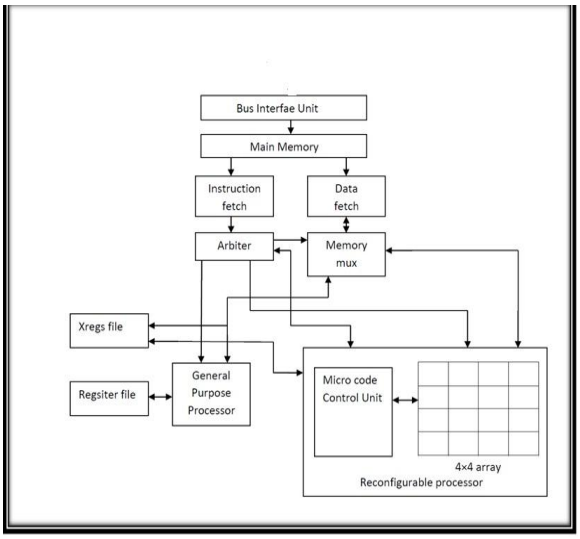

Fig. 1 Architecture block diagram

III. SURVEY ON MOLEN RESULTS

There are various paper survey about MOLEN CGRA and shown the following ideas,

1. The performance analysis results show that, we were able to reduce the superscalar machine execution clock cycles by 29% for the JPEG encoding benchmark. For the JPEG decoding benchmark, the reduction is about 32%.

Considering the MPEG-2 encoding benchmark, the superscalar machine cycles were decreased by about 32. In the case of the MPEG2 benchmark, this could be accomplished and the results show an additive behaviour.

This means that the reduction of the total number of execution clock cycles (when supporting both functions) is equal to addition of the reduction of total number of execution clock cycles of both functions when they are supported exclusively on the reconfigurable hardware structure.

Other simulation results other than the total number of execution clock cycles were also investigated, i.e., the total number of executed instructions, branches, loads, and stores. In most cases, the simulation results show that their numbers are also substantially decreased.

The number of executed instructions is decreased by about 66%. The number of executed branches is decreased by about 65%. The number of executed loads is decreased by about 46%. The number of executed stores is decreased by about 8%.

2. In order to determine the speedup gain, both the MOLEN AES and the reference AES code v2.2 are simulated. First, both sources are compiled by using Simple Scalar's compiler. Second, the binaries are simulated using the simulator that was described in the previous chapter.

Due to a low tolerance factor, the eject of the platform environment can be disregarded. It is depicted that the speedup gain is 4.9. In other words, it can be concluded that the AES encryption process can be 4.9 times speeded up on the MOLEN platform. As depicted the decryption process can be speedup up to 5.9 times. From the results presented in table 5.1 and in table 5.2.

One can see that the MOLEN architecture speeds up the decryption process up to 5.9 times. the cipher throughput performance is shown for both the MOLEN AES as well as for the reference code. As illustrated the throughput for the MOLEN AES is about 69 M bit/s. The throughput performances on the MOLEN platform for both cipher directions. The throughput of the reference code is 14.1 M bit/s and 11.8 M bit/s for respectively the encryption and the decryption process.

The throughputs of the AES that are implemented using higher programming language, varies from 4.60 M bit/s to 42.6 M bit/s. One can note, that the speed performance of some of these implementations is relatively high that compared to our 14.1 M bit/s of our reference AES implementation. The fact is that most of these implementations were optimized by utilizing large recalculated lookup tables and that for some implementation even the MMX instructions were utilized.

The following table shows the encryption and decryption performance result analysis.

TABLE I

The performance results for the encryption direction:

	MOLEN AES	**Reference Code**
Input Data size	29,276 bytes	29,276 bytes
Processor Clock Rate	1 GHz	1 GHz
Number of executed instruction	13,299,996	38,977,457
Tolerance factor	0.0075 %	0.0025 %
execution time	3,405,814 cycles	16,601,869 cycles
Speedup	4.9	1

TABLE II

The performance results for the decryption direction

	MOLEN AES	**Reference Code**
Input Data size	29,276 bytes	29,276 bytes
Processor Clock Rate	1 GHz	1 GHz
Number of executed instruction	13,314,131	51,075,628
Tolerance factor	0.0075 %	0.0020 %
execution time	3,381,644 cycles	19,797,636 cycles
Speedup	5.9	1

3. In MOLEN polymorphic paradigm suggest that the considered kernels can be speeded up to 300 times and one can incorrectly assume that the entire application can be speeded up to the same orders of magnitude. Table 3.1 shown below

4. For the DCT implementation, varying the ML-value between 100 cycles and 400 cycles is able to reduce the total number of clock cycles by between 30% and 25%.

Assuming our implementation (ML = 282) for the DCT results in a reduction of about 27%. For the VLC implementation, the reduction is between 8% and 3%. Allowing the CCU to be reconfigured to either the DCT or VLC implementations shows a decrease of about 30%. In this case, we have assumed an ML-value of 200 for the VLC implementation.

We can clearly see that the reconfiguration latencies is having an effect on the performance. This can be seen by adding the reduction of the only using the DCT and only the VLC implementations which is higher than 30%. Besides looking at the total number of clock cycles, we have also taken a look at some other metrics. In the cases, that the CCU can be configured to both the DCT and the VLC implementation, the following results were also obtained from the simulations: the total number instructions was reduced by 40.16%, the total number of branches was reduced by 28.55%, the total number of loads was reduced by 44.70%, and the total number of stores was reduced by 33.83%.

In the simulations we have intentionally left out the results of both the IDCT and VLC implementations as they did not provide much performance increases.

TABLE III

	MPEG 2 Encoder*			**MPEG 2 Decoder**		
	Theory	**Impl.**	**Impl. /th.**	**Theory**	**Impl.**	**Impl. /th.**
Car phone	2.85	2.64	93%	2.02	1.94	96%
Claire	2.99	2.8	94%	1.6	1.56	98%
Container	3.12	2.96	95%	1.68	1.63	97%
Tennis	3.37	3.18	94%	1.68	1.65	98%

The main reason is that they only constitute a small part of all the operations in the MPEG-2 encoding scheme. We can see that the biggest contributor to the overall performance gain is the SAD which is able to decrease the number of cycles by about 35% (in a more ideal case). Using our implementation (ML = 234) results in a reduction of the number of clock cycles by 20%. Allowing the CCU to be reconfigured to either the DCT or the SAD implementation shows a decrease of 32%. Here we observe that the result is additive suggesting that the reconfiguration and execution of both operations do not influence each other. Besides looking at the metric the total number of execution cycles.

IV. CONCLUSION:

In this survey, We presented a MOLEN Coarse gain reconfigurable architecture a custom computing machine based on the co-processor architectural paradigm. That allows the programmer/designer to modify and extend the processor functionality and hardware at will without architectural and design modifications. The proposal solves a number of limitations of existing approaches, such as the opcode space explosion, and it requires only a one time extension of the instruction set to incorporate an almost unlimited number of reconfiguration functions per single programming space. Finally we compared the various Molen architecture results.

V. SCOPE

To design a high speed and low power MPSoC architecture for reconfigurable application. We are going to compare our proposed architecture with MOLEN architecture. We are going to achieve 95% efficiency as outcome for our proposed architecture.

REFERENCE:

1. S. Vassiliadis, G. N. Gaydadjiev, K. Bertels, and E. M. Panainte. The molen programming paradigm. In Proc. Third Intl. Workshop on Systems, Architectures, Modelling, and Simulation (SAMOS'03), pp. 1–7, 2003.

2. S. Vassiliadis, S. Wong, and S. Cotofana. The MOLEN reconfigurable microcode-Coded Processor. In 11th intl. *Conf. FPL'01*, Springer-Verilog LNCS, vol. 2147, pp. 275–285, 2001.

3. S. Vassiliadis, G. Gaydadjiev , K. Bertels and E. Moscu Panainte , "The Molen Programming Paradigm," Proc. Third Int'l Workshop Systems, Architectures, Modelling, and Simulation, pp. 1-7, July 2003.

4. E. Moscu Panainte, K. Bertels, and S. Vassiliadis, "Compiling for The Molen Programming Paradigm," Proc. 13th Int'l Conf. Field
Programmable Logic and Applications (FPL), pp. 900-910, Sept. 2003

5. S. Vassiliadis, S. Wong, and S. Cotofana, "Microcode Processing Positioning and Directions," IEEE Micro, vol. 23, no. 4, pp. 21-30, July/Aug. 2003.

6. S. Wong, S. Vassiliadis, and S. Cotofana , "Microcoded Reconfigurable Embedded Processors", in Proceedings of 1st Workshop on System Architecture Modelling and Simulation (SAMOS2001), (Samos Island, Greece), July 2001 .

7. S. Vassiliadis, S. Wong, G. Gaydadjiev, K. Bertels, G. Kuzmanov, and E. M. Panainte, The molen polymorphic processor," IEEE Transactions on Computers, vol. 53, pp. 1363 - 1375, 2004.

8. J. Hauser and J. Wawrzynek, Garp: a mips processor with a reconfigurable coprocessor,"
in The 5th Annual IEEE Symposium on FPGAs for Custom Computing Machines, pp. 12 -21, April 1997.

9.S. Hauck, T. Fry, M. Hosler, and J. Kao, The chimaera reconfigurable functional unit," in The 5th Annual IEEE Symposium on FPGAs for Custom Computing Machines, pp. 87 - 96, April 1997

10. E. M. Panainte, The Molen Compiler for Reconfigurable Architectures. PhD thesis, TUDelft, June 2007.