Multiprocessor Architecture Design Using VHDL Structural Modeling

Mohamed EL KHAILI
SSDIA Laboratory, ENSET, Mohammedia
Hassan II University of Casablanca, Morocco
elkhailimed@gmail.com

***Abstract*—This paper presents design concept of multiprocessor architecture based on multiple Processor Element (PE) interconnected on one FPGA chip. Design methodology has been changing from schematic design to HDL based design. We use a VHDL structural and dataflow level design. Each module of the Processor Element is divided into smaller modules. All the modules in logical unit and Arithmetic and Logic Unit (ALU) design are realized using VHDL design. We use a basic algorithm configuration for Network-on-Chip Architecture with reconfigurable ability. This NoC reduces the complexity of the design and makes the layout of the NoC more flexible. Functionalities are validated through synthesis and simulation process. Multiprocessor Architecture (MA) using VHDL fulfills the needs for different high performance processing for applications such as image processing, path planning, numeric analysis and mathematic problem resolution.**

Keywords—component; Multiprocessor Architecture, Arithmetic and logic unit (ALU), FPGA Circuits, VHDL, System-on-Chip (SoC),Network-on-Chip (NoC), Parallel processing.

I. INTRODUCTION

Modern digital systems demand increasing electronic resources, so the multiprocessor platforms are a suitable solution for them. This approach provides better results in term of area, speed and power consumption compared to traditional uni-processor systems. Reconfigurable multiprocessor systems are particular type of embedded system, implemented using reconfigurable hardware (FPGA).

Multiprocessor System-on-Chip MPSoC represent an important trend in digital embedded electronic systems to reach Real-Time deadlines while overcoming other critical constraints such as power consumption and low area. MPSoC seem to be the better solution for complex systems. In the field of MPSoC, the reconfigurable multiprocessor system is a new and increasing trend.

In this paper, a Processor Element PE is presented and designed by a VHDL structural modeling. The Processor Element PE uses a Processing Unit already described in[13]. The PE includes also registers for storage and flags and logic system. This unit will be used in an implementation of a multiprocessor system on one FPGA chip (SoC). A Network-on-Chip NoC provides communications between PE's, memory and I/O Interfaces.

The Arithmetic and Logic Unit ALU is the part of the Processor Element PE that is involved with executing operations of an arithmetic or logical nature. All logic circuits in this units are entirely combinational (i.e. consists of gates with no feedback and no flipflops).The ALU is an extremely versatile and useful device since, it makes available facility for performing many different logical and arithmetic operations in order to get a complete set of instructions. ALU is a critical component of a microprocessor and is the core component of central processing unit[9][10].

Our project aimed to design and implement an FPGA-based multiprocessor architecture to speed up multiprocessor architecture research and easy parallel software simulation. This system had to be a flexible, inexpensive multiprocessor machine that would provide reliable, fast simulation results for parallel software and hardware development and testing.

In the following section, we present a context of the our project. The next sections provide a more detail description of Multiprocessor Architecture components (Processors and Interconnection Bloc). Results and perspectives are presented at the end before concluding this study.

II. CONTEXT OF THE PROJECT

Our project is focused on a multiprocessor hardware development and its implementation on FPGA. The FPGA-multiprocessor system have some advantages like :

- flexibility and reconfiguration,

- less time-to-market,

- less cost,

- and scalability.

The FPGA is the best choice in certain cases :

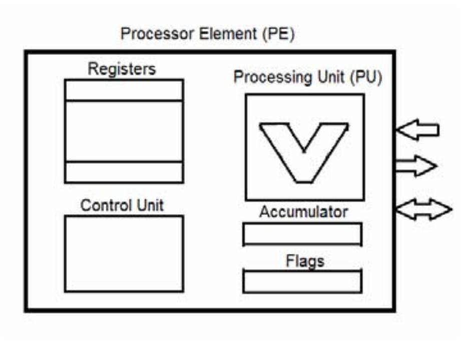
- Low-volume, mission-critical designs (e.g., telecommunication systems, radar and military applications);

- Rapid design of new, reconfigurable multiprocessor systems;

- Research field : new architecture, memory hierarchies, inter-processor communication, and so forth, can be developed;

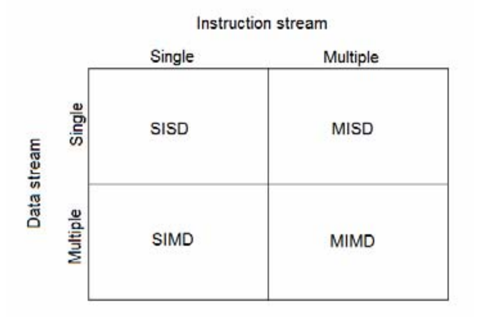
- Naturally-grown systems : systems that have to be able to grow in features depending on the stage of development.

In a multiprocessing system, all Processors Elements PE's may be equal, or some ones may be reserved for special purposes. A combination of hardware and operating system software design considerations determine the symmetry in a given system[11][13] [14]. For example, hardware or software considerations may require that only one particular Processor Element PE respond to all hardware interrupts, whereas all other work in the system may be distributed equally among PE's; or execution of kernel-mode code may be restricted to only one particular PE, whereas user-mode code may be executed in any combination of processors. Multiprocessing systems are often easier to design if such restrictions are imposed, but they tend to be less efficient than systems in which all PE's are utilized. M. Youssfi et al. proposed an emulator for multiprocessor system [13] [14]. This study is focused on a fine grained parallel architecture that has been largely studied in the literature and for which several parallel algorithms for scientific calculus were developed. From the theoretical point of view, each computational model has its motivations and its exciting proofs. In the practice, some models were technologically realized and served as real computational supports, but some others remain in their theoretical proposition state. They are not realized because of their technological complexities and to their very high production cost.



Basic presentation of Processor Element (PE)

Our project provides a hardware implementation of a multiprocessor system on one FPGA chip reducing development time and production cost. The target is a reconfigurable system that we present in the following [12].



Flynn's taxonomy of multiprocessor systems

A. SISD multiprocessing

In a Single-Instruction stream, Single-Data stream computer, one PE sequentially processes instructions; each instruction processes one data item. One example is the "von Neumann" architecture with RISC.

B. SIMD multiprocessing

In a Single-Instruction stream, Multiple Data stream computer, one PE handles a stream of instructions, each one of which can perform calculations in parallel on multiple data locations. SIMD multiprocessing is well suited to parallel or vector processing, in which a very large set of data can be divided into parts that are individually subjected to identical but independent operations. A single instruction stream directs the operation of multiple processing units to perform the same manipulations simultaneously on potentially large amounts of data.

For certain types of computing applications, this type of architecture can produce enormous increases in performance, in terms of the elapsed time required to complete a given task. However, a drawback to this architecture is that a large part of the system falls idle when programs or system tasks are executed that cannot be divided into units that can be processed in parallel.

Additionally, programs must be carefully and specially written to take maximum advantage of the architecture, and often special optimizing compilers designed to produce code specifically for this environment must be used. Some compilers in this category provide special constructs or extensions to allow programmers to directly specify operations to be performed in parallel (e.g., DO FOR ALL statements in the version of FORTRAN used on the ILLIAC IV, which was a SIMD multiprocessing supercomputer).

SIMD multiprocessing finds wide use in certain domains such as computer simulation, but is of little use in general-purpose desktop and business computing environments.

C. MISD multiprocessing

MISD multiprocessing offers mainly the advantage of redundancy, since multiple PEs perform the same tasks on the same data, reducing the chances of incorrect results if one of the units fails. MISD architectures may involve comparisons between PEs to detect failures. Apart from the redundant and fail-safe character of this type of multiprocessing, it has few advantages, and it is very expensive. It does not improve performance. It can be implemented in a way that is transparent to software. It is used in array processors and is implemented in fault tolerant machines.

Another example of MISD is pipelined image processing where every image pixel is piped through several hardware units performing several steps of image transformation.

D. MIMD multiprocessing

MIMD multiprocessing architecture is suitable for a wide variety of tasks in which completely independent and parallel execution of instructions touching different sets of data can be put to productive use. For this reason, and because it is easy to implement, MIMD predominates in multiprocessing.

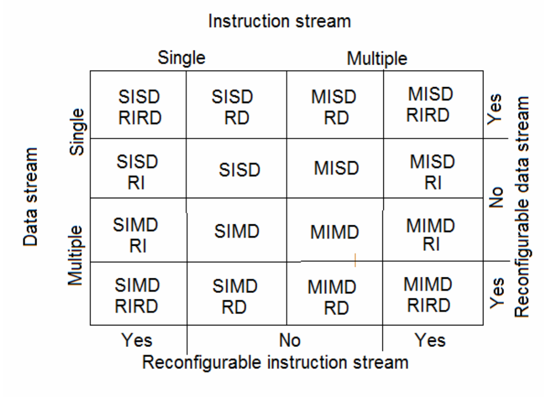
Processing is divided into multiple threads, each with its own hardware PE state, within a single software-defined process or within multiple processes. Insofar as a system has multiple threads awaiting dispatch (either system or user threads), this architecture makes good use of hardware resources.

MIMD does raise issues of deadlock and resource contention, however, since threads may collide in their access to resources in an unpredictable way that is difficult to manage efficiently. MIMD requires special coding in the operating system of a computer but does not require application changes unless the programs themselves use multiple threads (MIMD is transparent to single-threaded programs under most operating systems, if the programs do not voluntarily relinquish control to the OS). Both system and user software may need to use software constructs such as semaphores (also called locks or gates) to prevent one thread from interfering with another if they should happen to cross paths in referencing the same data. This gating or locking process increases code complexity, lowers performance, and greatly increases the amount of testing required, although not usually enough to negate the advantages of multiprocessing.

Similar conflicts can arise at the hardware level between PEs (cache contention and corruption, for example), and must usually be resolved in hardware, or with a combination of software and hardware (e.g., cache-clear instructions).

E. From Flynn's taxonomy to Göhringer taxonomy

Efficient On-Chip communication between cores is required to accommodate the increasing of the number of PE's in FPGA-based multiprocessors. Traditional bus-shared approach is not bandwidth efficient. Network-on-Chip NoC seems to be the best solution to interconnect cores. There are several studies on NoC use for FPGA-based multiprocessor system. Göhringer et al.[15] present a study of different reconfigurable communication architecture targeting FPGA-based multiprocessors systems.



		Instruction stream				
		Single		Multiple		Reconfigurable data stream
Data stream	Single	SISD RIRD	SISD RD	MISD RD	MISD RIRD	Yes
		SISD RI	SISD	MISD	MISD RI	No
	Multiple	SIMD RI	SIMD	MIMD	MIMD RI	
		SIMD RIRD	SIMD RD	MIMD RD	MIMD RIRD	Yes
		Yes	No		Yes	
		Reconfigurable instruction stream				

New taxonomy for reconfigurable FPGA-based systems proposed by Göhringer et al.[]

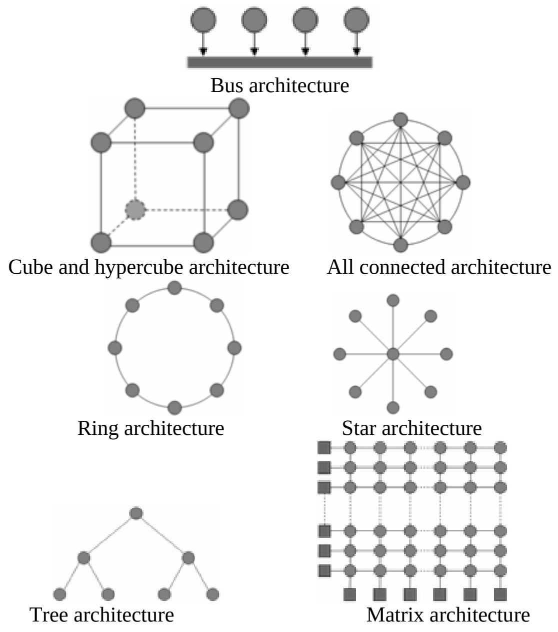
F. Interconnection Topology

A network allows communication between PEs, Output/Input devices and memory. Topology has a significant influence on the properties of the system. Almost all topologies were used, one can nevertheless illustrate some (figure 4).

G. Shared Memory Systems

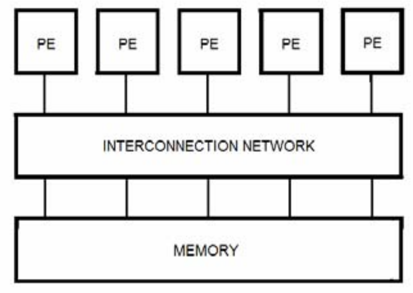
We can classify Multiprocessors Architectures in accordance with the memory architecture. There are two types : shared-memory (figure 5) and distributed memory (figure 6).

In shared-memory systems, all PE share the same memory resources. In this case, all changes made by a PE to a given memory location become visible to all the other PE's. The shared-memory architectures are poorly scalable because of the limited band-width of the memory. They require synchronization mechanisms such as semaphores, barriers and locks since no explicit communication exists.[17]



Multiprocessor Architecture Typologies

In shared-memory architecture, different processes can easily exchange information through shared variables; however, it requires careful handling of synchronization and memory protection.

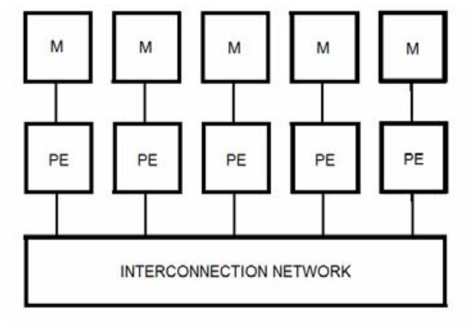


Shared memory multiprocessor systems

In distributed-memory architecture, each PE has its own private memory. Therefore, one PE cannot read directly in the memory of another PE. Data transfers are implemented using message-passing protocols. Distributed-memory architecture are more scalable since only the communication medium may be shared among PE's.

The communications infrastructure is required in order to connect PE's and theirs memories and allows the exchange of information.

Our Multiprocessors Architecture is based on shared-memory and shared-bus system because it is highly flexible in order to satisfy changes deriving from application environment. [21]



Distributed memory multiprocessor systems

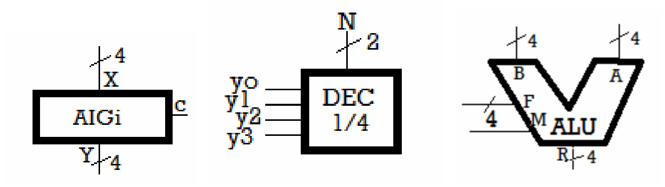
III. Processor Element (PE) Structure

In each PE, the ALU works in conjunction with the register array for many of these, in particular, the accumulator and flag registers. The accumulator holds the results of operations, while the flag register contains a number of individual bits that are used to store information about the last operation carried out by the ALU. More on these registers can be found in the register array section. In this part, we present the Processor Element structure with VHDL description.

A. Processing Unit (PU) structure [17]

Processing Unit consists of three blocks for different operations:

- 4-bit Arithmetic and Logic Unit ALU;
- Switching system AIGi;
- 1/4 Decoder DEC.



Bloc systems used in Processing Unit architecture

1) Arithmetic and Logic Unit (ALU) Description

An ALU which was widely used when only TTL integrated circuits were available: 74181, a 4-bit ALU. This circuit is intended to be connected in cascade to treat other integers greater than 4 bits size. It is the basis of UAL Alto and many models PDP-11 (16 bits)[1][7][9][10].

a) Arithmetic operations

These operations are performed by constructs of logic gates, such as half adders and full adders. While they may be termed 'adders', with the aid of they can also perform subtraction via use of inverters and 'two's complement' arithmetic. A binary adder-subtractor is a combinational circuit that performs the arithmetic operations of addition and subtraction with binary numbers. Connecting n full adders in cascade produces a binary adder for two n-bit numbers.

TABLE I. THE TRUTH TABLE OF THE TTL 74181 CIRCUIT ACCORDING TO THE INPUT M AND F VALUES

Function	M = 1	M = 0
F	**Logic Operation**	**Arithmetic Operation**
0000	R = not A	R = A
0001	R = not (A or B)	R = A or B
0010	R = (not A) and B	R = A or (not B)
0011	R = 0	R = - 1
0100	R = not (A and B)	R = A + (A and (not B))
0101	R = not B	R = (A or B) + (A and (not B))
0110	R = A xor B	R = A - B - 1
0111	R = A and (not B)	R = (A and (not B)) - 1
1000	R = (not A) or B	R = A + (A and B)
1001	R = not (A xor B)	R = A + B
1010	R = B	R = (A or (not B)) + (A and B)
1011	R = A and B	R = (A et B) - 1
1100	R = 1	R = A + (A << 1)
1101	R = A or (not B)	R = (A or B) + A
1110	R = A or B	R = (A or (not B)) + A
1111	R = A	R = A - 1

b) Logical operations

Further logic gates are used within the ALU to perform a number of different logical tests, including seeing if an operation produces a result of zero. Most of these logical tests are used to then change the values stored in the flag register, so that they may be checked later by separate operations or instructions. Others produce a result which is then stored, and used later in further processing.

2) Switching system AIGi

AIGi is a tri-state device. If c = '0', the output Y coded on 4 bits, is set to high impedance (HZ). If c = '1', the output Y assumes the value of the input X, as coded on 4 bits.

3) Decoder DEC

Dec 1/4 is a decoder 1 from 4, receives a number N coded on 2 bits and provides an active output at ‘1’ level among the four outputs.

4) VHDL codes

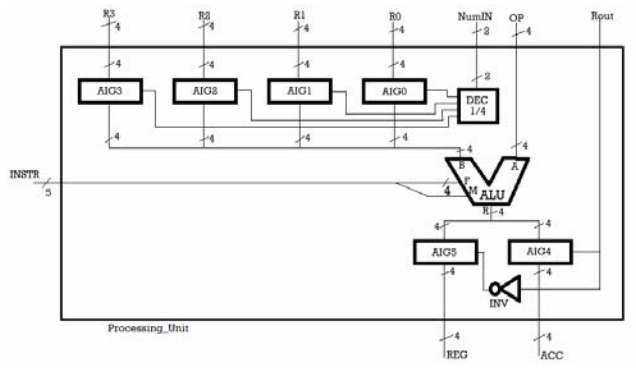
The VHDL descriptions of Aigi, Dec 1/4 and ALU are as follows [2][3][5][6] :

```
Entity aigi is
Port (x : in std_logic_vector(3 downto 0); c : in std_logic;
          Y : out std_logic_vector(3 downto 0));
End aigi ;

Architecture str_aigi of aigi is
Begin
          Y <= x when c=’1’  else “ZZZZ”;
End str_aigi;

Entity dec_1_4  is
Port (n : in std_logic_vector(1 downto 0);
          Y : out std_logic_vector(3 downto 0));
End dec_1_4  ;
Architecture str_dec of dec_1_4 is
Begin
```

```
    with n select
              Y <= "0001" when 0 ,
                  <="0010" when 1,
                  <= "0100" when 2,
                  <= "1000" whenothers;
End str_dec;

Entity alu  is
Port (a,b,f : in std_logic_vector(3 downto 0); m : in std_logic;
              r : out std_logic_vector(3 downto 0));
End alu  ;

Archticture str_alu of alu is
Begin
--The same implementation described in [9]
End str_alu;
```

B. Processing Unit (PU) Description [17]

The processing unit receives the number of Ri input to switch to the operand B of the ALU and the instruction code to execute. The ALU provides a result that will be switched to an accumulator ACC or a register REG. The data OP is recovered from the accumulator ACC. The signals NumIN, Rout and Rout are provided from a logic block that is not part of our study.



Processing Unit Architecture

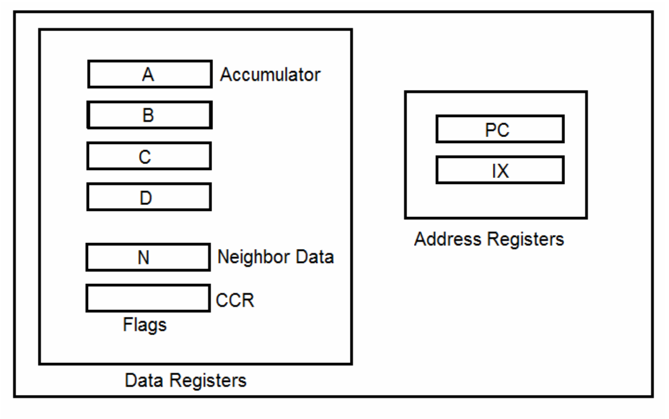
VHDL program

```
Entity Processing_unit  is
Port (instr : in std_logic_vector(4 downto 0);
              R3,R2,R1,R0,OP : in std_logic_vector(3 downto 0);
              numIn : in std_logic_vector(1 downto 0);
              Rout : in std_logic;
              Reg, Acc : out std_logic_vector(3 downto 0);)
End Processing_unit  ;

Architecture str_ Processing_unit  is

Component aigi
Port (x : in std_logic_vector(3 downto 0); c : in std_logic;
              Y : out std_logic_vector(3 downto 0))
End Component  ;

Component  dec_1_4
Port (n : in std_logic_vector(1 downto 0);
              Y : out std_logic_vector(3 downto 0))
End Component  ;

Component  alu
Port (a,b,f : in std_logic_vector(3 downto 0); m : in std_logic;
              r : out std_logic_vector(3 downto 0))
End Component  ;

for Aig0, Aig1,  Aig2,  Aig3,  Aig4,  Aig5   : aigi use entity
work.aigi(str_aigi);
for dec   : dec_1_4  use entity work. dec_1_4  (str_ dec_1_4  );
for alu_4 : alu use entity work.alu(str_alu);

signal R_alu, B_alu, co : std_logic_vector(3 downto 0) ;

Begin
   Aig0 : aigi port map (R0,co(0),B_alu);
   Aig1 : aigi port map (R1,co(1), B_alu);
   Aig2 : aigi port map (R2,co(2), B_alu);
   Aig3 : aigi port map (R3,co(3), B_alu);
   Aig4 : aigi port map (R_alu,Rout,Acc);
   Aig5 : aigi port map (R_alu,not(Rout),Reg);
   dec : dec_1_4  port map (NumIn,co);
  alu_4   :   alu   port   map   (op,B_alu,Instr(3   downto
0),Instr(4),R_alu);
End str_ Processing_unit  ;
```

C. PE Configuration

Each small module is simulated in separate test benches before the all are integrated and tested as a whole [4][8].

Each PE has a set of registers to manipulate data and address. N is a virtual register who receive a data from one of the four neighbors PE.



Processor Element Registers structure

A PE can execute arithmetic instructions, logic instructions and special instructions (jump, call, jccr, nop). PE structure is like a RISC machine.

Since the first processor architecture, the quest for higher performance has been present in every computer model and architecture. This has been the driving force behind the introduction of every new architecture or system organization. There are several ways to achieve performance: technology advances, better machine organization, better architecture, and also the optimization and improvements in compiler technology.

By technology, machine performance can be enhanced only in proportion to the amount of technology improvements and this is, more or less, available to everyone. It is in the machine organization and the machine architecture where the skills and experience of computer design are shown. RISC deals with these two levels. The work that each instruction of the RISC machine performs is simple and straight forward. Thus, the time required to execute each instruction can be shortened and the number of cycles reduced.

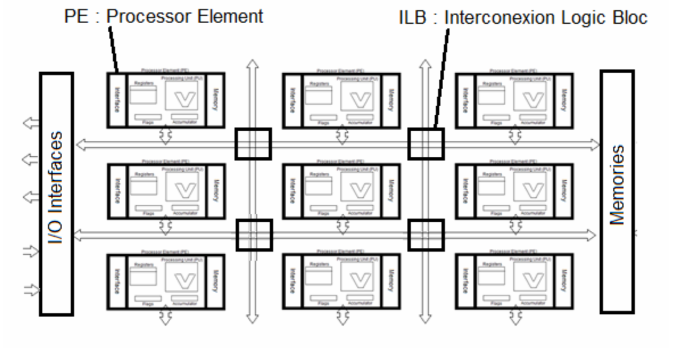
Typically the instruction execution time is divided in five stages, machine cycles, and as soon as processing of one stage is finished, the machine proceeds with executing the second stage. However, when the stage becomes free it is used to execute the same operation that belongs to the next instruction. The operation of the instructions is performed in a pipeline fashion, similar to the assembly line in the factory process. Typically those five pipeline stages are: Instruction Fetch, Instruction Decode, Execute, Memory Access and Write Back.

D. I/O Interfaces Management

Each PE must be connected to a matrix of Multiprocessor Architecture to exchange data and addresses. The Interconnection Logic Bloc is the bus for transmitting and receiving information. The I/O Interfaces are programmable devices able to connect PE's to a grid via ILB.

IV. INTERCONNECTION LOGIC BLOCS ILB

The ILB provide links and buses to realize all communications between PE's, shared memory and I/O Interfaces. We use a reconfigurable Network-on-Chip NoC architecture. Asghari et al. introduce the design process based on a reconfigurable network architecture and present a discussion about Algorithm Configuration to complete NoC design. This approach is selected for our Multiprocessor Architecture.

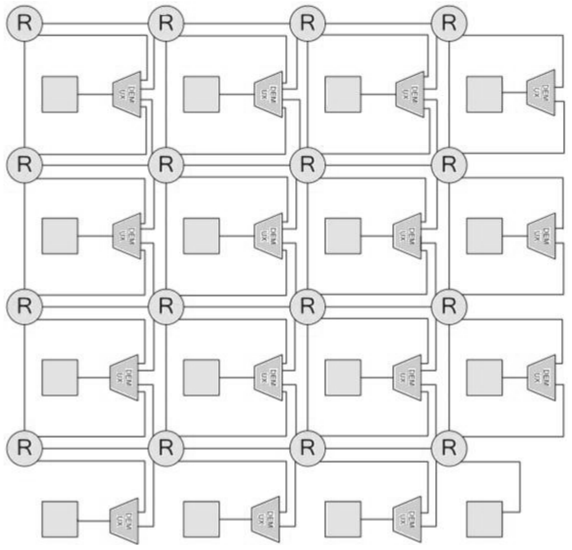


A reconfigurable machine using a Multi Processing Unit

A. NoC Reconfigurable Architecture

NoC Reconfigurable Architecture is presented in Figure 9. Let's take for example a 4 × 4 grid. This structure is based on two-dimensional mesh [8] that its PE's are connected to fixed operators. The difference is that each operator holds 8 ports which makes it possible operator's connectivity to more than 4 cores of PE. Accordingly, as shown in the Figure, one PE core gets. [18] [19] [20] [21]

Connected to all its adjoining operators. Besides, the multiplexer is introduced to the links connected to PE's and operators. Therefore, As the operator of one core PE is connected in order to change according to a different connectivity relation which is presented as the result of Algorithm Configuration. After obtaining the connectivity relation between the PE's cores and operators, we can set all multiplexers so they can select the operator, and each PE core is connected and the final structure of the network is obtained.



NoC Reconfigurable Architecture

B. Design Process

Asghari et al. [18] propose a process for designing NoC based on three steps:

Step 1 : Specify the size of NoC reconfigurable architecture

Step 2 : Specify the location of each PE and operator and the PE's connected to it.

Step 3 : Reconfigure NoC Reconfigurable Architecture according to the obtained data in the second step.

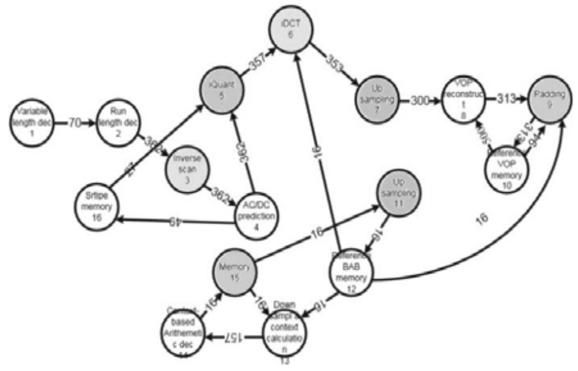
Input of the design process is the adjoining matrix which is obtained by one application and provides the data which PE would interact with other PE, and the size of the relation between two PE is distinct. And according to the number of PE's in a specific application, we can determine NoC reconfigurable architecture which is appropriate for this purpose.

Then, According to the provided information by the adjoining matrix, we run the proposed Algorithm for changing the shape. We therefore attain the connectivity relation between the PE's and the operators and the position of each PE in a reconfigurable NoC architectures. Finally, we reconfigure NoC reconfigurable architecture in such a way that the multiplexer can be configured according to the information obtained in previous stages we set.

C. *The Algorithm Configuration for NoC Reconfigurable Architecture*

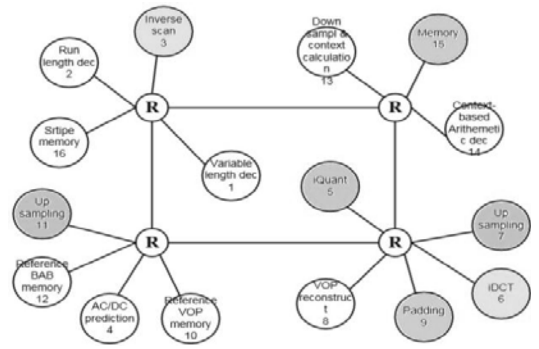
1) *Description of the problem[18]*

A functional relation graph ACG= G(V, E, W) is a weight-oriented graph consisting of a set of vertex V and a set of E – directed edges. Each edge v_i € V denotes a core. Each directed edge $e_{i,j}$ € E implies that there v_i is a relationship v_j between edge and edge v_j. And the weight of the edge $e_{i,j}$ implies the size of the relation between edges $e_{i,j}$ and v_i and v_j . Asghari et al. applied the ACG for an activity, called Video Objective Plane Decoder (VOPD) which is shown in Figure 11.



Example of ACG applied to VOPD by Ashgari et al.

A Topology Graph TG = G(R, P, L) is an undirected graph consisting of a set of R operators, a set of PE's and a set of L links. Each PE p_i € P is connected to a carrier through a link, and every carrier is connected to several operators through other links. One example of TG for VOPD is shown in Figure 12.



Example of TG applied to VOPD by Ashgari et al.

Regarding the above definition, the design of the network is described as according to the connectivity relation and connectivity sizes in ACG, is obtained to TG satisfaction:

- the performance of TG should be maximum ,
- the average delay of data transmission should be minimal,
- Power consumption should be minimal.

In this study, we aimed to lower power through Algorithm Configuration for NoC application. And the proposed energy consumption model is applied for calculating the power consumption of the network. The consumed E power for data transmission can be calculated as follows [18]:

a) $E = E_s + E_l$

Which E_s, E_l implies the power consumption by carriers and links in a r elative way.

The $E_{bit}^{i,j}$ consumed power due to transmitting one bit of operator r_i to operator r_j is as follows:

b) $E_{bit}^{i,j} = (h_{i,j} + 1) \times E_{sbit} + h_{i,j} \times E_{lbit})$

Which $h_{i,j}$ implies on hop between the operators and r_i and r_j , E_{sbit}, E_{lbit} implies on the consumed power by each operator on each link. The total network power consumption is as follows:

c) $E_t = \sum_{i-1,j-1}^{i-N_r,j-N_r} C_{i,j} \times E_{bit}^{i,j}$

which $C_{i,j}$ implies the relation of the carrier r_i to carrier N_r and r_j implies on the number of the operators.

2) *Algorithm Configuration*

As it is mentioned above, two phases before configuration of Reconfigurable NoC Architecture are:

- Determining the size of NoC Reconfigurable Architecture
- Determining the connectivity relation between operators and PE's and placing each PE in its right place.

In the first phase, select an NoC Architecture with reconfigurable ability for a specific purpose. We use the principle which the number of candidate’s opportunities for PE's in NoC Reconfigurable Architecture should be as near as possible to the number of PE's at work. In addition, the sizes of the two dimensions should be very different. This thought is carried out like a dummy code shown below in Figure 13:

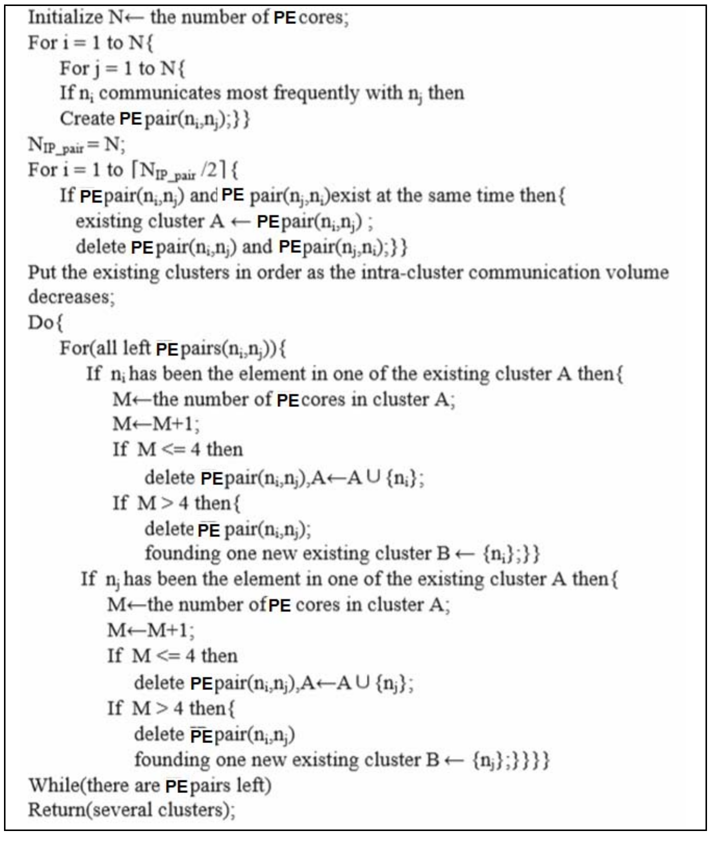
```
Initialize N← the number of PE cores, x ← 1, y ← 1, $g_{min}$ ← 1;
While ($g_{min}$ == 1) {
    For x = 1 to N{
       For y = 1 to N{
         If N == x*y && x >= y && (x-y)/x <= 1/3 && (x-y)/x < $g_{min}$ then
             $g_{min}$ ← (x-y)/x;}}
    N←N+1;}
Return(x, y);
```

Pseudo code to specify the size of the reconfigurable ability for NoC architecture

In the pseudo code, inequality is applied to reduce the distance between the sizes of two dimensions. However, we can also determine the distance according to different needs. Since the on-chip resources are limited, we can achieve the aim to increase the efficiency of resource use of the chip through this principle.

In the second step, we investigated the connectivity relation between PE's and operators, and are looking for a right place for each PE. Since NoC reconfigurable architecture is based on two dimensional mesh in which specific areas are for PE's and operators which there is no need for mapping for a specific place. And at the same time, complexity of the design reduces. For improving the performance of the network, we do our best in order to connect PE's which always had connection with each other through one operator. We introduce the category concept for expressing the PE's which have connection. And we place the group in the same category. After dividing all the PE's into several categories according to the proposed algorithm, we determine the method of placing the categories based on connectivity relation among categories in order to reduce the total power consumption of the network. The dummy code of figure 14 indicates the division process of PE's.

After EP's are divided into several categories, we place each core of our PE in a suitable location in the NoC reconfigurable architecture and connect each PE to one appropriate operator in a way that the final network becomes optimized. With the above steps, the communication within the cluster is the main connectivity. To reduce power consumption, we should connect PE's in the clusters which have the highest correlation within clusters in the first place. The selected operator to connect the PE's is the central operator. The definition of central operator for networks in different size. Then, we focus on the relationships within the group. We look for a category which is always in connection with proper categories, and determine the position of each PE in category and operator. Each PE is connected in order to reduce power consumption. Finally, we repeat the above steps until all the categories are placed.



```
Initialize N← the number of PE cores;
For i = 1 to N{
    For j = 1 to N{
    If $n_i$ communicates most frequently with $n_j$ then
    Create PE pair($n_i$,$n_j$);}}
$N_{IP\_pair}$ = N;
For i = 1 to ⌈$N_{IP\_pair}$ /2⌉{
    If PE pair($n_i$,$n_j$) and PE pair($n_j$,$n_i$)exist at the same time then{
      existing cluster A ← PE pair($n_i$,$n_j$) ;
      delete PE pair($n_i$,$n_j$) and PE pair($n_j$,$n_i$);}}
Put the existing clusters in order as the intra-cluster communication volume
decreases;
Do{
    For(all left PE pairs($n_i$,$n_j$)){
        If $n_i$ has been the element in one of the existing cluster A then{
            M←the number of PE cores in cluster A;
            M←M+1;
            If M <= 4 then
                delete PE pair($n_i$,$n_j$),A←A ∪ {$n_i$};
            If M > 4 then{
                delete PE pair($n_i$,$n_j$);
                founding one new existing cluster B ← {$n_i$};}}
        If $n_j$ has been the element in one of the existing cluster A then{
            M←the number of PE cores in cluster A;
            M←M+1;
            If M <= 4 then
                delete PE pair($n_i$,$n_j$),A←A ∪ {$n_j$};
            If M > 4 then{
                delete PE pair($n_i$,$n_j$)
                founding one new existing cluster B ← {$n_j$};}}}}
While(there are PE pairs left)
Return(several clusters);
```

Pseudo code to divide PE's into clusters

V. Simulation Results

Design is verified through simulation, which is done in a bottom-up fashion. Each small module is simulated in separate test benches before the all are integrated and tested as a whole [4][8]. All operations available in the design are tested with the various inputs. The sequence of operations done in the simulation is addition. The results of operation on the test vectors are manually computed and are referred to as expected result.

VI. Future Work

The current implementation is a prototype that represents a proof-of-concept. Next step will be devoted to the development of a programming model to implement some processing approaches on a multiprocessor platform. The remainder of this work sketches the most important hardware and software issues. On the hardware side, the current VHDL code is to be enhanced by all the required interfaces such as a LAN-connection and USB interfaces for transferring data, interfacing the peripherals and connecting a CAN bus.

On the software side, the next steps are as follows: First of all, the existing software functionalities have to be ported to the new system. Then, the distribution of the data and communication objects, i.e., the memory map, should be optimized with respect to the system's runtime behavior.

After the completion of the steps mentioned above, a final field study has to prove both sufficient processing speed and fulfillment of the required timing behavior.

VII. Conclusions

In this paper, we have proposed efficient VHDL behavioral coding verification method for a Multiprocessors Architecture implementation. We have also proposed a VHDL program using different design levels. Reconfigurable ability is reached using NoC Architecture. Xilinx chip provides all element to implement our architecture easily.

Our proposals have been implemented in Verilog and verified using Xilinx ISE 14 analyzer. We have reduced the number of bus lines and all the designs have been implemented and tested. This Multiprocessors Architecture design using VHDL is successfully designed, implemented, and tested.

In the future work, we are conducting further researches that integrate a software system using standard interfaces. Several areas of application will be treated for validation of parallel processing algorithms and retching high performances as speed and a real-time constraints.

Reference

[1] Brown S., Vranesic Z "Fundamental of Digital Logic Design with VHDL" McGraw Hill, 2nd Edition.

[2] Bhasker J, "A *VHDL Primer"*, P T R Prentice Hall, Pages 1-2, 4-13, 28-30.

[3] Jiang Hao, Li Zheying, "FPGA design flow based on a variety of EDA tools" in *Micro-computer information,* 2007(23)11-2:201-203.

[4] Xilinx, Inc. Xilinx Libraries Guide, 2011.

[5] *A Comparative Study",* In Proce. of the *IEEE Student Conference on Electrical, Electronics and Computer Sciences 2012,* 1-2 Mar 2012, NIT Bhopal, India.

[6] Douglas L. Perry, *VHDL,* McGraw-Hill, 1995.

[7] “Digital Integrated Circuits” by Jan M. Rabaey , Anantha Chandrakasan and Borivoje Nikolic

[8] Jiang Hao, Li Zheying, “FPGA design flow based on a variety of EDA tools” in *Micro-computer information*, 2007(23)11-2:201-203.

[9] Rupali Jarwal, Ulka Khire “4-Bit Arithmetic And Logic Unit Design Using Modelling In VHDL” International Journal of Engineering Research & Technology (IJERT) Vol. 2 Issue 3, March - 2013

[10] Navdeep Kaur, Neeru Malhotra, « FPGA Implementation Of ALU Using BIST » International Journal of Engineering Research & Technology (IJERT) Vol. 2 Issue 3, March - 2013

[11] Hauck, Scott and Dehon, André. 2008. Reconfigurable computing. Burlington, Massachusetts, USA : Morgan Kauffman, 2008.

[12] Sumedh.S.Jadhav, C.N.Bhoyar, « FPGA Based Embedded Multiprocessor Architecture » International Journal of Electrical and Electronics Engineering (IJEEE), Vol-1, Iss-3, 2012

[13] J. Elmesbahi, O. Bouattane, A. Raihani, M. El khaili, A. Rabaa, "Parallel algorithms for quadtree representation of points and curvilinear data", Systems Analysis Modeling Simulation, Vol. 33, Num. 4, pp. 479-494, 1998

[14] M. YOUSSFI, O. BOUATTANE, M. O. BENSALAH , “A Massively Parallel Re-Configurable Mesh Computer Emulator: Design, Modeling and Realization » J. Software Engineering & Applications, January 2010, 3: 11-26

[15] D. Göhringer, T. Perschke, M. Hübner and J. Becker , “A taxonomy of reconfigurable single/multiprocessor Systems-on-Chip » International Journal of Reconfigurable Computing, Article ID 570279, vol. 2009, 11 pages, 2009

[16] T. Dorta, J. Jimènez, J.L. Martin, U. Bidarte and A. Astarloa , “Reconfigurable Multiprocessor Systems : A Review » International Journal of Reconfigurable Computing, Article ID 395018, vol. 2010, 10 pages, 2010

[17] Mohamed EL KHAILI, « 4-Bit processing unit design using VHDL strucural modeling for multiprocessor architecture » IRACST – Engineering Science and Technology: An International Journal (ESTIJ), Vol. 4 No. 5, October 2014

[18] A. Asghari, A. A. Zoraghchian and M. Tarik, « Presntation of an algorithm configuration for Netword-on-Chip architecture with reconfiguration ability » International Journal of Electronics Communication and Computer Engineering (IJECCE), Vol. 5 Issue 5, 2014

[19] Z. E. Alaoui Ismaili and A. Moussa, « Self-Partial and Dynamic Reconfiguration Implementation for AES using FPGA», IJCSI International Journal of Computer Science Issues, Vol. 2, 2009

[20] I. R. Quadri, S. Meftali, J-L Dekeyser. « A Model based design flow for Dynamic Reconfigurable FPGAs». International Journal of Reconfigurable Computing, Hindawi Publishing Corporation, 2009.

[21] M. Monchiero, G. Palermo, C. Silvano, O. Villa, «Exploration of Distributed Shared Memory Architectures for NoC-based Multiprocessors», International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation, July 2006. IC-SAMOS 2006.

AUTHORS PROFILE

Mohamed EL KHAILI received Bachelor’s Degree of Electronics Engineering from ENSET (Ecole Normale Supérieure de l’Enseignement Technique) of Mohammedia,Morocco in 1992 and received a Master’s Degree of science in Information Processing (1994) and Doctorat in Image Processing (1998). He is a Assistant Professor at ENSET (Ecole Normale Supérieure de l’Enseignement Technique) of Mohammedia, Morocco where his field of specialization is Electronics Engineering, Computer Sciences, Information Processing and Embedded Systems.