مجلة جامعة تشرين للبحوث والدراسات العلمية _ سلسلة العلوم الهندسية المجلد (30) العدد (5) 2008
Tishreen University Journal for Research and Scientific Studies - Engineering Sciences Series Vol. (30) No. (5) 2008

Design and Modeling of Systolic Array Based on VHDL and FPGA

Dr. Hassan Al-Ahmad [*]
Dr. Hasan Albustani [**]
Moatassem Ibrahim [***]

(Received 7 / 4 / 2008. Accepted 13 / 5 / 2008)

□ ABSTRACT □

Systolic array design based on Field Programmable Gate Array (FPGA) can be adapted to efficiently resolve a wide spectrum of computational problems; parallelism, which is also naturally explored in systolic array and in implementing this design in FPGA, allows the redefinition of the interconnections and operations even during run time (dynamically).We have designed 2D systolic array architecture that implements the matrix multiplication algorithm. A VHDL for this design is applied to Xilinx FPGA. This design would be faster than the software of any alternative algorithm.

Keywords: Systolic Array, FPGA, VHDL, Parallel Processing, Matrix Multiplication, Digital Synthesis, Computer Architecture, Modeling.

[*]**Assistant Professor, Department of Computers and Auto-Control, Faculty of Mechanical and Electrical Engineering, Tishreen University, Lattakia, Syria.**
[**]**Assistant Professor, Department of Communications and Electronics, Faculty of Mechanical and Electrical Engineering, Tishreen University, Lattakia, Syria.**
[***]**Postgraduate Student, Department of Computers and Auto-Control, Faculty of Mechanical and Electrical Engineering, Tishreen University, Lattakia, Syria.**

مجلة جامعة تشرين للبحوث والدراسات العلمية _ سلسلة العلوم الهندسية المجلد (30) العدد (5) 2008
Tishreen University Journal for Research and Scientific Studies - Engineering Sciences Series Vol. (30) No. (5) 2008

تصميم ونمذجة المصفوفة الانقباضية اعتماداً على لغة VHDL وتقنيةFPGA

الدكتور حسن الأحمد*
الدكتور حسن البستاني**
معتصم ابراهيم***

(تاريخ الإيداع 7 / 4 / 2008. قُبِل للنشر في 2008/5/13)

□ الملخّص □

يؤدي تصميم المصفوفة الانقباضية اعتماداً على تقنية الـ FPGA إلى وضع حل فعال لِطيف واسع من المشاكل الحسابية، حيث تسمح المعالجة التفرعية الموجودة أصلاً في المصفوفة الانقباضية والمُحققة بتقنية الـ FPGA بإعادة التهيئة حتى أثناء زمن التشغيل (ديناميكياً).

قمنا بتصميم مصفوفة انقباضية ثنائية البعد تنفذ خوارزمية ضرب مصفوفتين، ثم تمت نمذجة هذا التصميم باستخدام لغة وصف الكيان الصلب للدارة المتكاملة العالية السرعة VHDL، وتم تحقيق النمذجة على لوحة مصفوفة البوابة القابلة للبرمجة من عائلة Xilinx .

إن زمن التنفيذ لهذا التصميم أسرع بكثير من أي برنامج يقوم بتنفيذ أي خوارزمية بديلة لضرب مصفوفتين.

الكلمات المفتاحية: المصفوفة الانقباضية، المعالجة التفرعية، ضرب المصفوفات، التركيب الرقمي، بنية الحاسب،النمذجة، FPGA، VHDL.

* مدرس– قسم هندسة الحاسبات والتحكم الآلي– كلية الهندسة الميكانيكية والكهربائية– جامعة تشرين– اللاذقية–سورية.
** مدرس–قسم الاتصالات والالكترونيات– كلية الهندسة الميكانيكية والكهربائية– جامعة تشرين– اللاذقية–سورية.
*** طالب دراسات عليا (ماجستير)– قسم هندسة الحاسبات والتحكم الآلي– كلية الهندسة الميكانيكية والكهربائية– جامعة تشرين– اللاذقية–سورية.

Introduction:

Dealing with real-time constraints, the strong needs for shorter time-to-market and life products are always problems in a typical System-on-chip design. FPGA prototyping is a quick way to do a real-time simulation of the system and identify the potential problems. It is a promising approach to overcome the traditional trade-off between flexibility and performance in the design of computer architectures [1].The more specialized a computer architecture is in a particular application, the faster it will be in executing this application.

On the other hand, regular architectures based on identical processing elements (PEs) of so-called systolic array are a natural choice for application-specific implementations of such a process; due to the development of VLSI fabrication technologies, systolic algorithms are now being implemented as a practical hardware; so it is becoming more important to know their design methodologies [2, 3, 4]. We have combined these two purposes, and the result is designing a systolic array based on FPGA.

What can a systolic array do? This question is very important, and the answer is that it is every sequential algorithm that can be transformed to a parallel version suitable for running on array processors that execute operations in the so-called systolic manner, and systolic array is one of solutions to the need for a highly parallel computational power. Due to the use of matrix-multiplication algorithm in wide fields such as Digital Signal Processing (DSP), image processing, solutions of differential comparison, non-numeric application, and complex arithmetic operations, we shall design a 2D systolic array that implements this algorithm.

There are several projects in systolic array such as:

1- A comparison of block-matching algorithms mapped to systolic-array implementation [5].

2- Programming Systolic Arrays [6].

3- A Multilevel Systolic Approach for Fuzzy Inference Hardware [7].

4- An efficient Systolic Array for the Discrete Cosine Transform Based on Prime-Factor Decomposition [8].

This research is carried out in Faculty of Mechanical and Electrical Engineering for a whole year.

Our contributions in this research are:

1- Designing a fast-arithmetic Multiply-Accumulate (MAC) unit based on pipeline technique and using Carry Save Adder (CSA) and Carry Lookahead Adder (CLA) units. This design executes the arithmetic operations much faster than the nonpipelined MAC unit.

2- Designing a control unit that verifies the synchronization of all processing elements and considering it a global control.

3- Modeling the systolic Array architecture based on VHDL, following the top-down methodology in designing.

Purpose and Importance of this Research

The purpose of this research is to efficiently resolve a wide spectrum of computational problems by implementing the systolic array architecture on FPGA.

Designers search for high performance and flexible architectures; so this research introduces one of the most important ways to meeting them. Our research combines different properties that are seldom found together. These properties are:

1- flexible hardware and software (i.e. FPGA technology).

2- Spatial and temporal parallelism (i.e. systolic array Architecture).

Research Methodology

A system can be described at different levels, namely system level, RTL level, gate level, and transistor level. System level design methodologies introduce new design flows that are complementary to the ones provided by existing tool-sets based on Hardware Description Languages HDLs**.** The methodology of reconfigurable circuits and systems is evolving from Tinkertoy approach to an innovative parallel computing paradigm which combines computing in time with computing in space. We have used the top-down system level design methodology which means here describing a complete system at an abstract level using HDLs and automated tools. Figure (1) shows the register transfer level (RTL) and top-down methodology for a systolic array design.

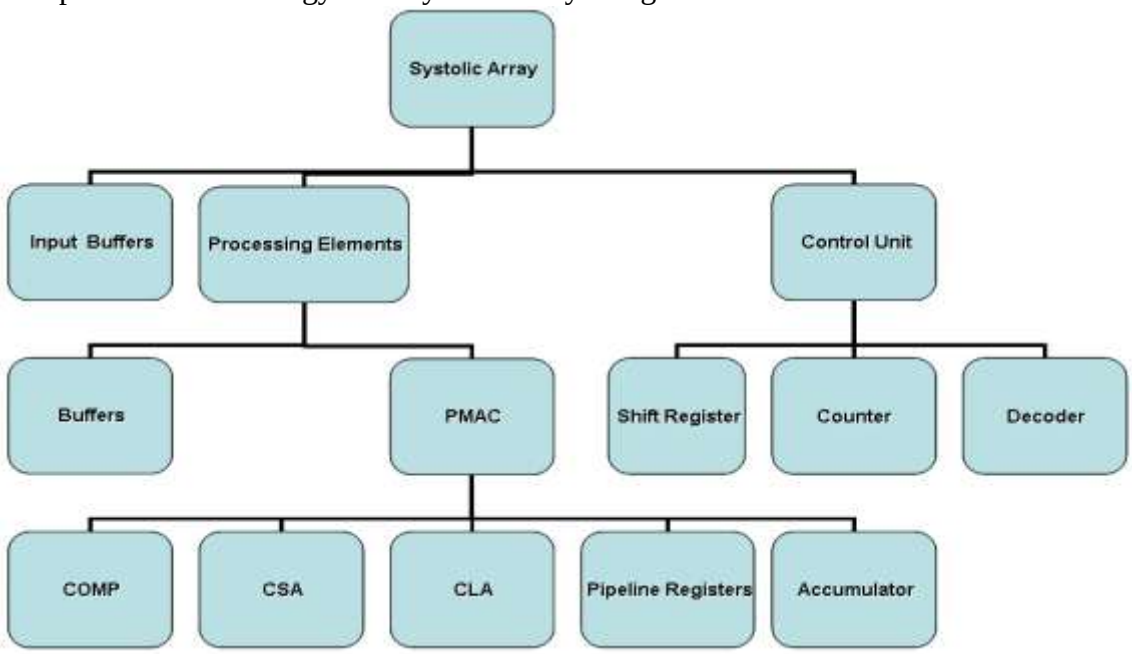


Figure (1): RTL level, top-down methodology for systolic array design.

VHDL Language:

VHDL is a language for describing digital electronic systems. It is developed through the US government's Very High Speed Integrated Circuits (VHSIC) program.

In VHDL, we model Hardware in the form of a programming language that contains:

1- An **ENTITY** being a list with specifications of all input and output pins (PORTS) of the circuit. Its syntax is shown below[9, 10, 11]:

```
ENTITY entity_name IS
    PORT(
        Port_name : signal_mode  signal_type;
        Port_name : signal_mode  signal_type;
        … );
END entity_name;
```

2- The **ARCHITECTURE** being a description of how the circuit should behave (function).

Its syntax is the following:

```
ARCHITECTURE architecture_name OF entity_name IS
    [ declarations]
```

```
BEGIN
        (code)
END architecture_name;
```

The architecture body can take a few forms since we can describe hardware systems in a number of ways. They are [9]:

1. **Dataflow Description:** models the hardware in terms of the movement of data over continuous time between combinational logic components such as adders, decoders and primitive logic gates.
2. **Behavioral Description:** describes how the given entity behaves, given various inputs as a function of control statements.
3. **Structural Description:** describes how the given entity functions, based on physical hardware implementation.

VHDL has various data types as Variables, constants, integers, floating point, time, characters, Booleans, standard logic, composite data types, and user defined data types.

There are two basic kinds of statement in VHDL [11, 12]:

1. **Concurrent statements** are used inside architectures. Concurrent statements are executed in parallel, so they make VHDL fundamentally different from most software languages.

Concurrent statements include the following:

- Signal assignments (selected and conditional).
- Component instantiations.
- Generate statements.
- Procedure and function calls.
- Process statements.

2. **Sequential statements** are used inside processes and functions, and they include:

Signal assignment, variable assignment, constant assignment, Wait, if-then-else, case, loop, while loop, for loop and next statements.

VHDL is designed to allow the description of the structure of a hardware system. This includes a composition of subsystems and how those subsystems interconnect. It also allows the description of hardware systems as a system of programming terms. Finally, it permits the simulation of a defined system based on the Hardware Description given.

Field Programmable Gate Array (FPGA)

Field Programmable Gate Arrays (FPGAs) are becoming a critical part of every system design [1, 13]. FPGA chip generally consists of:

1.Configurable Logic Block (CLB).
2.Configurable I/O blocks.
3.Programmable interconnect.
4.Clock circuitry.
5.Programmable elements: Static RAM, Anti-fuses, and Flash.
6.Additional logic resources: Arithmetic/Logic Units (ALU), Memory, and Decoders.

The FPGA architecture based on Look-Up Table (LUT) dominates the existing programmable chip industry, in which the basic programmable logic element is a *K*-input lookup table. Most FPGAs are hierarchical in nature. For example, Altera Stratix III device families and Xilinx Virtex-5 device families provide logic array blocks (LABs) or CLBs that can accommodate a cluster of basic logic elements (BLEs) with fast local interconnects.The FPGA is the latest in the family of Programmable Logic Devices (PLDs). Figure (2) shows the structure of FPGA.

With the development of FPGAs. there are now more opportunities than ever for implementing quite different systems. It could be used for implementing random logic and glue logic in low volume systems with non-aggressive speed and capacity demands. If the capacity of a single FPGA is not enough to handle the desired computation, multiple FPGAs may be included on the board, distributing the computation among these chips. It can be used in a much more flexible manner than standard gate arrays.

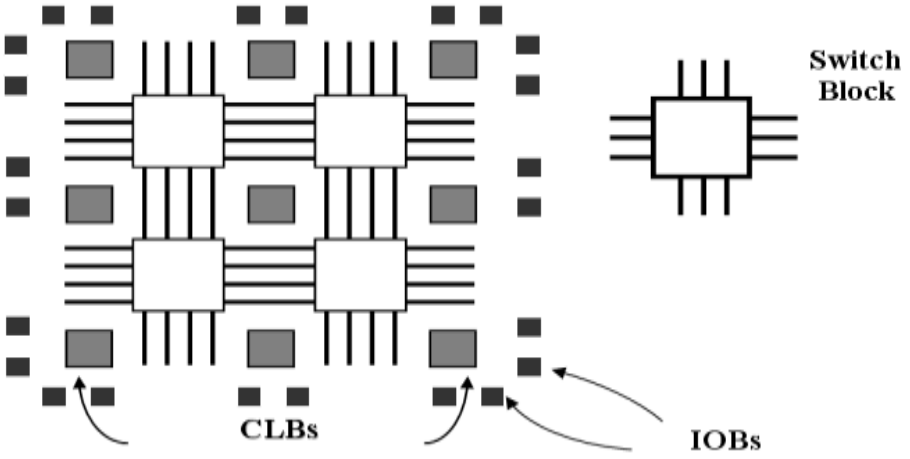


Figure (2): The structure of FPGA.

Systolic Array

A Systolic Array is a type of parallel computer architecture that consists of:

- An array of Processing Elements (PEs).
- Interconnected with localized data links.

The features that make a systolic array a distinct architecture are modularity, rhythm, synchrony, extensibility, and pipelinebility [2, 3]. The general architecture of systolic array is illustrated in figure (3). The term "systolic" is used because of the analogy of these systems with the circulatory system of the human body. In the circulatory system, the heart sends and receives a large amount of blood as a result of the frequent and rhythmic pumping of a small amount of blood though arteries and veins; so if compared we find that the heart in systolic computer systems would correspond to the global memory as the source and destination of data. The arterial-venous network would similarly correspond to processors and communication links.

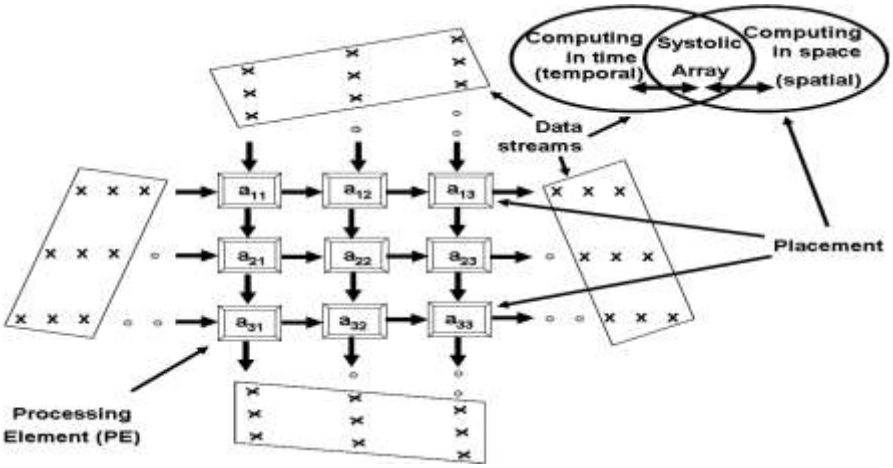


Figure (3): general architecture of systolic array

Systolic Concept

A Systolic system is such a combination of an algorithm and an integrated circuit that implements it [2, 3, 4]. It is generally classified as high-performance, special-purpose VLSI computer system suitable for specific application requirements that must balance intensive computations with demanding I/O bandwidths. It is a massively parallel architecture, organized as a network of identical and relatively simple PEs, which execute operations synchronously. Systolic algorithms address the performance requirements of special-purpose systems by achieving significant speedup, due to parallel processing and prevention of I/O and memory bandwidth bottleneck. Data are pumped in a rhythmic manner from memory through the systolic array before the end result is returned to the memory. The global clock and explicit timing delays synchronize the system.

Two Dimensional Systolic Array (Mesh-connected Array) Design

There are three types of systolic array based on its topology:

1.One dimensional systolic array (Linear Array).

2.Two dimensional systolic array (Mesh-connected Array).

3.Three dimensional systolic array.

We have designed the Two dimensional systolic array. However, the proposed arrays of processors for this type of systolic array are [14, 15]:

1.Hexagonal array.

2.Pipelined array.

3.Semibroadcast array.

4.Broadcast array.

5.Wavefront array.

Our choice for our design is the Wavefront array topology because it represents a well-synchronized structure.

1. Steps of Mapping Procedure

There are five steps of mapping algorithm to systolic architecture:

1. Buffer all the variables.
2. Determine the PEs functions by collecting the assignment statements in the loop bodies into m input and n output functions {Determine dependence matrix}.
3. Find transformation (T).
4. Apply a linear reindexing transformation T.
5. Find connections between processors and the direction of data flow.

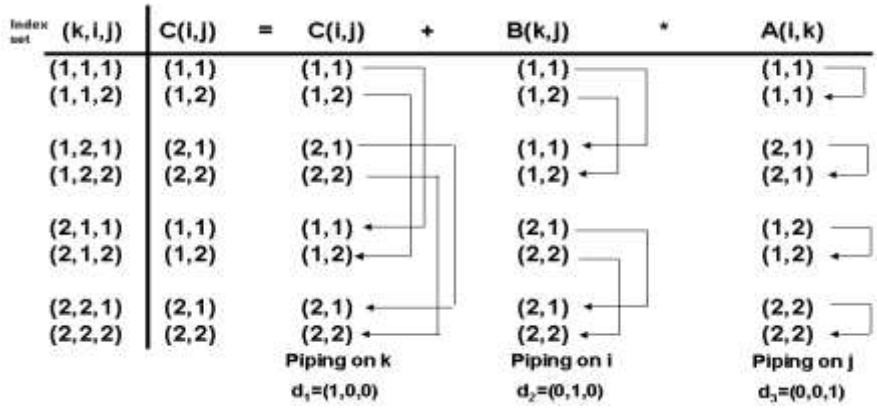
We'll apply these steps to the algorithm of multiplication of 2-by-2 matrices.

First, consider the following algorithm which represents multiplication of two 2-by-2 matrices A and B [3, 14, 15, 16]:

```
for (k =1;k<=2;k++)
   for (i=1;i<=2;i++)
      for (j=1;j<=2;j++)
          C(i,j) = C(i,j) + B(k,j) * A(i,k);
```

Second, represent the index set for this algorithm as in table (1):

Table (1): index set for matrix multiplication algorithm.



By applying the above mentioned steps to this algorithm:

Step one: buffering all variables:

Each index element is shown as a three-tuple (k,i,j). Note that for both index elements (k,i,1) and (k,i,2), the same value of A(i,k) is used; that is, the value A(i,k) can be piped on the j direction. Similarly, values B(k,j) and C(i,j) can be piped on i and k directions, respectively. Based on these facts, the algorithm can be rewritten by introducing buffering variables A^{j+1}, B^{i+1}, and C^{k+1}, as follows:

```
for (k =1;k<=2;k++)
  for (i=1;i<=2;i++)
    for (j=1;j<=2;j++)
      {          A^{j+1}(i,k) = A^j(i,k);
            B^{i+1}(k,j) =B^i(k,j);
            C^{k+1}(i,j) = C^k(i,j) + B^i(k,j) * A^j(i,k); }.
```

Step two: Determine dependce matrix:

The set of data dependence vectors can be found by equating indices of all possible pairs of generated and used variables. So the dependence matrix D= [d_1| d_2| d_3] can be expressed as:

$$\mathbf{D} = \begin{bmatrix} \mathbf{d_1} & \mathbf{d_2} & \mathbf{d_3} \\ \mathbf{1} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{1} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{1} \end{bmatrix}$$

Step three: Find transformation (T):

we are looking for a transformation T that is of the form:

$$\mathbf{T} = \begin{bmatrix} \mathbf{\Pi} \\ \mathbf{S} \end{bmatrix} \text{ Where } \quad \mathbf{\Pi d_i > 0} \qquad \text{and Let: } \mathbf{T} = \begin{bmatrix} \mathbf{t_{11}} & \mathbf{t_{12}} & \mathbf{t_{13}} \\ \mathbf{t_{21}} & \mathbf{t_{22}} & \mathbf{t_{23}} \\ \mathbf{t_{31}} & \mathbf{t_{32}} & \mathbf{t_{33}} \end{bmatrix}$$

The condition $\mathbf{\Pi d_i > 0}$ (for i=1, 2, 3) implies, and to reduce the turnaround time, we try to choose the smallest values for t_{11}, t_{12}, and t_{13} such as:

$t_{11} = t_{12} = t_{13} = 1$; that is $\mathbf{\Pi = (1,1,1)}$

Our choice of S will determine the interconnection of the processors.

A large number of possibilities exist, each leading to different network geometries.

our option is:

$$\mathbf{S} = \begin{bmatrix} \mathbf{0} & \mathbf{1} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{1} \end{bmatrix} \qquad \text{Thus:} \quad \mathbf{T} = \begin{bmatrix} \mathbf{1} & \mathbf{1} & \mathbf{1} \\ \mathbf{0} & \mathbf{1} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{1} \end{bmatrix}$$

In general, for the multiplication of two n-by-n matrices, 2^n PEs are needed thus for our design, four PEs are needed. The interconnection between these processors is defined by:

$$\mathbf{S_1 d_i} = \begin{bmatrix} \mathbf{x} \\ \mathbf{y} \end{bmatrix} \begin{matrix} \longrightarrow \mathbf{i} \\ \longrightarrow \mathbf{j} \end{matrix}$$

Where x and y refer to the movement of the variable along the direction i and j, respectively. Thus

$$\mathbf{S_1 d_1} = \begin{bmatrix} \mathbf{0} \\ \mathbf{0} \end{bmatrix} \quad \mathbf{S_1 d_2} = \begin{bmatrix} \mathbf{1} \\ \mathbf{0} \end{bmatrix} \quad \mathbf{S_1 d_3} = \begin{bmatrix} \mathbf{0} \\ \mathbf{1} \end{bmatrix}$$

Step four: Apply a linear reindexing transformation (T):
When we apply a linear reindexing transformation, table (2) will result:

Table (2): reindexing transformation.

Transfer matrix		Original index		Transform index	C 's	A 's	B 's	Time	PE element
T	*	$[1,1,1]^t$	=	$[3,1,1]^t$	c_{11}	a_{11}	b_{11}	3	(1,1)
T	*	$[1,1,2]^t$	=	$[4,1,2]^t$	c_{12}	a_{11}	b_{12}	4	(1,2)
T	*	$[1,2,1]^t$	=	$[4,2,1]^t$	c_{21}	a_{21}	b_{11}	4	(2,1)
T	*	$[1,2,2]^t$	=	$[5,2,2]^t$	c_{21}	a_{21}	b_{12}	5	(2,2)
T	*	$[2,1,1]^t$	=	$[4,1,1]^t$	c_{12}	a_{12}	b_{21}	4	(1,1)
T	*	$[2,1,2]^t$	=	$[5,1,2]^t$	c_{12}	a_{12}	b_{22}	5	(1,2)
T	*	$[2,2,1]^t$	=	$[5,2,1]^t$	c_{22}	a_{22}	b_{21}	5	(2,1)
T	*	$[2,2,2]^t$	=	$[6,2,2]^t$	c_{22}	a_{22}	b_{22}	6	(2,2)

Step five: Find connections between processors and the direction of data flow:
At first unit of time, b_{11} and a_{11} enter $PE_{1,1}$ which contains variable c_{11}. Then each PE performs a multiply and an add operation. Figure (4) shows the required interconnections between the PEs.

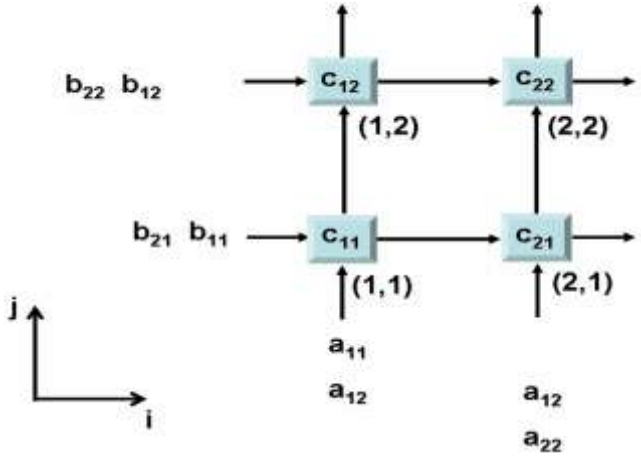


Figure (4): required interconnections between the PEs.

HDL design for systolic array

We'll now create an HDL design for systolic array which performs multiplication of two 2-by-2 matrices A and B. The hierarchical organization of a VHDL design for systolic array based on top-down methodology which explained earlier is illustrated in Figure (5).

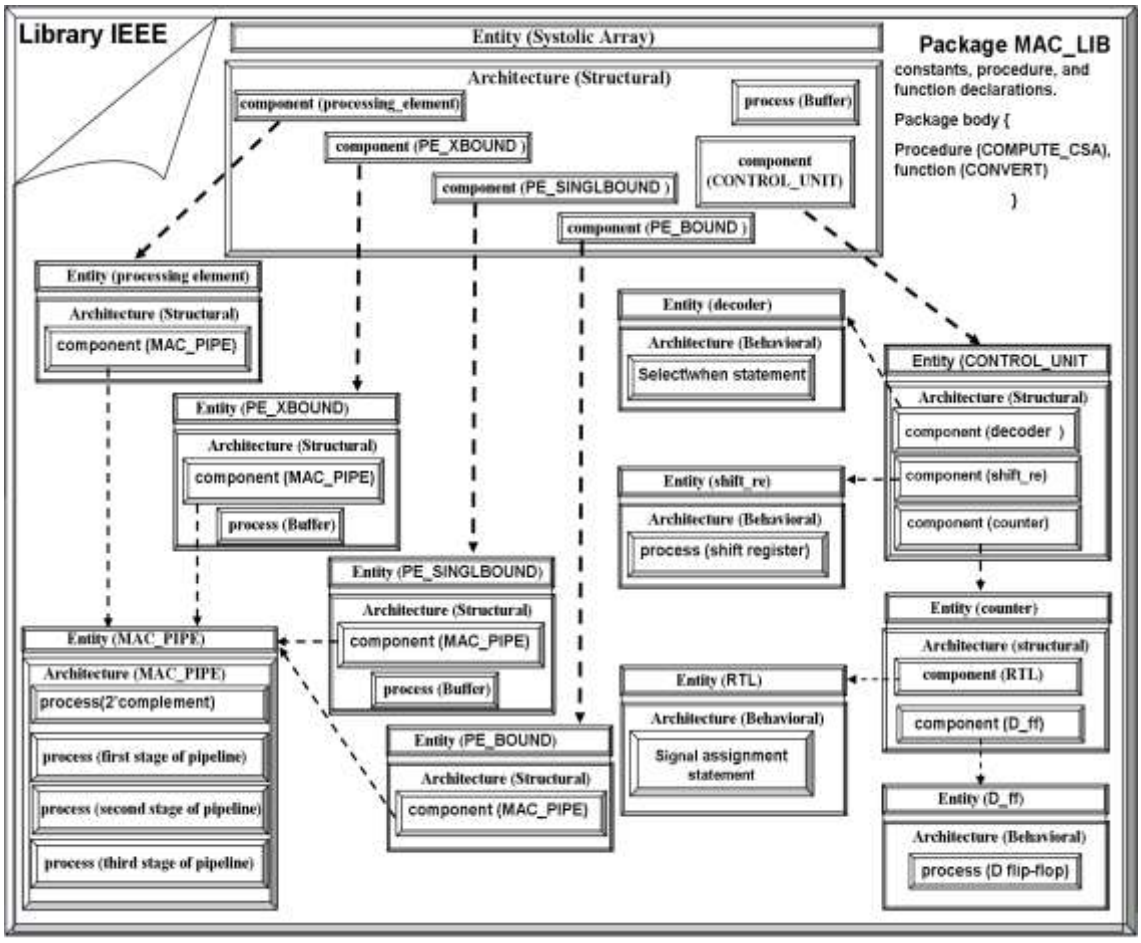


Figure (5): Hierarchical organization of a VHDL design for systolic array based on top-down methodology.

The names of units in figure (5) are the same of that we have used in VHDL code.The VHDL code for our design is about \800\ programming lines; so we use it as an appendix of this research, and we have mentioned parts of the code when necessary. The RTL level schematic for 2D systolic array that we have designed by Xilinx ISE V9.1 program is depicted in Figure (6). From figure (6), we can ratiocinate that the VHDL code for systolic Array must contain two input buffers and five components. One component for control unit and the other for four PEs that perform the same algorithm but is different in inputs and outputs. The entity of systolic array is described as:

```
-------------------------------------------------------------
-- The bold font is used to describe VHDL keywords.
entity systolic is
   port (CLK, reset: in  std_logic;
 A1, A2: in  std_logic_vector (DATA_BITS-1 downto 0);
 B1,B2 : in  std_logic_vector (DATA_BITS-1  downto 0);
CPE1, CPE2  : inout std_logic_vector (RESULT-BITS-1 downto 0) :=(others=>'0');
CPE3, CPE4  : inout std_logic_vector (RESULT-BITS-1 downto 0) :=(others=>'0');
 end systolic;
-------------------------------------------------------------
```

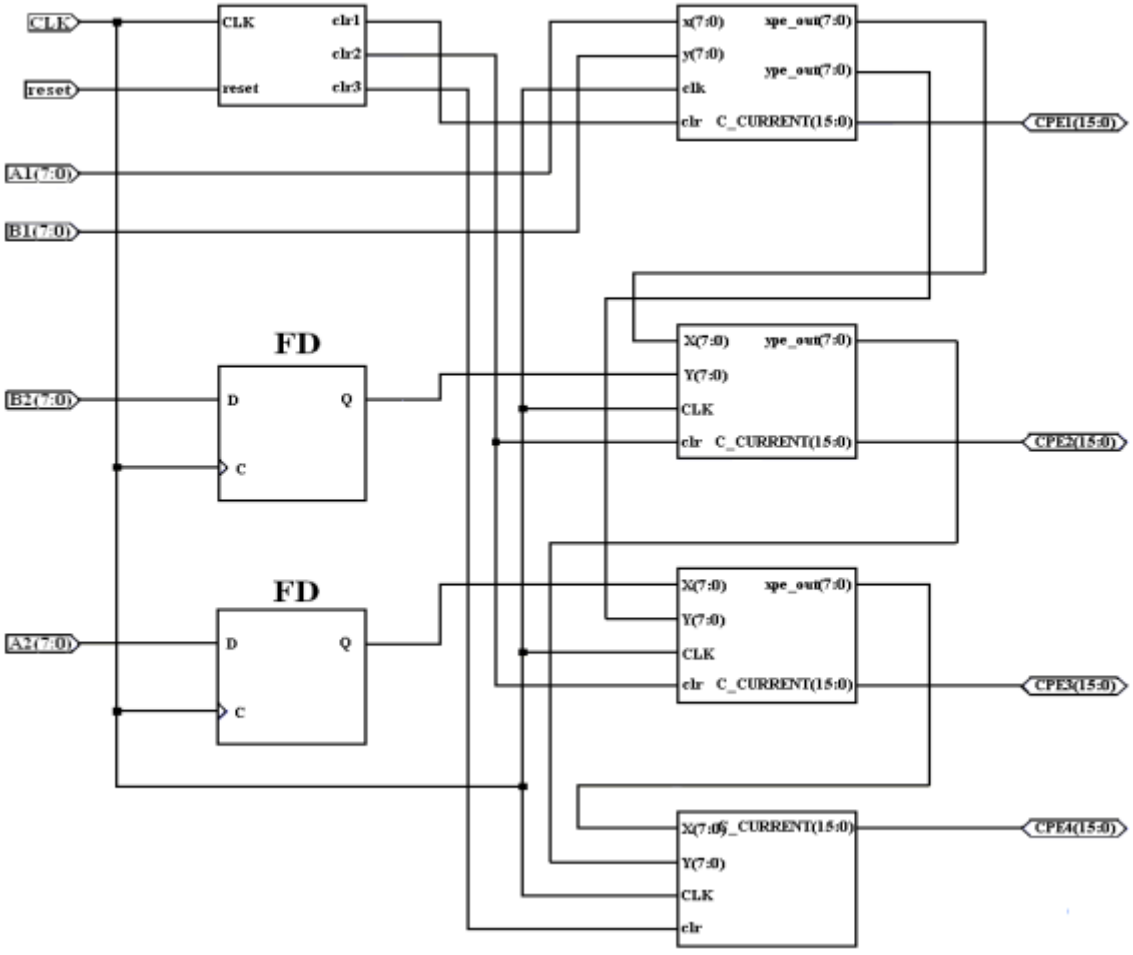


Figure (6): RTL level for 2D systolic array.

1. Control Unit

The control unit creates the required signals. The synthesis of such control units is described for designing sequential circuits. Figure (7) shows the structure of a control unit. It consists of shift register, counter and decoder. The control signals are periodic which Triggered by the maximal clock cycle. There are three control signals (clr1, clr2, clr3) synchronize the MAC units in the PEs.

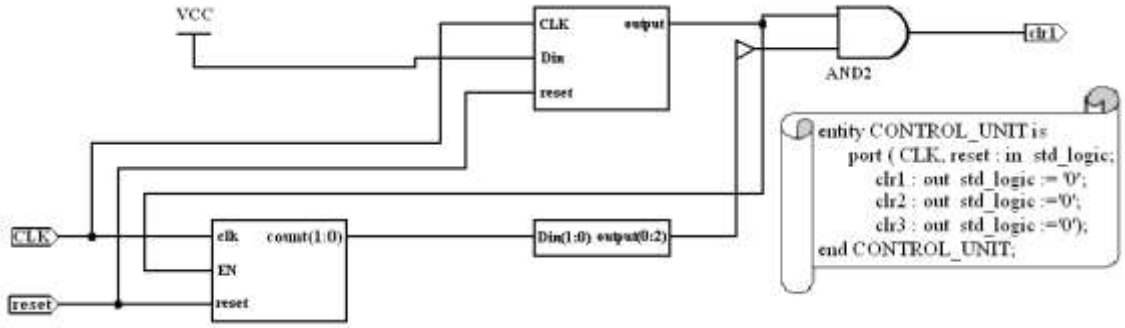


Figure (7): control unit architecture.

2. Processing Element:

There are four PEs. They do the same function, but are just different in the output, so we'll define I/O for each PE alone:

PE1: has two 8-bit inputs, two 8-bit outputs, and one 16-bit output (register mode).

PE2: has two 8-bit inputs, one 8-bit output, and one 16-bit output (register mode).

PE3: has two 8-bit inputs, one 8-bit output, and one 16-bit output (register mode).

PE4: has two 8-bit inputs and one 16-bit output.

The RTL level of the PE shown in figure (8):

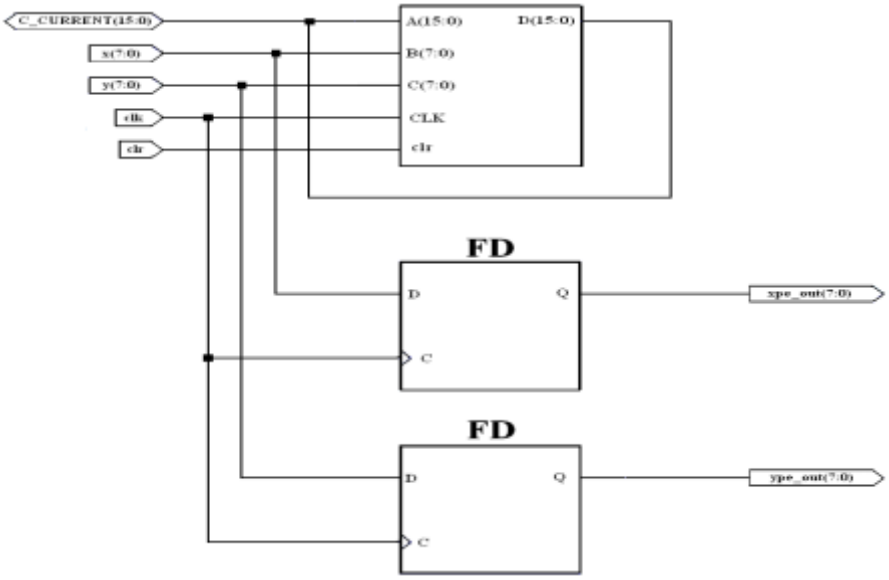


Figure (8): RTL level of PE Architecture.

The VHDL code for the PE consists of one process that buffersthe input data and one component (MAC unit) that will be designed later.

The entity of PE is described as:

```
-------------------------------------------------------------
entity processing_element is
   port ( x : in  std_logic_vector (DATA_BITS-1 downto 0);
        y: in  std_logic_vector (DATA_BITS-1 downto 0);
        c_current : inout std_logic_vector (RESULT-BITS-1 downto
0):=(others=>'0');
   xpe_out, ype_out: out  std_logic_vector (DATA_BITS-1 downto 0);
            CLK, clr: in  std_logic);
   end processing_element;
   -------------------------------------------------------------
```

3. Pipelined Multiply-Accumulate (PMAC) Unit

The PMAC unit that we'll design is an 8-bit PMAC such that it accepts data (and outputs results) at a rate of 1/ (6 logic gate delays), in another words we'll insert the pipeline technique to achieve the desired data processing rate. The structure of this PMAC is illustrated in figure (9). It contains a two-complement unit, two kinds of adder units, and registers. In order to pipeline this type of MAC unit, we must place pipeline registers at appropriate points within the CSA array structure. In deciding where to place theses pipeline registers, we should follow the guidelines [17]:

1- The resulting pipeline segments should have approximately equal delays.

2- Pipeline registers should be placed such that pipeline segments are created using the smallest possible pipeline registers.

3- The delay through a CSA is the same, regardless of the data word size.

4- The delay through an inverter can be ignored, since an inverter can be combined with other basic logic gates (e.g., $a'b$ can be implemented as one "special" gate instead of an inverter followed by an AND gate).

Since the multiplication of two 8-bit numbers results in a 16-bit number, the final adder needs to be a 16-bit adder. The final adder can be implemented as a carry-lookahead adder (CLA).

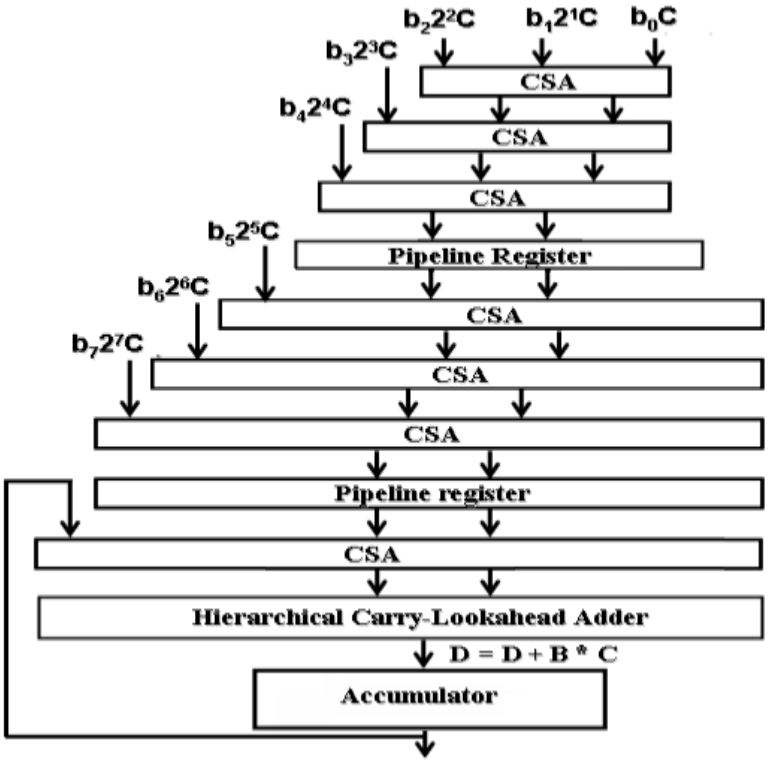


Figure (9): A multiply-accumulate (MAC) unit with pipeline registers inserted at approximately equal delay points.

The depth of pipeline is three stages which mean:

- We'll use three registers (register for each stage).
- First two stages contain three CSA units.
- Third stage contains one CSA unit and one CLA unit.

Dealing with sign numbers, we convert data into a two-complement form by COMP unit before it inputs into the MAC unit. Figure (10) shows the COMP block and the gate-level for a two-complement form of 1-bit of input data.

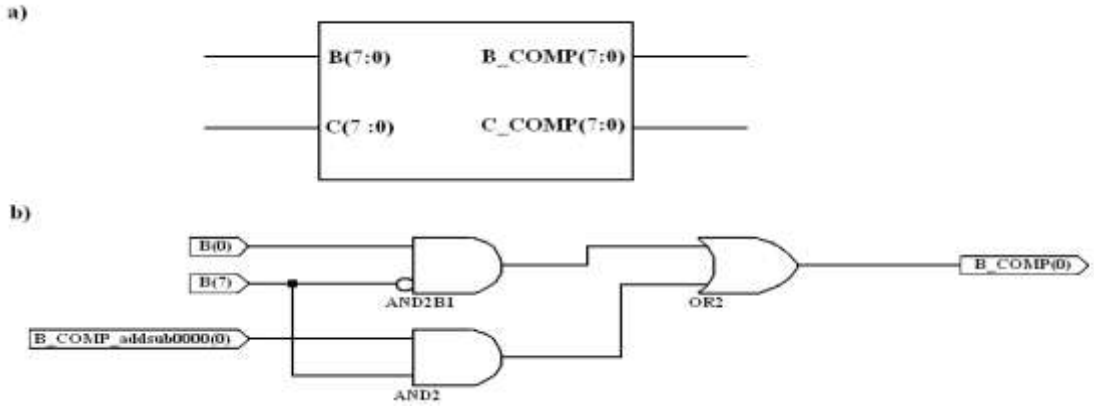


Figure (10): COMP unit a) block diagram .b) gate-level for two-complement form of 1-bit.

3.1. Carry Save Adder

This type of adder is useful when more than two numbers are added [13, 14]. For example, when there are numbers (X, Y, Z, and W) to be added, the CSA first produces the

sum and the saved carry for the first 3 numbers. Assuming that X=0101, y=0111, and Z=0110, the produced sum and saved carry are:

	0101	X
	0111	Y
+	0110	Z
	0100	sum
	1110	saved carry

In the next step, the sum, the saved carry, and the fourth number (W) are Added in order to produce a new sum and a new saved carry, Assuming that W=0010,

	0100	sum
	1110	saved carry
+	0010	W
	1000	new sum
	1100	new saved carry

In the last step, a carry lookahead adder is used to add the new sum and the new saved carry.

The structure of the Carry Saved Adder illustrated in figure (11) [14].

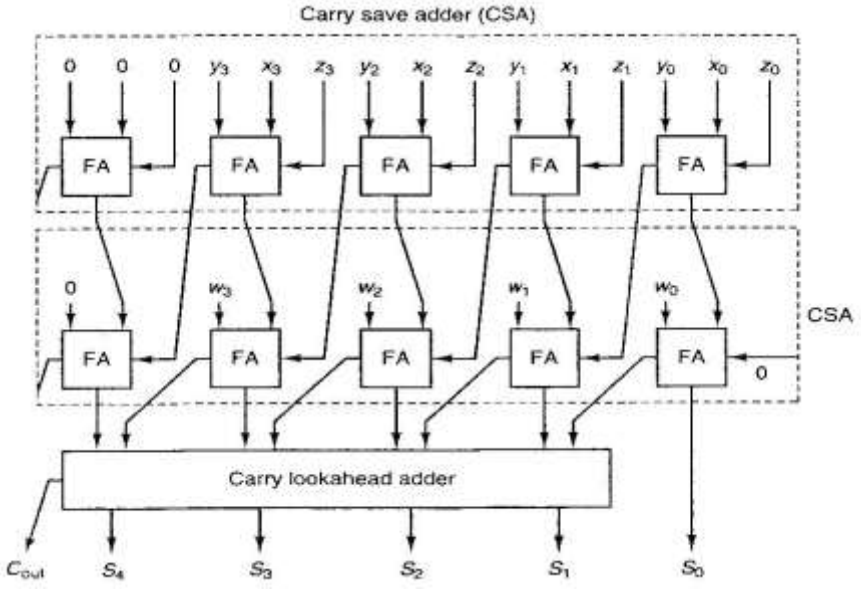


Figure (11): Block diagram of an adder for adding four 4-bit numbers.

3.2. Carry Lookahead Adder

Several compromises are employed between the pseudo parallel and strictly parallel alternative, so this increases the speed of the carry propagation in a ripple carry adder [13, 14]. It produces the input carry bit directly, rather than allowing the carries to ripple from full adder (FA) to full adder. Figure (12) shows the block diagram of the carry lookahead

adder that adds two 4-bit integers [14]. We note that the inputs to each carry block are only the input numbers and the initial carry input (C_0). The Boolean expression for each carry block can be defined by using the carry-out expression of a FA:

$$C_{i+1}= x_i\, y_i + C_i\, (x_i + y_i)$$

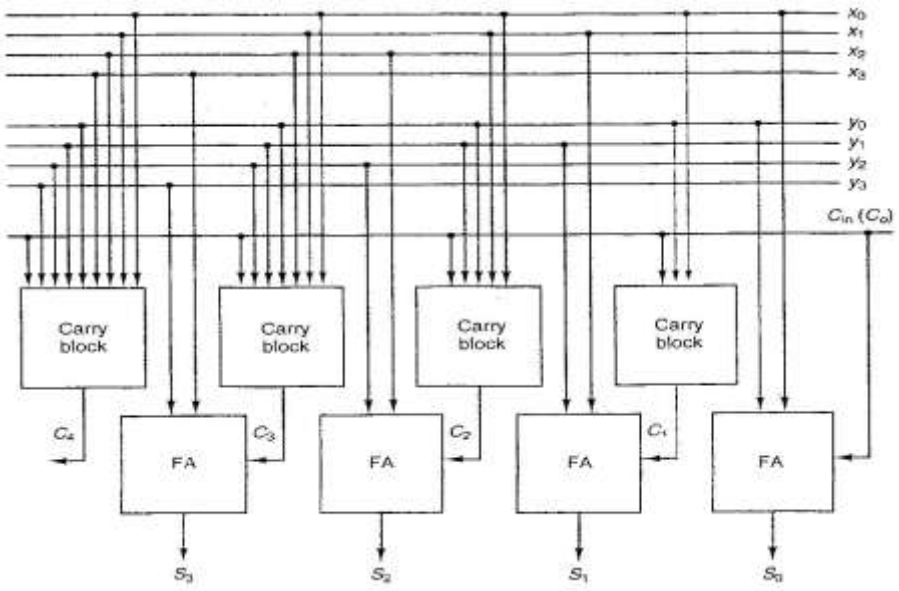


Figure (12): Block diagram of a 4-bit CLA

3.3. VHDL code for PMAC

The VHDL implementation of the pipelined MAC unit is written based on the pipeline design method. In VHDL, variables should be used liberally to aid in behavioral description. While signals should be used only for physical signals in the circuit. The VHDL code for this design will start off with a definition of global constants. These global constants will be defined in a separate (MAC_LIB) package within the default (work) library. The constant DATA_BITS, defined as the number of bits in the input data words, will be set to (8). Since the multiplication of two k-bit numbers will result in a 2k-bit number, RESULT_BITS is defined as twice DATA_BITS. Finally PIPE_DEPTH (the number of pipeline segments) is set to three.

```
------------------------------------------------------------
package MAC_LIB is
-- Declare constants
constant DATA_BITS : integer :=8;
constant RESULT_BITS : integer := DATA_BITS *2;
constant PIPE_DEPTH : integer:=3;
-- Declare functions and procedure
function CONVERT (b_bit: std_logic; c: std_logic_vector (DATA_BITS-1 downto
0); start :INTEGER) return std_logic_vector ;
procedure COMPUT_CSA (pp_carry, pp_sum: out std_logic_vector
(RESULT_BITS-1 downto 0); data1, data2, data3: std_logic_vector ( RESULT_BITS-1
downto 0):= (others => '0'));
end package;
------------------------------------------------------------
```

One VHDL function will be used for simple sign-extension task. This function (CONVERT) will take as inputs a data of size DATA_BITS and an offset value (named start). Then, it will create an output of size RESULT_BITS, in which the data input has been shifted left by the desired offset, with (0)values stored into the rightmost (start) bit positions, and the sign bit (the MSB of the original data input) copied into all bit positions to the left of the data input after it has been shifted. The copying of the MSB value serves to preserve the sign of the original data.

```
-------------------------------------------------------------
-- function definition
-- function to shift and sign-extend (B(i) and C) data.
function CONVERT (b_bit:std_logic; c: std_logic_vector(DATA_BITS-1 downto  0);
start: integer ) return std_logic_vector is
   variable temp     : std_logic_vector (RESULT_BITS-1 downto 0);
 begin
   for i in 0 to start-1 loop
   temp (i) := '0';
   end loop;
   for i in 0 to DATA_BITS-1 loop
   temp (start+i) := b_bit and c(i);
   end loop;
   for i in DATA_BITS to RESULT_BITS-start-1 loop
   temp (start+i):=b_bit and c(DATA_BITS-1);
   end loop;
   return temp;
   end function CONVERT;
-------------------------------------------------------------
```

Next we must define the CSA blocks. So we'll write one procedure to do this purpose. This procedure (COMPUT_CSA) describes the CSA blocks in this CSA array. The VHDL code for a CSA block corresponds to the design shown in figure (11).

```
-------------------------------------------------------------
procedure COMPUT_CSA (pp_carry, pp_sum : out std_logic_vector
(RESULT_BITS-1 downto 0); data1, data2, data3: std_logic_vector (RESULT_BITS-1
downto 0) := others=>'0') is
 variable add_result : unsigned (1 downto 0);
 begin
   pp_carry(0) := '0';
   for i in 0 to RESULT_BITS -2 loop
   add_result :=
   unsigned (std_logic_vector('0' & data1(i)))+
   unsigned (std_logic_vector('0' & data2(i)))+
   unsigned (std_logic_vector('0' & data3(i)));
   pp_carry (i+1):=add_result(1);
   pp_sum(i) := add_result(0);
  end loop;
 pp_sum(RESULT_BITS -1):= add_result(0);
 end procedure COMPUT_ CSA;
-------------------------------------------------------------
```

The VHDL code for pipelined MAC circuit has six process blocks: two for complementing the input data, and one for buffering them, the others for the three pipeline segments used. Each process block executes concurrently with every other process block.

The entity of PMAC unit is described as:

```
-------------------------------------------------------------
entity mac_pipe is
 port(D : out std_logic_vector(RESULT_BITS-1 downto 0):= (others=>'0');
  A       : in std_logic_vector(RESULT_BITS-1 downto 0):= (others=>'0');
  B,C     : in std_logic_vector(DATA_BITS-1 downto 0);
 CLK,clr  : in std_logic);
end mac_pipe;
-------------------------------------------------------------
```

Simulation Results

The result that we have obtained is 2D systolic array design based on FPGA. This systolic array architecture executes the computation of matrix multiplication in time that can't be performed in the recently used software, and the data stored in the accumulator in each PE can return to global memory; so this architecture can deal with a large amount of data and overcome the drawbacks that result from I/O and memory bandwidth bottleneck. Because we have designed the PMAC based on pipeline technique, the speed-up has increased and the computation has become faster than nonpipelined MAC. Figure (13) represents the waveform that simulates our design using Xilinx-ISE V9.1 program. By multiplication of two matrices:

$$\mathbf{A}=\begin{bmatrix}\mathbf{23} & \mathbf{26}\\ \mathbf{34} & \mathbf{-4}\end{bmatrix},\ \mathbf{B}=\begin{bmatrix}\mathbf{20} & \mathbf{16}\\ \mathbf{-12} & \mathbf{9}\end{bmatrix}$$

The result is matrix C whose elements appear in the registers of processing elements (CPE1, CPE2, CPE3, CPE4).

$$\mathbf{A * B = C} \Rightarrow$$

$$\begin{bmatrix}\mathbf{23} & \mathbf{26}\\ \mathbf{34} & \mathbf{-4}\end{bmatrix}*\begin{bmatrix}\mathbf{20} & \mathbf{16}\\ \mathbf{-12} & \mathbf{9}\end{bmatrix}=\begin{bmatrix}\mathbf{148} & \mathbf{602}\\ \mathbf{728} & \mathbf{508}\end{bmatrix}=\begin{bmatrix}\mathbf{CPE1} & \mathbf{CPE2}\\ \mathbf{CPE3} & \mathbf{CPE4}\end{bmatrix}$$

In this simulation, we verify that this design is correct, and we can note that it needs four clock cycles to do the computation (in our case, data stay in PEs).

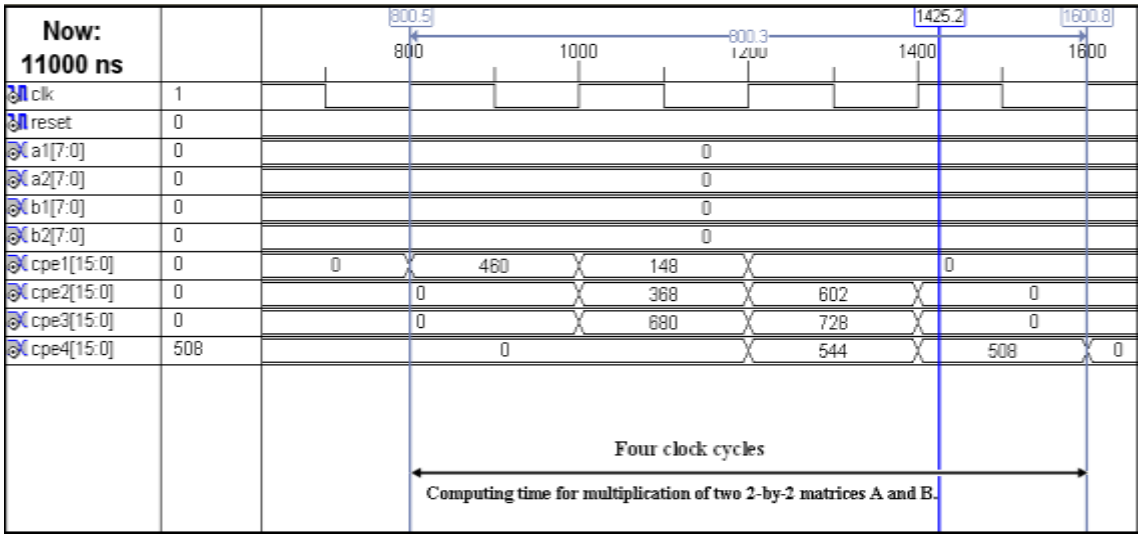


Figure (13): waveform diagram for simulation 2D systolic array design.

Conclusion and Recommendations

Systolic array architectures have provided significant performance improvements for many algorithms especially when we combine this architecture with FPGA technology. We have introduced a new comprehension in using top-down system level methodology for dealing with both temporal and spatial computing which characterize the systolic array. There are some interesting ideas that require future work:

1. There are no standardized methods for evaluating performance parameters of a systolic array.
2. Users need tools that support systolic array design.
3. Debugging of FPGA-based systems has not been addressed yet. FPGA debugging in its current state is hardware debugging i.e., single stepping through synchronous circuits, tracing signals, and analyzing.
4. Support for Partitioning of systolic array to enable very large sized problems to be mapped to FPGA.

Acronyms and Terminology:

HDL	Hardware Description Language	CSA	Carry Save Adder
CLB	Configurable Logic Block	CLA	Carry Lookahead Adder
FPGA	Field Programmable Gate Array	PE	Processing Element
VHSIC	Very High Speed Integrated Circuit	I/O	Input/Output.
MAC	Multiply and Accumulator	LUT	Look-Up Table
PMAC	Pipelined Multiply and Accumulator	LAB	Logic Array Block
PLD	Programmable Logic Device	BLE	Basic Logic Element
VLSI	Very Large Scale Integration	FA	Full Adder
COMP	2'complement Transformation Unit	RTL	Register Transfer Level
DSP	Digital Signal Processing	MSB	Most Significant Bit
VHDL	VHSIC Hardware Description Language		
ASIC	Application Specific Integrated Circuit		

References:

[1]. SMITH, D. J. HDL *Chip Design "A practical guide for designing, synthesizing and simulating ASICs and FPGAs using VHDL or Verilog"*. Doone publication, Madison, AL, USA, 1996, 555.

[2]. ZOMAYA, A. Y. *Parallel and Distributed Computing Handbook.* McGraw-Hill, New York, USA, 1996, 500-536.

[3]. PIRSH, P. *Architectures for Digital Signal Processing*. John Wiley & Sons Ltd, Baffin's Lane, Chichester, West Sussex PO19 1UD, England, 1998, 419.

[4]. KUMAR, V.; GRAMA, A; GUPTA, A; KARYPIS, G. *Introduction to Parallel Computing "Design and Analysis of Algorithms"*. 2nd.ed, Addison Wesley, 2003, 491-523.

[5]. CHENG, S.; HANG H. *A Comparison of Block-Matching Algorithms Mapped to Systolic-Array Implementation.* IEEE, Transactions on Circuits and Systems for Video Technology, VOL. 7, N°. 5, OCTOBER 1997, 741-757.

[6]. HUGHEY, R. *Programming Systolic Arrays*. IEEE Computer Society, VOL. 41, N°. 8, Aug, 1992, 604-618.

[7]. DESALVADOR, L.; GUTIERREZ, J. *A Multilevel Systolic Approach for Fuzzy Inference Hardware*. IEEE Micro, Vol. 15, N°. 5, 1995, 61-71.

[8]. LIM, H.; SWARTZLANDER, E. E. *An efficient Systolic Array for the Discrete Cosine Transform Based on Prime-Factor Decomposition*. IEEE VLSI in computers & Processor, VOL. 39, N°. 11, 1995, 644- 649.

[9]. NAYLOR, D.; JONES, S. *VHDL: A Logic Synthesis Approach.* Chapman & Hall, London, UK, 1997, 339.

[10]. COHEN, B. *VHDL "Coding Styles and Methodologies"*. Kluwer Academic Publishers, USA, 1995, 365.

[11]. PERRY, D. L. *VHDL: Programming by Example.*4th.ed, McGraw-Hill, USA, 2002, 476.

[12]. ASHENDEN, P. J. *The Designer's Guide to VHDL.* 2nd.ed, Morgan Kaufmann Publishers, San Francisco, USA, 2002, 739.

[13]. NELSON, V. P.; NAGLE, H. T.; CARROLL, B.D.; IRWIN, J. D. *Digital Logic Circuit Analysis and Design.* Prentice-Hall, Inc, New Jersey, USA, 1995, 842.

[14]. ZARGHAM, M. R. *computer Architecture "single and parallel systems"*. Prentice–Hall International, Inc, 472.

[15]. ZHANG, D. *Parallel VLSI Neural system Design*. Springer, Singapore, 1999, 257.

[16]. MOLDOVAN, F. I. *Parallel Processing "from Applications to Systems"*. Morgan Kaufmann Publishers, San Mateo, California, USA, 1993,567.

[17]. LEE, S. *Advanced Digital Logic Design "Using VHDL, State Machines, and Synthesis for FPGAs"*. THOMSON, Canada, 2006, 488.