Design and Verification of an Application-Specific PLD Using VHDL and SystemVerilog

Gi-Yong Song

Chungbuk National University, Korea

Regularity

Recursiveness

Locality

Brief introduction to a systolic array

Systolic array formed by interconnecting a set of identical data-processing cells in a uniform manner is a combination of an algorithm and a circuit that implements it, and is closely related conceptually to arithmetic pipeline.

Systolic array for convolution

The problem of convolution is defined as follows: Given two sequences $u(i)$ and $w(i)$, $i = 0, 1, \cdots, N-1$, the convolution of two sequences is

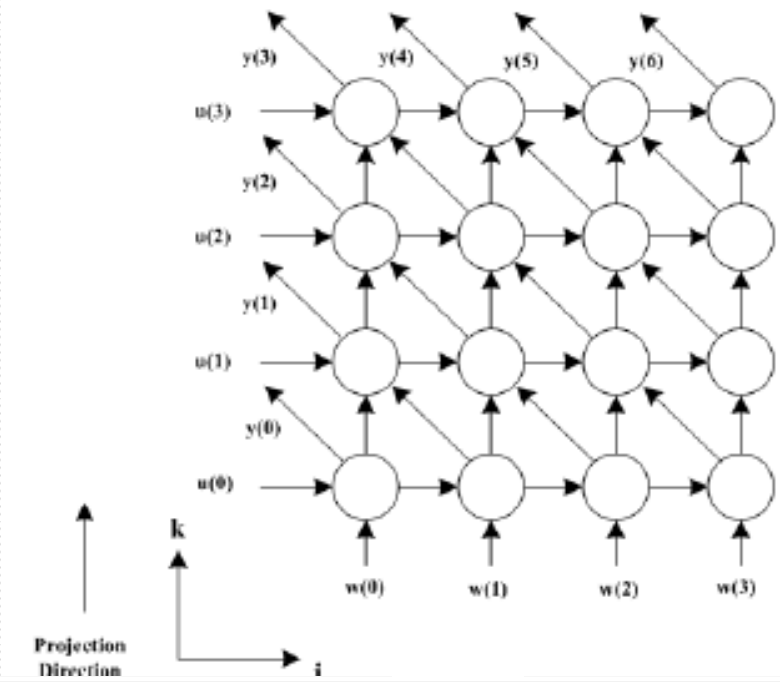
$$y(i) = \sum_{k=0}^{N-1} u(k) * w(i-k)$$

where $i = 0, 1, \cdots, 2N-2$

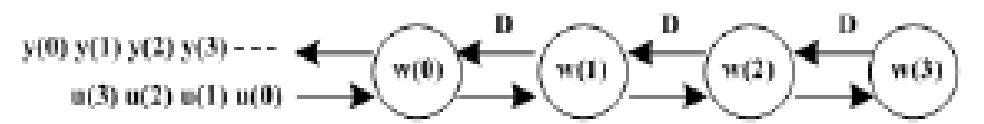
Dependence Graph

Dependence graph is a graph that shows the dependence of the computions that occur in an algorithm. A DG can be considered as the graphical representation of a single assignment code.

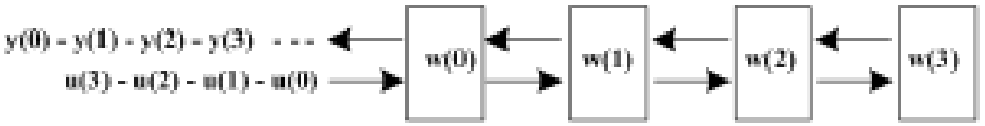
Single assignment code is a form where every variable is assigned one value only during the execution of the algorithm.



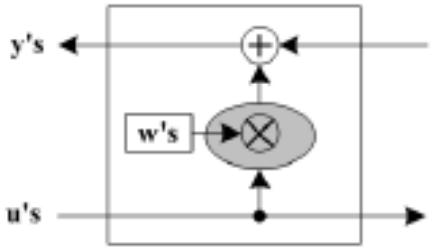
DG for convolution



SFG



Systolic array



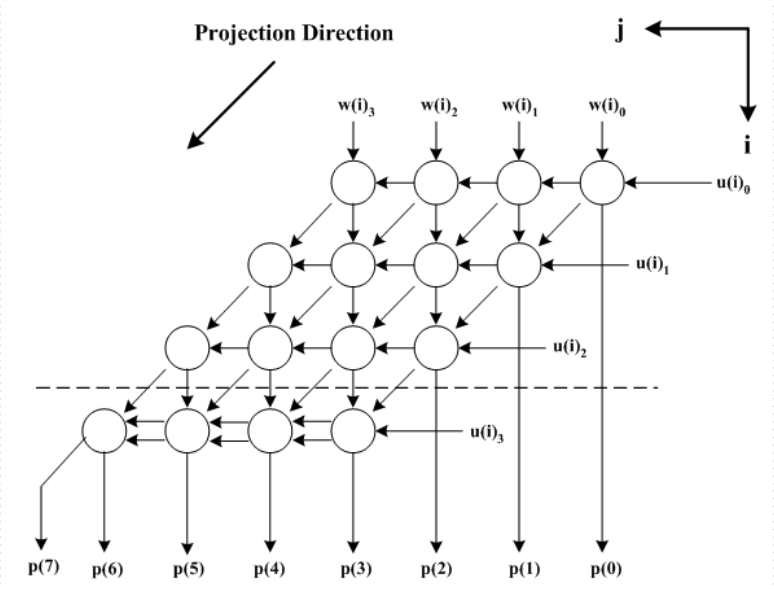
Systolic array cell

Motivation for super-systolic array

Multiplier in each cell of the systolic array for convolution is a natural candidate for systolization and can be implemented using systolic array.

Systolic Array Multiplier

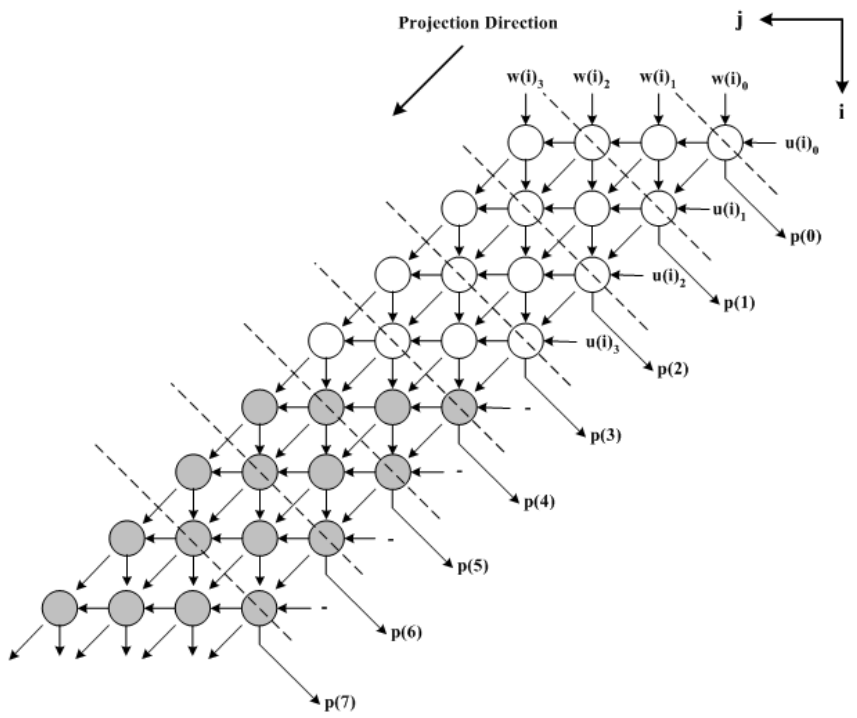
DG for *M*-bit multiplier performing $p(i)=u(i)*w(i)$ is shown for the case of $M=4$.

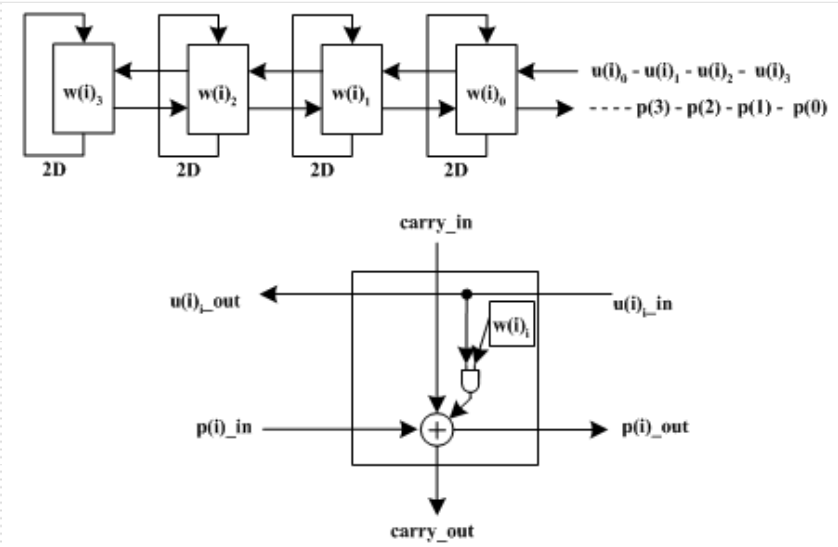


DG for 4-bit multiplier

Data flow pattern divides the DG into upper and lower part as shown with dash line, each part resulting in systolic array with different interconnection.

At the same time, output data are produced from every node, resulting in a large number of output port in the SFG generated by the *ij*–projection.





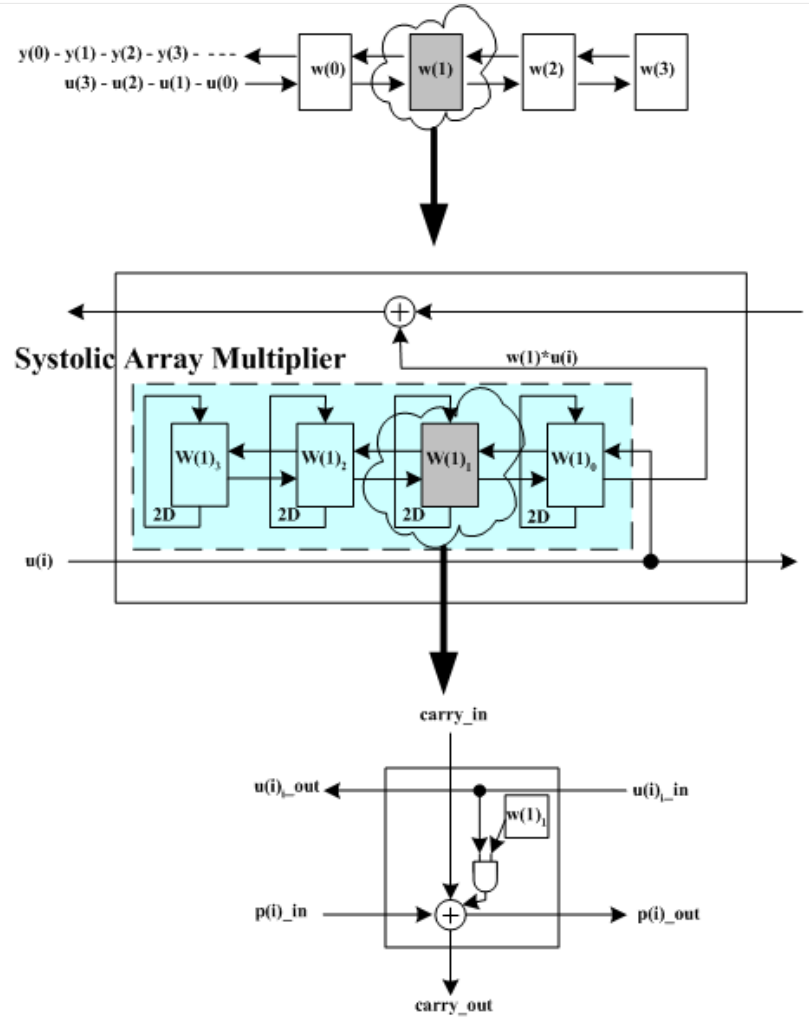
Systolic array for 4-bit multiplier with two ports and internal structure of the cell

Modified DG using index space extension

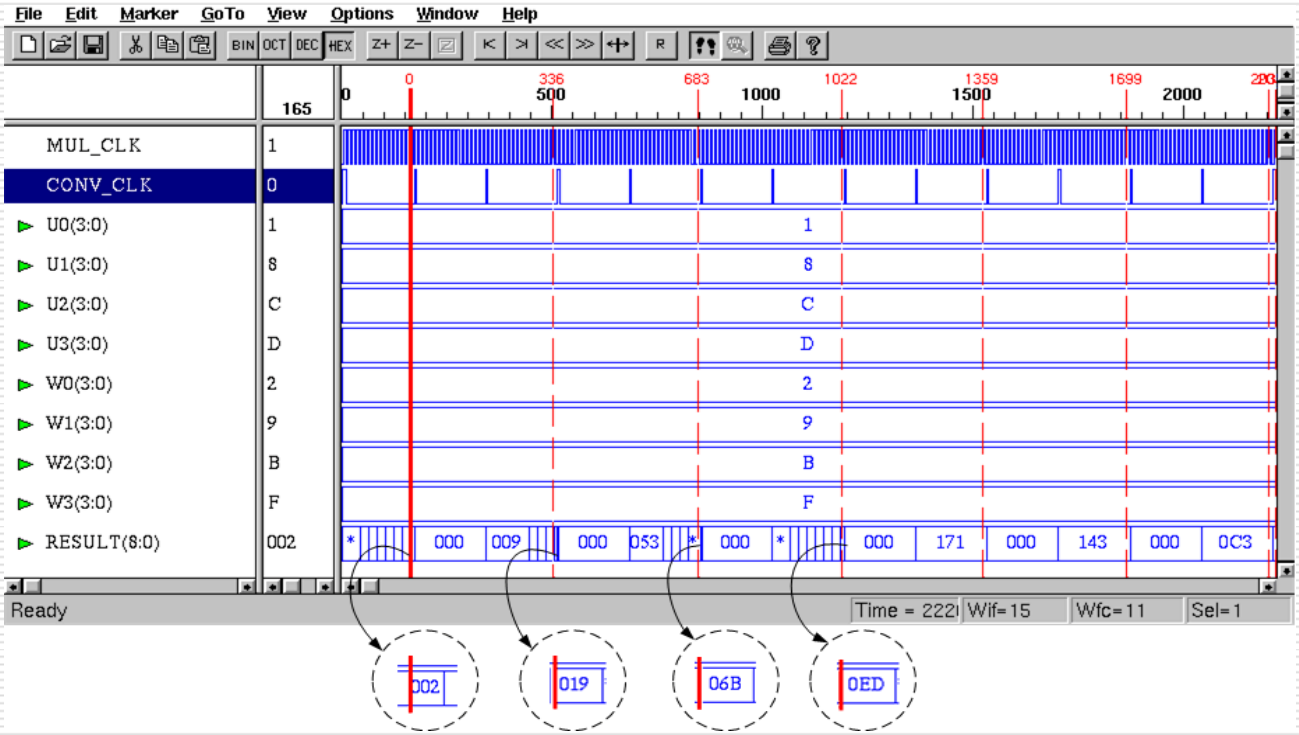
Super-Systolic Array

Making cells of a systolic array themselves systolic array as well results in even higher degree of concurrency, and even lower resource consumption, referring to the original systolic array as a super-systolic array.

Multiplier is designed again using systolic array, making systolic array for convolution a super-systolic array



Super-systolic array for convolution



Simulation waveform for super-systolic array for convolution

[Reference]

J.J. Lee and G.Y. Song,
"Implementation of the super systolic array for convolution,"
ASP-DAC 2003, pp.491-494, Jan. 2003.

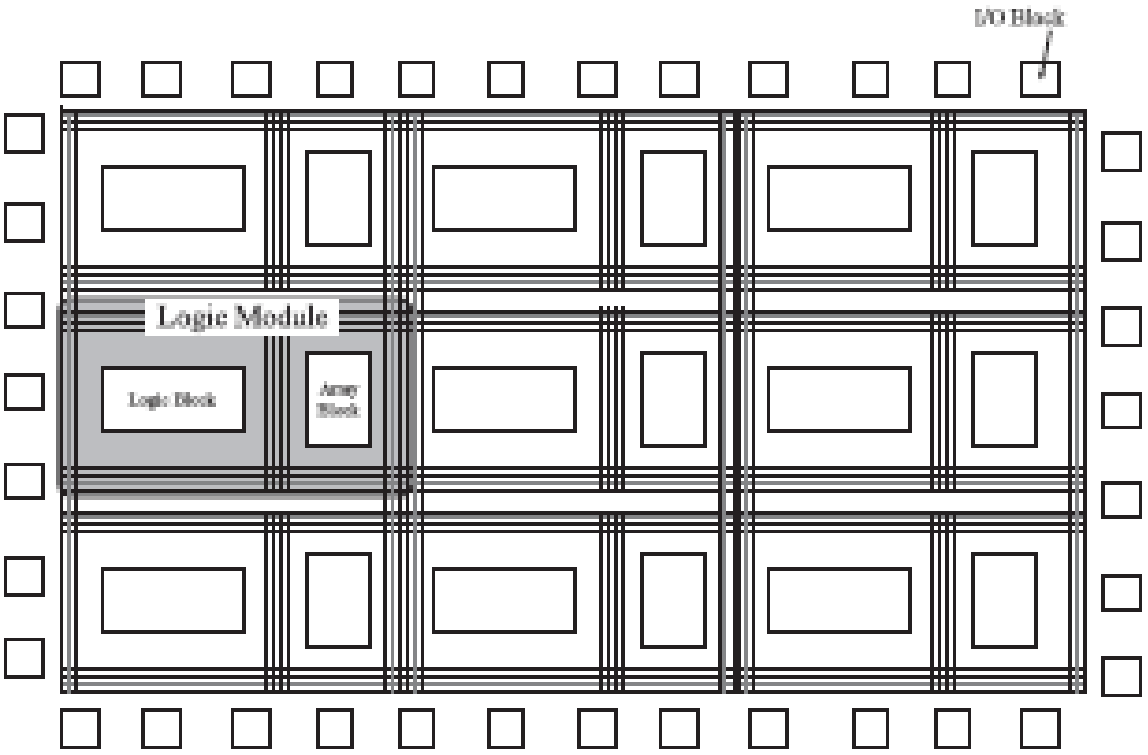
Application-Specific PLD

Motivation

While general-purpose routing architecture allows any SLICE to communicate directly with any other SLICEs in the FPGA architecture, it tends to substantially degrade performance.

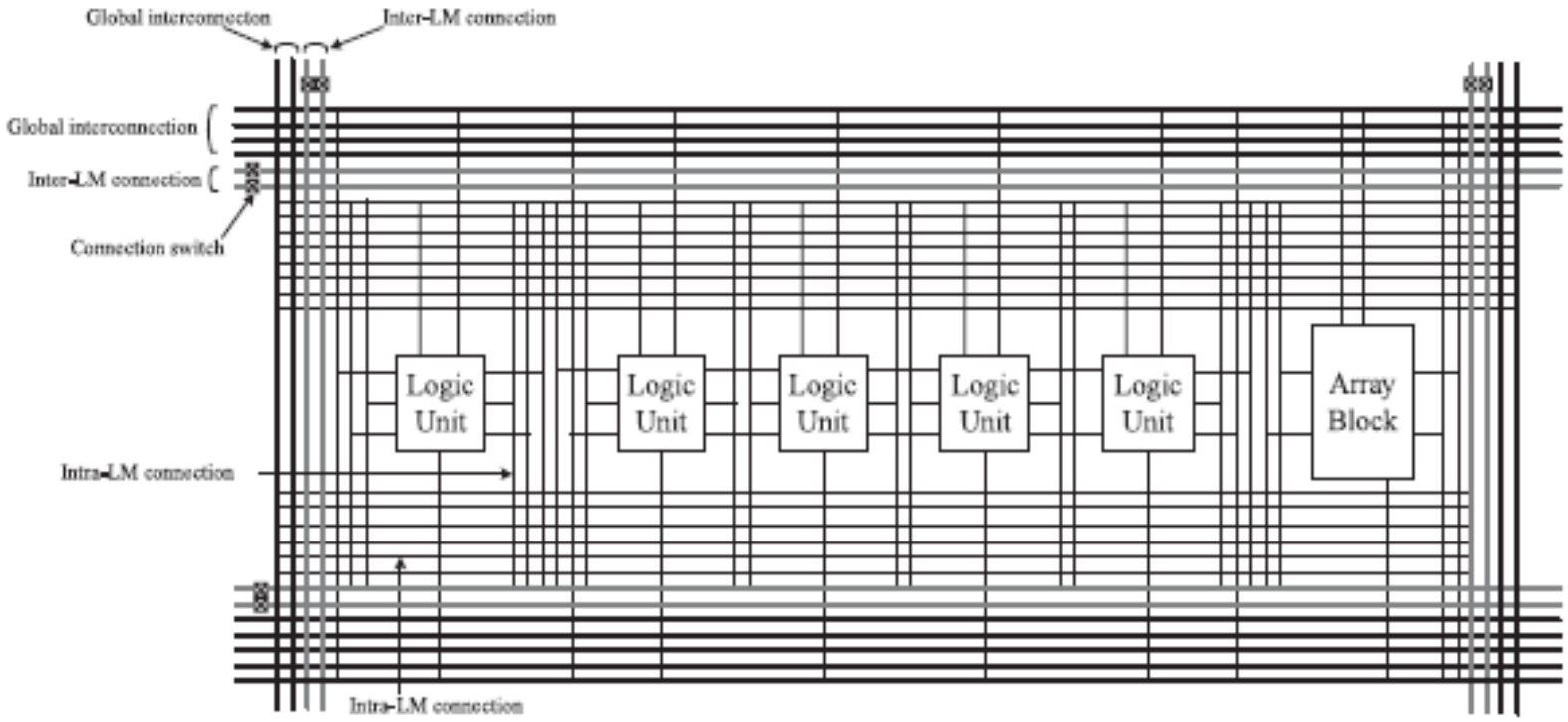
Proposed PLD architecture

A newly proposed PLD architecture aims to configure systolic array for operations with recurrence, resulting in regularity on layout, and especially on routing.



A proposed PLD architecture.

The architecture consists of configurable Logic Modules (LMs), Configurable I/O Blocks, and Programmable Interconnections to route signals.



Layout of a logic module.

There are three kinds of programmable interconnections: global interconnection, inter-LM connection and intra-LM connection.

How to implement Boolean functions

Each LB which implements 4-bit ripple carry adder or 4-bit systolic multiplier is based on a systolic array.

How to generalize

Interpret each Boolean function as a symmetric function,

Then, program the PSFG(programmable symmetric function generator) according to the corresponding symmetric function.

Symmetric function

[Def]

A switching function of n variables $f(x_1, x_2, \cdots, x_n)$ is called symmetric if and only if it is invariant under any permutation of its variables

[Thm]

The necessary and sufficient condition for a function $f(x_1, x_2, \cdots, x_n)$ to be symmetric is that it may be specified by a set of numbers $\{\mathbf{a}_1, \mathbf{a}_2, \cdots, \mathbf{a}_k\}$, where $0 \leq \mathbf{a}_i \leq n$, such that it assumes the value 1 when and only when $\mathbf{a}_i$ of the variables are equal to 1. The numbers in the set $\{\mathbf{a}_1, \mathbf{a}_2, \cdots, \mathbf{a}_k\}$ are called the a-numbers

Figure (a) shows symmetric indices of three-input Boolean function.

Figure (b), (c) show K-maps of two outputs, sum and carry, of a full adder, respectively. These two functions are symmetric functions.

ab \ c	0	1
00	S^0	S^1
01	S^1	S^2
11	S^2	S^3
10	S^1	S^2

(a)

ab \ c	0	1
00	0	1
01	1	0
11	0	1
10	1	0

(b)

ab \ c	0	1
00	0	0
01	0	1
11	1	1
10	0	1

(c)

(a) symmetric indices of three-input Boolean functions,

(b) $sum = a \oplus b \oplus c = S^{13}$

(c) Carry = ab+bc+ca = S^{23}

How about Non-symmetric functions?

It has been shown that a non-symmetric function can be symmetrized by repeating its variables.

[Reference]

S.B. Akers, “A rectangular logic array,” Computer, vol.C-21, no.8, pp.848-857, 1972.

K-map demonstrates the symmetrization by repeating a variable of a non-symmetric Boolean function: $F = a' + b$.

a \ b	0	1
0	1	1
1	0	1

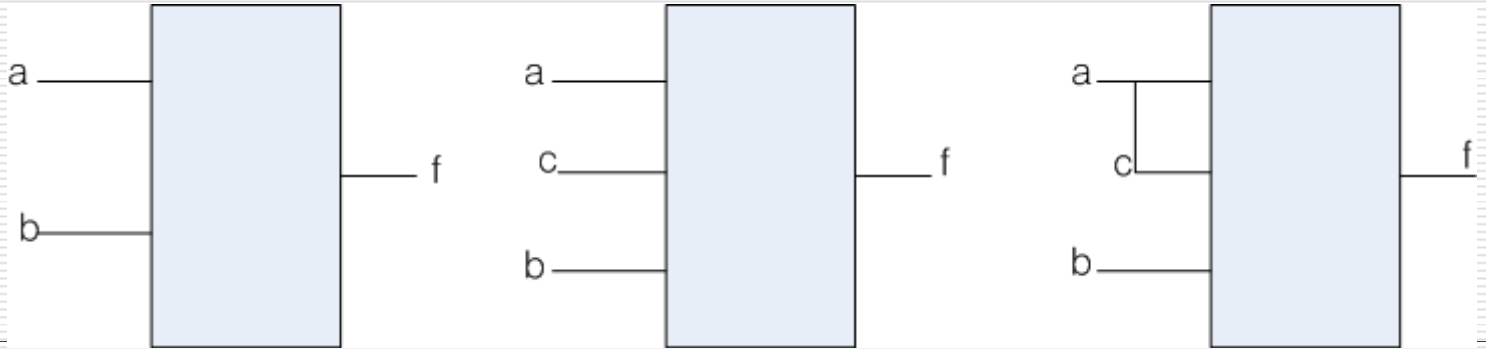
(a)

aa \ b	0	1
00	1	1
01	-	-
11	0	1
10	-	-

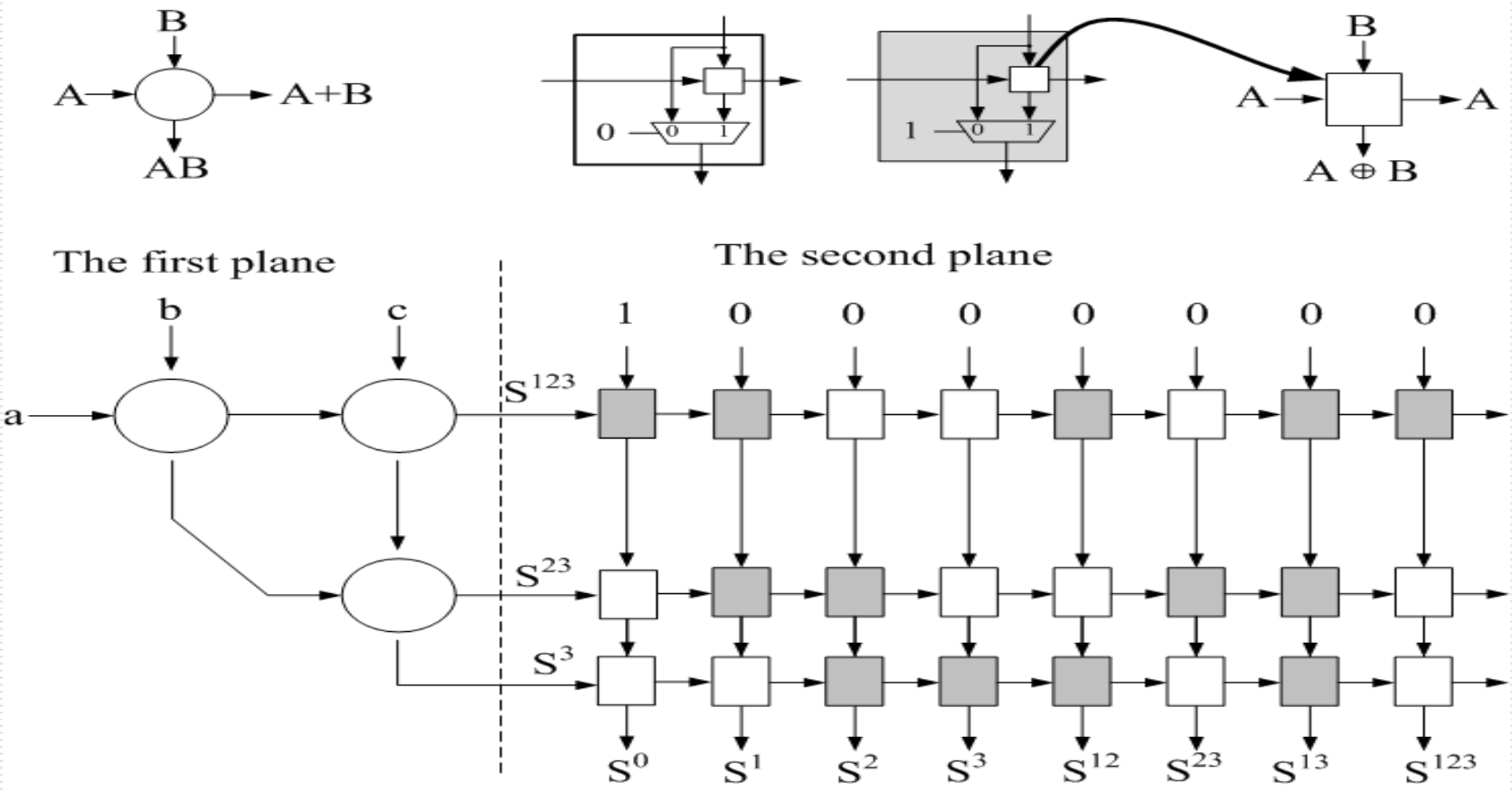
(b)

(a) Non-symmetric Boolean function, (b) symmetric Boolean function obtained by repeating variable *a*.

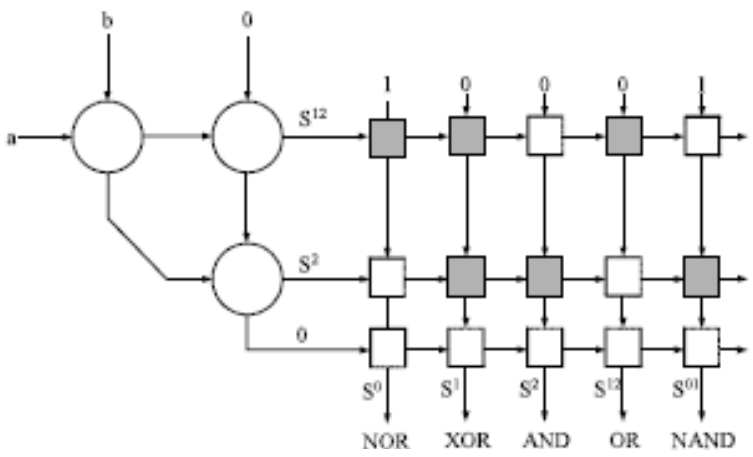
- add redundant variable c
- make minterms with a and c being complemented don't-cares
- make function symmetric by assigning 1 or 0 to don't-cares accordingly
- remove c by letting a =c



Programmable Symmetric Function Generator

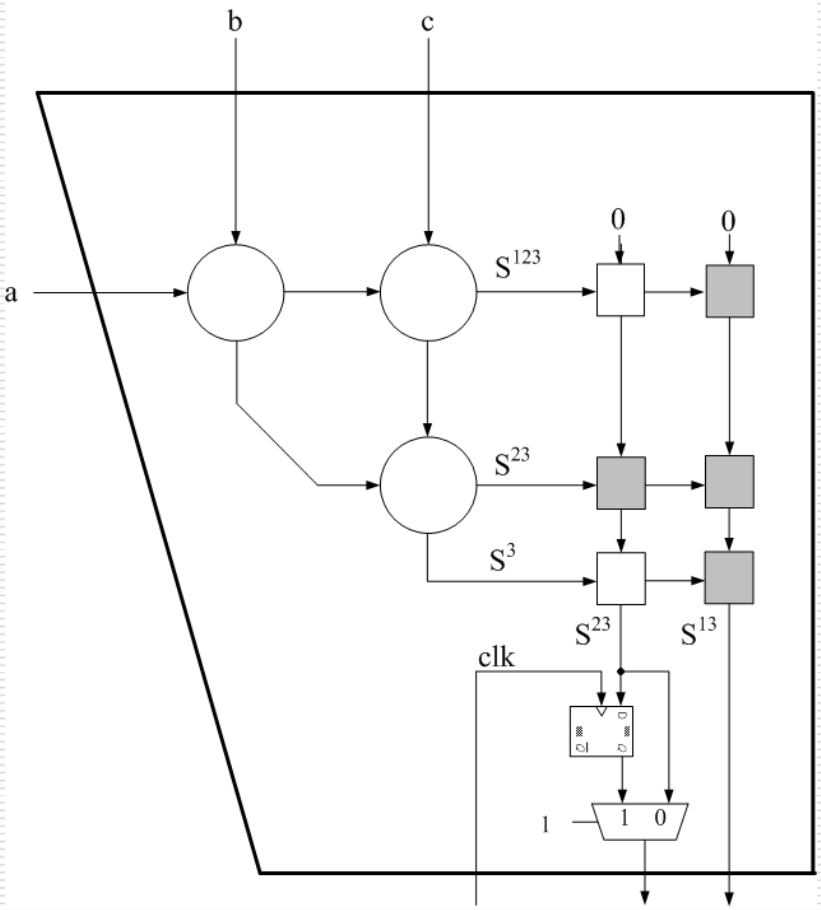


3-variable PSFG

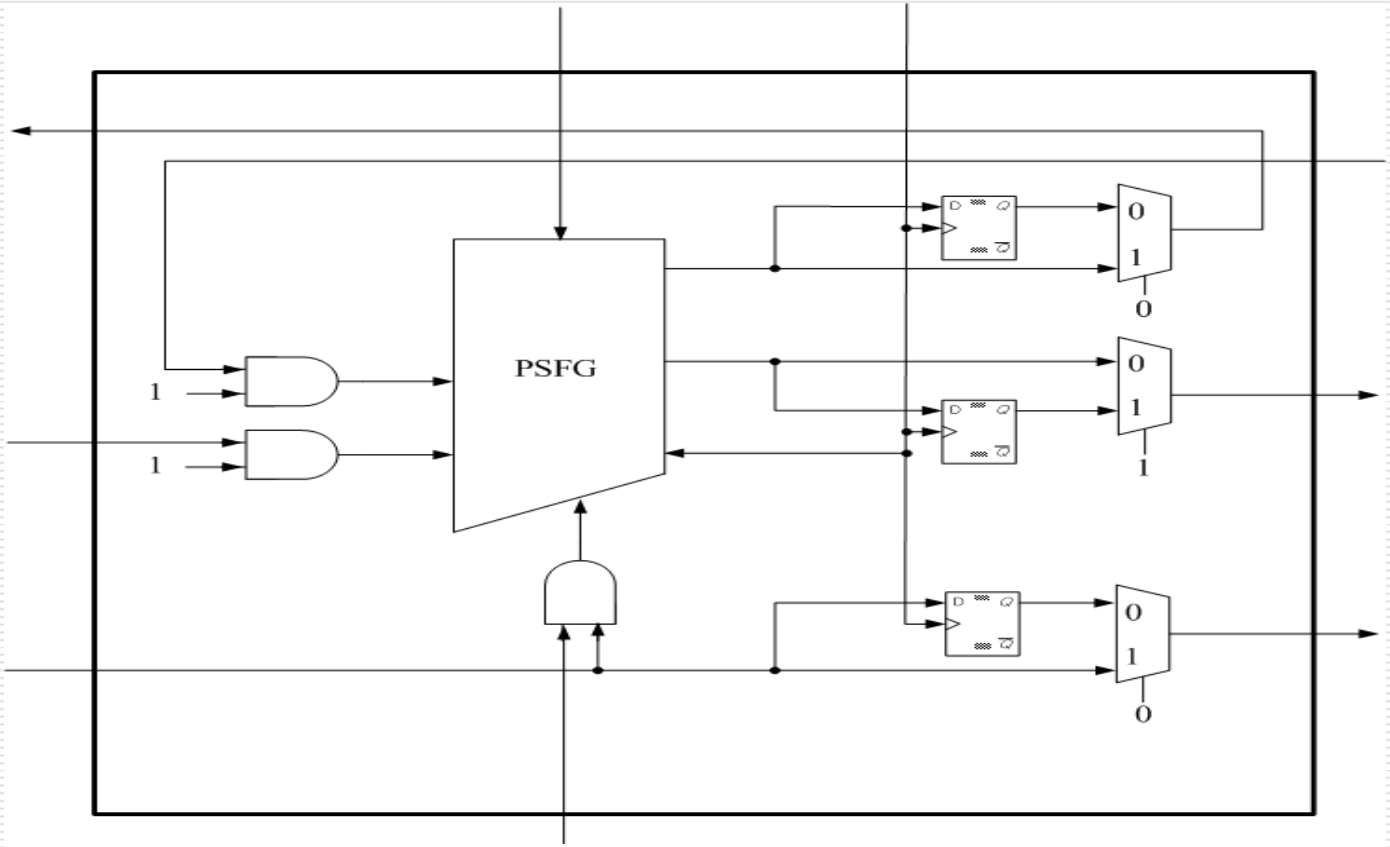


Realization of 2-variable symmetric functions.

PSFG can implement arbitrary 2- or 3-variable symmetric functions



Structure of a PSFG for a 1-bit Full-Adder



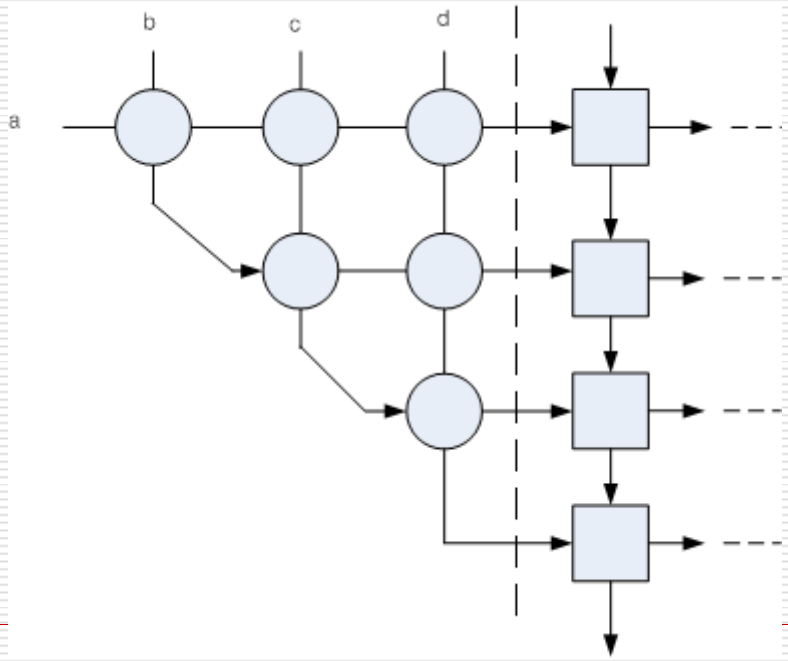
Structure of a logic unit for a systolic multiplier

How to expand 3-var. PSFG to more than 3-var.

First,

extending the LU based on 3-variable PSFG

-> feasible



but, constitute building blocks of different scale

so, extended LU will prevent the layout from being compact -> more unlikely

Second, a symmetric function with more than three inputs can be implemented as a cascade of 3-variable PSFG through EXOR-based Davio expansion.

In order to implement arbitrary *n-variable* symmetric function (*n >4) on the proposed PLD architecture we use* EXOR-based Davio expansion defined as:

For an arbitrary logic function $f(x_1, x_2, \ldots, x_n)$

$$f = f_0 \oplus x_1 f_2$$

where $f_0 = f(0, x_2, \ldots, x_n)$, $f_1 = f(1, x_2, \ldots, x_n)$, and $f_2 = f_0 \oplus f_1$ *with respect to* x_1.

[Reference]

T. Sasao, "EXMIN2: A simplification algorithm for exclusive-ORsum-of products expressions for multi-valued input two-valued output functions," IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst., vol.12, no.5, pp.621-632, May 1993.

cd \ ab	00	01	11	10
00	0	0	1	0
01	0	1	0	1
11	1	0	1	0
10	0	1	0	1

(a)

ab \ c	0	1
00	0	0
01	0	1
11	1	0
10	0	1

(b)

ab \ c	0	1
00	0	1
01	1	0
11	0	1
10	1	0

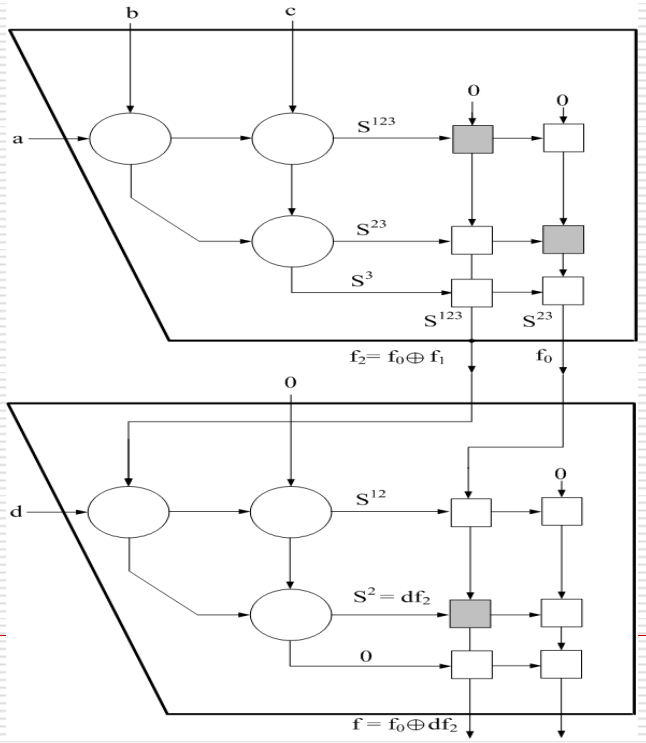
(c)

ab \ c	0	1
00	0	1
01	1	1
11	1	1
10	1	1

(d)

(a) $f(a,b,c.d) = S^{24}$, (b) $f_0=f(a,b,c,0)=S^2$, (c) $f_1=f(a,b,c,1) =S^{13}$, (d) $f_2=f_0 \oplus f_1=S^{123}$

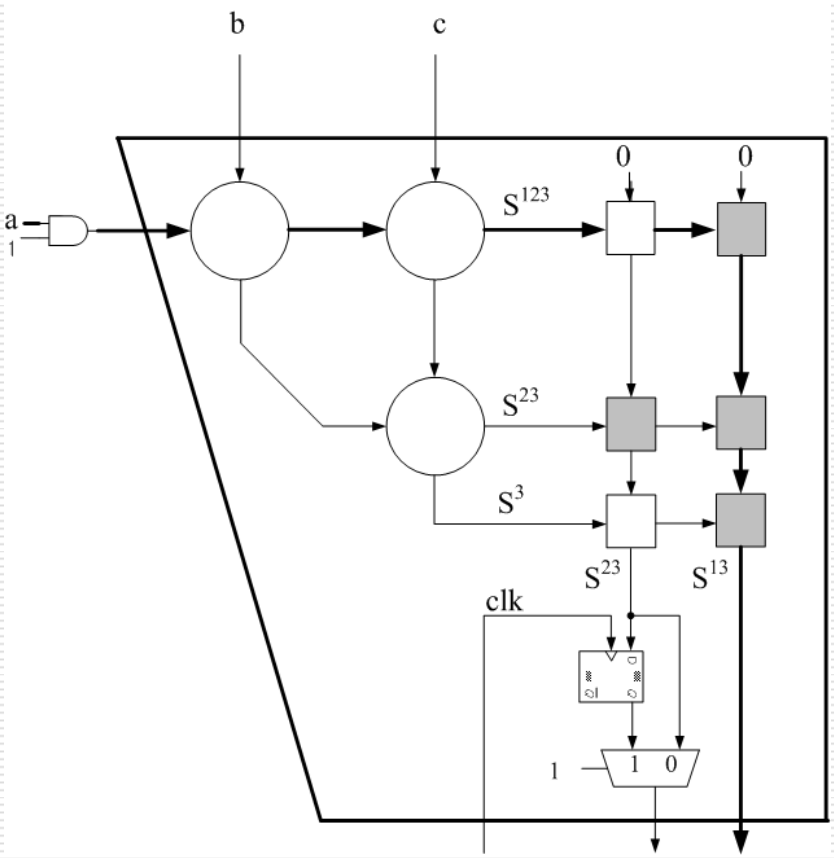
The function is implemented on the proposed PLD by cascading LUs after Davio expansion.



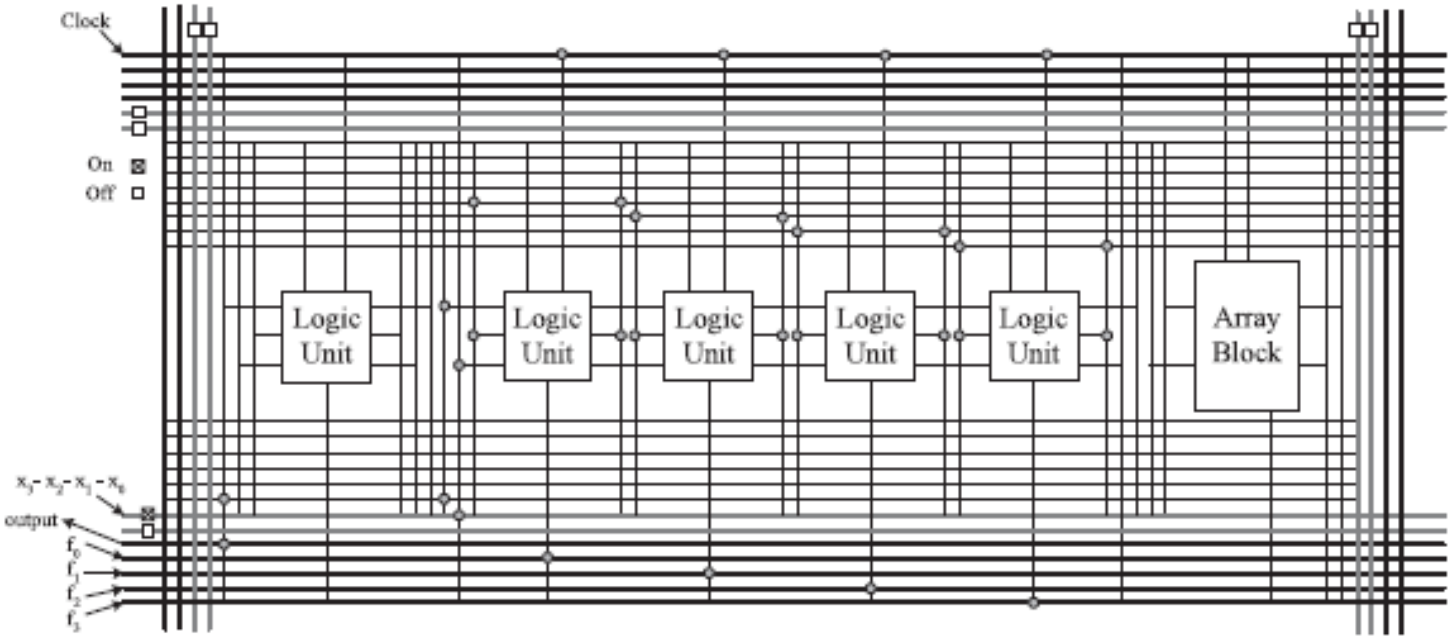
Implementation of a 4-varible symmetric function S^{24}.

Arbitrary non-symmetric random logic function can be implemented
on the proposed PLD by cascading LUs after the function is symmetrized by repeating its variables.

The critical path of a design is limited by only one 2-input AND gate and one PSFG as represented by bold line in the Fig.



PSFG structure of a design example



Schematic of the programmed interconnections for a systolic multiplier.

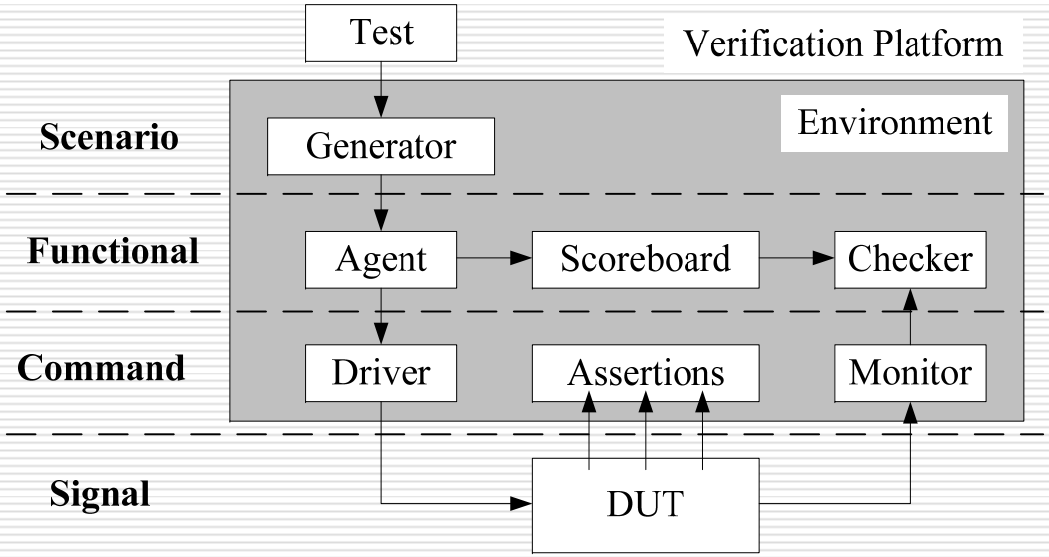
Conclusion

We propose a new application-specific PLD architecture aiming to configure systolic array for arithmetic operations with recurrence relation.

This architecture makes layout as well as routing to be regular.

It offers a significant alternative view on the programmable logic devices

The proposed PLD can achieve implementation results that are better than those achieved on existing FPGAs for such a design that is highly regular like arithmetic circuits.


Test
Verification Platform
Environment
Scenario
Generator
Functional
Agent
Scoreboard
Checker
Command
Driver
Assertions
Monitor
Signal
DUT

Transcript

```
#  [Environment] : gen_cfg ( ) -- Trans_num = [b]
#  [Generator] Operation : x = [1], h0 = [0110],h1 = [0101],h1 = [0001],h3 = [1011]
#  [Monitor]  operand_y = 0
#  [Driver] Operation : x = [0], h0 = [0000], h1 = [0000], h2 = [0000], h3 = [0000]
#  [Checker] Received : 0
#  [Monitor]  operand_y = 0
#  [Generator] Operation : x = [0], h0 = [1110],h1 = [1001],h1 = [0100],h3 = [0001]
#  [Checker] Received : 0
#  [Driver] Operation : x = [1], h0 = [0110], h1 = [0101], h2 = [0001], h3 = [1011]
#  [Generator] Operation : x = [0], h0 = [0110],h1 = [1010],h1 = [0001],h3 = [0001]
#  [Monitor]  operand_y = 0
#  [Driver] Operation : x = [0], h0 = [1110], h1 = [1001], h2 = [0100], h3 = [0001]
#  [Checker] Received : 0
#  [Monitor]  operand_y = 0
#  [Generator] Operation : x = [0], h0 = [0110],h1 = [0100],h1 = [0011],h3 = [1100]
#  [Checker] Received : 0
#  [Driver] Operation : x = [0], h0 = [0110], h1 = [1010], h2 = [0001], h3 = [0001]
#  [Generator] Operation : x = [1], h0 = [1011],h1 = [1100],h1 = [1100],h3 = [1101]
#  [Monitor]  operand_y = 0
#  [Driver] Operation : x = [0], h0 = [0110], h1 = [0100], h2 = [0011], h3 = [1100]
#  [Checker] Received : 0
#  [Monitor]  operand_y = 1
#  [Generator] Operation : x = [1], h0 = [1000],h1 = [0100],h1 = [1001],h3 = [1110]
#  [Checker] Received : 1
#  [Driver] Operation : x = [1], h0 = [1011], h1 = [1100], h2 = [1100], h3 = [1101]
#  [Generator] Operation : x = [1], h0 = [1000],h1 = [1000],h1 = [0100],h3 = [0100]
#  [Monitor]  operand_y = 0
#  [Driver] Operation : x = [1], h0 = [1000], h1 = [0100], h2 = [1001], h3 = [1110]
#  [Checker] Received : 0
#  [Monitor]  operand_y = 0
#  [Generator] Operation : x = [0], h0 = [0100],h1 = [1101],h1 = [0111],h3 = [1101]
#  [Checker] Received : 0
#  [Driver] Operation : x = [1], h0 = [1000], h1 = [1000], h2 = [0100], h3 = [0100]
#  [Generator] Operation : x = [1], h0 = [0110],h1 = [0101],h1 = [1101],h3 = [0101]
```

The results of a bit-level super-FIR filter

[Reference]

J.J. Lee and G.Y. Song,
"A New Application-Specific PLD Architecture",
IEICE Trans. on Fundamentals of Electronics,
Communications and Computer Sciences,
vol.E88-A, no.6, pp.1425-1433, June 2005.