Real Time Image Processing
using FPGA devices

Professor Michel PAINDAVOINE
Université de Bourgogne
Aile des Sciences de l ’Ingenieur
BP 400 - 21011 DIJON Cedex - France
email:paindav@u-bourgogne.fr
OUTLINE

I- Objectives of the Real Time Image Processing

II- Algorithm To Architecture Methodology

III- Digital filters applied to Image Processing

IV- Signal and Image Processing Algorithms Optimization

V- FPGA Implementation of Real Time Image Processing Algorithms using VHDL
OBJECTIVES

* Computation Time Limitation in the goal to obtain Real-Time:

Examples:
- Edge Detection using the Canny-Deriche Algorithm corresponds to 12 additions and 12 multiplications per pixel in less than 100ns (Pixel frequency=10MHz)

- Face Recognition using Neural Networks:
  Image of 200x200 pixels corresponds to 40K inputs for the Neural Network and it needs to process a 40Kx40K matrix in less than 40ms (25 images per second)
* Computer Size Limitation :

Example:

- High Speed Images:

  500 images per second, each image=1024x1024 pixels

  corresponds to a 500 Mpixels per second and needs large memories.

  Real Time Image Processing needs Algorithm to Architecture Methodology
Example: Real Time Imaging System

Applied to Human Movement Analysis

FAST CAMERA 500 IMAGES/S
CMOS Image Sensor from MICRON
Embedded Image processing inside FPGA

IMAGE Format: 1280x1024
Fast Smart Camera Architecture

FPGA:
- Sensor and interfaces control
- Embedded Real Time Image Processing

Memory:
SRAM: stores temporary data during processing

CMOS sensor (Micron)
MTM9M413
1.3 MPixels

EEPROM (Prg FPGA)

FPGA (Xilinx)
Virtex-II
XC2V3000
(3,000 gates)

USB2 Interface

SRAM Memory
(512 Mbytes)
Internal Camera view:

CMOS sensor board
FPGA board
Interface board

Camera back-face:

Connectors:
- SCSI
- Power supply
- USB2
- Video
- JTAG and Camera control
Real Time Calculation Component: FPGA

Storage of the parameters for the target detection,
Target detection in real time,
Target position transmission using USB2 interface.
Targets Detection Principle

PARAMETERS OF DETECTION:

• GREY LEVEL THRESHOLD
• MINIMUM LENGTH OF THE TARGET
• MAXIMUM LENGTH OF THE TARGET
DETECTION VISUALIZATION

IMAGE NO. 2452
Tracking Application in Real Time
(300 Images/sec)
Results for this implementation

• Real Time Movement Analysis working at 500 Images per second (Image Format: 1024x1280 pixels)

• Using Programmable Component (FPGA)

• Applications: Human Movement Analysis like Sport, Reeducative Medicine, Ergonomic Interfaces Design, Cinema,…

In order to increase performances it’s necessary to study other solutions:

Using of Algorithm to Architecture Methodology
OUTLINE

I- Objectives of the Real Time Image Processing

II- Algorithm To Architecture Methodology

III- Digital filters applied to Image Processing

IV- Signal and Image Processing Algorithms Optimization

V- FPGA Implementation of Real Time Image Processing Algorithms using VHDL
II- Algorithm To Architecture
Methodology Description

(Results research from French GDR-CNRS ISIS)
[http://gdr-isis.org]

◊ Choice and Test of the algorithm using a standard computer
◊ Comparison of different specific architectures (using CAD tools)
◊ Algorithm modification
◊ Architecture validation
◊ End modification of the architecture
Image Processing
Algorithms Classification

LOW LEVEL PROCESSING
- Filtering
- Segmentation
- FFT, ...

MEDIUM LEVEL PROCESSING
- Coding
- Compression

HIGH LEVEL PROCESSING
- Classification
- Pattern Recognition
- Measurements

Decision
New Architectures:
from the first Integrated Circuit in 1960 to ... Complex Architectures in 2010!

Source: Michel Robert
Moore law: from the sixties each 18 months, the number of transistors is multiplied by 2
Memories less and less expensive:

Today’s semiconductors are a million times cheaper than in the seventies

Price of 1 Mbit of memory:
- 1973: 75,000 euros
- 1977: 5,000 euros
- 1981: 400 euros
- 1984: 120 euros
- 1987: 30 euros
- 1990: 5 euros
- 1995: 0.5 euros
- 2000: 0.05 euros

Source: Siemens
Challenges for development of new architectures:
Algorithms are more and more complex

- Algorithms Complexity
- Processor/DSP
- Memory
- GAP
- 1G
- 2G
- 3G

# transistors (log)

year


- Analog
- Digital

- Cellular tel
  - UMTS
  - Multimedia

- Cellular tel
  - GSM

- Cellular phone
  - Radiocom 2000

...!
From now to ... 2010

<table>
<thead>
<tr>
<th></th>
<th>1998 / 0.25 μm</th>
<th>2008 / 0.08 μm</th>
</tr>
</thead>
<tbody>
<tr>
<td>Processor</td>
<td>10 MT</td>
<td>500 MT</td>
</tr>
<tr>
<td>ASIC</td>
<td>1 MT</td>
<td>50 MT</td>
</tr>
<tr>
<td>DRAM in 1 circuit</td>
<td>500 Mbits</td>
<td>50 Gbits</td>
</tr>
</tbody>
</table>

CMOS technology capabilities: source SIA

1998
ASIC
Function on chip
20-30%

2008
SOC
System on chip
70-80%

Source: Michel Robert
AAA Methodology

Algorithm adaptation

Algorithm
Matlab, C++

Data Flow

SynDex

Timing specification
Task repartition
Code Generation

Task

Task

Task

Task

Architecture adaptation

Architecture

Set of Components

DSP

FPGA

ASIC

Processor
I- Objectives of the Real Time Image Processing

II- Algorithm To Architecture Methodology

III- Digital filters applied to Image Processing

IV- Signal and Image Processing Algorithms Optimization

V- FPGA Implementation of Real Time Image Processing Algorithms using VHDL
III-1 Digital filtering operation

\[ y(n) = x(n) * h(n) \quad Z \quad Y(z) = X(z) \cdot H(z) \]

with

\[ H(z) = \frac{a_0 + a_1 z^{-1} + \ldots + a_I z^{-I}}{b_0 + b_1 z^{-1} + \ldots + b_J z^{-J}} \]

\[ Y(z) = \frac{a_0}{b_0} X(z) + \frac{a_1}{b_0} z^{-1} X(z) + \ldots + \frac{a_I}{b_0} z^{-I} X(z) - \frac{b_1}{b_0} z^{-1} Y(z) - \ldots - \frac{b_J}{b_0} z^{-J} Y(z) \]

\[ y(n) = \frac{a_0}{b_0} x(n) + \frac{a_1}{b_0} x(n-1) + \ldots + \frac{a_I}{b_0} x(n-I) - \frac{b_1}{b_0} y(n-1) - \ldots - \frac{b_J}{b_0} y(n-J) \]
Implementation using Matlab:

\[
\begin{align*}
&>> A = [a_0 \ a_1 \ \ldots \ a_I]; \\
&>> B = [b_0 \ b_1 \ \ldots \ b_J]; \\
&>> y = \text{filter}(A, B, x);
\end{align*}
\]

Synoptic Schematics:
III-2 Image Edge Detection with Sobel digital filter

\[
\begin{bmatrix}
-1 & -2 & -1 \\
0 & 0 & 0 \\
1 & 2 & 1
\end{bmatrix}
\]

\[
\begin{bmatrix}
-1 & 0 & 1 \\
-2 & 0 & 2 \\
-1 & 0 & 1
\end{bmatrix}
\]

\[
g(x,y) = |f(x,y) * h1| + |f(x,y) * h2|
\]

\[
g1(x,y) = f(x,y) * h1 \quad \text{et} \quad g2(x,y) = f(x,y) * h2
\]

\[
g1(x,y) = \left\{ f(x,y) + 2f(x-1,y) + f(x-2,y) \right\} - \left\{ f(x,y-2) + 2f(x-1,y-2) + f(x-2,y-2) \right\}
\]
\[ G_1(z) = F(z) + 2z^{-1}F(z) + z^{-2}F(z) - z^{-2N} \left[ F(z) + 2z^{-1}F(z) + z^{-2}F(z) \right] \]

\[ G_1(z) = F(z) \left[ 1 + z^{-1} \right] \left[ 1 + z^{-1} \right] - z^{-2N} F(z) \left[ 1 + z^{-1} \right] \left[ 1 + z^{-1} \right] \]

With \( N \): Number of Pixels per line
Absolute value calculation:

\[ g_1(x,y) \]

\[ \text{Abs}[g_1(x,y)] \]

\[ a_1(x,y) \]

\[ b_1(x,y) \]

\[ \text{Mux 2 to 1} \]

\[ 1 \]

\[ -g_1(x,y) \]

\[ \text{Mux} \]

\[ \text{Sign} \]

\[ \text{a1, b1, g1 and (-g1) on 8 bits} \]

\[ \text{Sign on 1bit} \]

\[ \text{Abs}[g1] \text{ on 8 bits} \]
Original Image with High Signal to Noise Ratio (SNR)

Original Image

Edge detection with Sobel Digital Filter
Original Image with Low Signal to Noise Ratio (SNR)

Original Image

Edge detection with Sobel Digital Filter

Countours are lost in the noise!
III-3 Edge detection using optimal filters:

**CANNY-DERICHE, SHEN-CASTAN, ...**

CANNY-DERICHE is based on 3 criteria:

- SNR increasing

- Best location

- Multiple responses elimination
Generalized Canny-Deriche filter description:

Image model:

\[ C(x) = 1 - \frac{e^{-sx}}{2} \quad \text{pour } x \geq 0 \]

\[ et \]

\[ C(x) = \frac{e^{sx}}{2} \quad \text{pour } x < 0 \]
Using of 2 filters: Derivative filter and Smoothing filter

If with consider the 3 Canny-Deriche criteria, we obtain for the horizontal direction 4 digital IIR filters:

**Z transform Derivative filter:**

\[
D^+(z) = \frac{a_1 z^{-1} + a_2 z^{-2}}{1 - a_3 z^{-1} + a_4 z^{-2} - a_5 z^{-3}} \quad (x \geq 0)
\]

\[
D^-(z) = \frac{a_1 z + a_2 z^2}{1 - a_3 z + a_4 z^2 - a_5 z^3} \quad (x < 0)
\]

**Z transform Smoothing filter:**

\[
L^+(z) = \frac{c_1 z^{-1} + c_2 z^{-2}}{1 + d_1 z^{-1} + d_2 z^{-2} + d_3 z^{-3}} \quad (x \geq 0)
\]

\[
L^-(z) = \frac{c_1 z + c_2 z^2}{1 + d_1 z + d_2 z^2 + d_3 z^3} \quad (x < 0)
\]
From Z space to spatial domain:

Example for $d^+(z)$:

$$d^+(z) = \frac{g^+(z^{-1})}{i(z^{-1})} = \frac{a_1 z^{-1} + a_2 z^{-2}}{1 - a_3 z^{-1} + a_4 z^{-2} - a_5 z^{-3}}$$

$$g^+(z^{-1}).(1 - a_3 z^{-1} + a_4 z^{-2} - a_5 z^{-3}) = i(z^{-1}).(a_1 z^{-1} + a_2 z^{-2})$$

$$g^+(z^{-1}) = a_1 z^{-1}.i(z^{-1}) + a_2 z^{-2}.i(z^{-1}) + a_3 z^{-1}.g^+(z^{-1}) - a_4 z^{-2}.g^+(z^{-1}) + a_5 z^{-3}.g^+(z^{-1})$$

as $Z^{-1}\{i(z^{-1})\} = i(x)$ and $Z^{-1}\{a z^{-k}.i(z^{-1})\} = a.i(x-k)$

$$g^+(x) = a_1.i(x-1) + a_2.i(x-2) + a_3.g^+(x-1) - a_4.g^+(x-2) + a_5.g^+(x-3)$$

Total of calculations: 3 Additions, 1 Subtraction and 5 Multiplications for one pixel and this for one filter!
Optimized Canny-Deriche Filter Implementation (cf Bourennane and Sarifuddin thesis)

\[ g(x,y) = i(x,y) + \left( l_x^+(x,y) + l_y^+(x,y) - l_x^-(x,y) - l_y^-(x,y) \right) \]

Image Memory

FIFO

LIFO

dx+ filter

dx- filter
Optimized Canny-Deriche filter implementation: Total electronics resources

Example for an image with 512x512 pixels coded on 8 bits each:

- **8 IIR filters:**
  - For each IIR filter: 3 additions, 1 subtraction and 5 multiplications
    so for the 8 IIR filters: 30 additions (24+4+1), 8 subtractions and 40 multiplications
    inside one FPGA Xilinx Virtex-II

- **4 FIFOs and 4 LIFOs:**
  inside the same FPGA Xilinx Virtex-II used for 8 IIR filters

- **Two Image Memories:**
  - each image memory stores results coded on 16 bits: 512x512x16bits (256Kx16 bits)
  - needs 512Kx16bits
    - outside FPGA, but with only one memory component
For example using of the Virtex-II XC2V3000 from Xilinx:

- 3,000 system gates organised in 14,336 slices
- 96 dedicated 18-bit x 18-bit multipliers blocks
- 1,728 Kbits of dual-port RAM in 18 K-bit SelectRAM resources
- 720 I/O pads allow to manage input from camera, exchange data between FPGA et Memory, output result of edge detection.
Optimized Canny-Deriche filter implemented in a simple board

Digital Camera \[ i(x,y) \] Xilinx FPGA Virtex-II
-8 IIR Filters
-4 FIFOs
-4 LIFOs

Image Memory
512Kx16bits

\[ g(x,y) \]
Results in SUN station

Modèle d’image:

\[ C(x) = 1 - \frac{e^{-sx}}{2} \quad \text{pour } x \geq 0 \]

et

\[ C(x) = \frac{e^{sx}}{2} \quad \text{pour } x < 0 \]

with \( s = 2^j \)
I- Objectives of the Real Time Image Processing

II- Algorithm To Architecture Methodology

III- Digital filters applied to Image Processing

IV- Signal and Image Processing Algorithms Optimization

V- FPGA Implementation of Real Time Image Processing Algorithms using VHDL
IV-1 Introduction

a) Two main goals for this optimization:
- Reducing time Computation
- Reducing size of circuits

b) Parallel Architecture classification: from Flynn

- SISD: Single Instruction Single Data
- SIMD: Single Instruction Multiple Data
- MISD: Multiple Instruction Single Data
- MIMD: Multiple Instruction Multiple Data

*SISD Architecture:*

PE: Processor Element
I: Instruction
**SIMD Architecture:**

**MISD Architecture:**

**Pipeline Architecture**

**MIMD Architecture:**
IV-2 Reducing time computation using parallel architectures

IV-2-1 Implementation with a SIMD approach: Systolic Network

\[ X_0 = z^{-1}X_i \]
\[ Y_0 = (aX_i + Y_i)z^{-1} \]
Example 1: FIR Filter with a 2 efficiency

\[ Y_1(z) = a_1 z^{-1} X_1(z) + a_2 z^{-3} X_1(z) + a_3 z^{-5} X_1(z) + a_4 z^{-7} X_1(z) \]

Question: FIR Filter with a 1 efficiency?

Example 2: IIR Filter with a 2 efficiency
IV-2-2 Implementation with a MIMD approach

(SynDEx Methodology: www.inria.fr, Yves Sorel)
Temporal diagram for Sobel Filter
a) **INTRODUCTION**

If we consider the following first order IIR filter:  
\[ y(n) = b \cdot x(n) + a \cdot y(n-1) \]  
with \( x(n) \) the input of the filter and \( y(n) \) the output of this filter.

Synoptic schematic of this filter is:
If we compute the previous for the (n+1) sample:
\[ y(n+1) = b \cdot x(n+1) + a \cdot y(n) \]
and with replacing \( y(n) \) with its previous equation:
\[ y(n+1) = b \cdot x(n+1) + a \cdot [b \cdot x(n) + a \cdot y(n-1)] \]
Or: \[ y(n+1) = b \cdot x(n+1) + ab \cdot x(n) + a \cdot y(n-1) \]
Expressing again \( y(n) \) using this new equation:
\[ y(n) = b \cdot x(n) + ab \cdot x(n-1) + a \cdot y(n-2) \]
This new equation includes an additional delay in the recursive loop of the filter

Time computation is divided by 2

Indeed, the sampling period is:

\[ T = \frac{1}{L} \text{Max} \left[ \frac{D_l}{M_l} \right] \]

with:

- \( D_l \): latency in loop \( l \)
- \( M_l \): number of delays (flipflops \( D \)) in loop \( l \)
- \( L \): sampling time for each flipflop
- \( (L \text{ times slower than sampling time } T \text{ of } x(n)) \)

In general \( D_l \) is a constant, so to in order to decrease \( T \), \( L \) and/or \( M_l \) have to be increased.

In the previous example, \( L \) is a constant and \( M_l \) moves from 1 to 2, thus \( T \) is divided by 2.
b) **LOOK-AHEAD WITH CLUSTERED POLES**

\[
H(z) = \frac{N(z)}{D(z)} = \frac{N(z)}{1 - \sum_{i=1}^{N} a_i z^{-i}}
\]

Exemple :

\[
H(z) = \frac{1}{1 - \frac{5}{4} z^{-1} + \frac{3}{8} z^{-2}} \rightarrow \text{Pôles} : \frac{1}{2} \text{ et } \frac{3}{4}
\]

\[\rightarrow \text{Stable recursive (IIR) – second order filter}\]

Diagram of the second-order filter with clustered poles.
Modification of $H(z)$ in order to eliminate the $z^{-1}$ term

$$H(z) = \frac{1}{1 - \frac{5}{4}z^{-1} + \frac{3}{8}z^{-2}} \cdot x \cdot \frac{1 + \frac{5}{4}z^{-1}}{1 + \frac{5}{4}z^{-1}} = \frac{1 + \frac{5}{4}z^{-1}}{1 - \frac{19}{16}z^{-2} + \frac{15}{32}z^{-3}}$$

We obtain a new third order IIR Filter: acceleration by 2 (2 D Flip-flops):

$\rightarrow$ Poles: $\frac{1}{2}, \frac{3}{4}$ et $-\frac{5}{4}$

$\rightarrow$ Instable Filter
c) **LOOK-AHEAD WITH SCATTERED POLES**

\[
H(z) = \frac{N(z)}{D(z)} = \frac{N(z)}{1 - \sum a_i z^{-i}} = \frac{\prod_{k=1}^{M-1} D(z e^{j2\pi k/M})}{\prod_{k=0}^{M-1} D(z e^{j2\pi k/M})}
\]

With \(M\): number of pipeline stages

**Example 1: First order recursive filter**

\[
H(z) = \frac{N(z)}{D(z)} = \frac{1}{1 - az^{-1}} \quad \text{with pole } z = a \quad \text{stable if } -1 < a < 1
\]

**Pour \(M = 3\) Addition Poles and Zeros in**

\[
z = a.e^{j2\pi/3} \quad \text{and in } z = a.e^{-j2\pi/3} = a.e^{-j4\pi/3}
\]

\[
H(z) = \frac{(z - a.e^{j2\pi/3}),(z - a.e^{-j2\pi/3})}{(1 - az^{-1}),(z - a.e^{j2\pi/3}),(z - a.e^{-j2\pi/3})} = \frac{1 + az^{-1} + a^2 z^{-2}}{1 - a^3 z^{-3}}
\]
Example 2: Second order recursive filter

\[ H(z) = \frac{N(z)}{D(z)} = \frac{1}{1-(r_1 + r_2)z^{-1} + r_1r_2z^{-2}} = \frac{1}{(1-r_1z^{-1})(1-r_2z^{-1})} \]

→ 2 poles \( r_1 \) and \( r_2 \) Filter stable if \(-1 < r_1 < 1 \) and if \(-1 < r_2 < 1 \)

For \( M = 3 \) Addition of Poles and Zeros in:

\[ z = r_1 e^{\pm \frac{2\pi}{3}} \text{ and } z = r_2 e^{\pm \frac{2\pi}{3}} \] which brings stability of the filter.

\[ H(z) = \frac{1}{(1-r_1 e^{\frac{2\pi}{3}}z^{-1})(1-r_1 e^{-\frac{2\pi}{3}}z^{-1})(1-r_2 e^{\frac{2\pi}{3}}z^{-1})(1-r_2 e^{-\frac{2\pi}{3}}z^{-1})} \]

\[ H(z) = \frac{1+(r_1 + r_2)z^{-1}+(r_1^2 + r_1r_2 + r_2^2)z^{-2} + r_1r_2(r_1 + r_2)z^{-3} + r_1^2 r_2^2 z^{-4}}{1-(r_1^3 + r_2^3)z^{-3} + r_1^3 r_2^3 z^{-6}} \]
Numerical application:

\[ H(z) = \frac{1}{1 - \frac{5}{4}z^{-1} + \frac{3}{8}z^{-2}} \rightarrow \text{Poles: } r_1 = \frac{1}{2} \text{ and } r_2 = \frac{3}{4} \]

(Here \( N=2 \) and \( M=3 \))

\[ H(z) = \frac{1 + \frac{5}{4}z^{-1} + \frac{19}{16}z^{-2} + \frac{15}{32}z^{-3} + \frac{9}{64}z^{-4}}{1 - \frac{35}{64}z^{-3} + \frac{27}{512}z^{-6}} \]

Architecture Complexity:
- Nb Multiplications non recursive part = \( N(M-1) \)
- Nb Multiplications recursive part = \( N \)
So a total of \( N.M \) multiplications (here \( N.M=6 \))
IV-3 Reducing time computation using Optimized arithmetic operators

IV-3-1 Optimized adders

- Ripple Carry Adder
- Carry look-ahead adder
- Carry selected

IV-3-2 Optimized multipliers

- Addition and Shift
- Braun Multiplier
- Booth algorithm
IV-3 Reducing size of circuits

**Simple solution:** limitation of the number of bits for data and coefficients

**Using of fixed point arithmetic:**

\[
\begin{align*}
&\text{Sign} \quad \text{« Decimal » part} \quad \text{Fractionnal part} \\
&a_n \quad a_{n-1} a_{n-2} \ldots \ldots a_1 a_0 \quad a_{-1} a_{-2} \ldots \ldots a_{-p}
\end{align*}
\]

**Limitation of p value (nb of bits of fractionnal part):**

Generation of an error between exact value and approximated value:

\[
a = a' + e
\]

with \(a=\) troncated coefficient, \(a'=\)exact coefficient and \(e=\)error
Example:

\[a'=0.26\]

\[\begin{array}{lcl}
P=1 & a=(0.5)_{10}=(0.1)_{2} & e=0.24 \\
P=2 & a=(0.25)_{10}=(0.01)_{2} & e=-0.01 \\
P=3 & a=(0.25)_{10}=(0.010)_{2} & e=-0.01 \\
P=4 & a=(0.3125)_{10}=(0.0101)_{2} & e=0.0525 \\
P=5 & a=(0.28125)_{10}=(0.01001)_{2} & e=0.0215 \\
P=6 & a=(0.265625)_{10}=(0.010001)_{2} & e=0.005625 \\
P=7 & a=(0.2578125)_{10}=(0.0100001)_{2} & e=-0.0021875 \\
P=8 & a=(0.26171875)_{10}=(0.01000011)_{2} & e=0.00171875 \\
\end{array}\]
If we want to implement the operation \( a \cdot x(n) \): we consider that \( x(n) \) is coded on 8 bits: \( d7d6d5\ldots d1d0 \)

1) with \( P=2 \) \( a=(0.01)_2 \)

\[ \begin{array}{c}
d7d6d5\ldots d1d0 \\
\hline
00d7\ldots d3d2
\end{array} \]

2 Right Shifts

2) with \( P=8 \) \( a=(0.01000011)_2 \) → Using of an adder-shift multiplier:

\[ \begin{array}{c}
d7d6d5d4d3d2d1d0 \\
X \quad 01000011 \\
d7d6d5d4d3d2d1d0 \\
d7d6d5d4d3d2d1d0 \\
d7d6d5d4d3d2d1d0 \\
P14P13P12P12P10P9P8P7P6P5P4P3P2P1P0
\end{array} \]
We consider the following digital filter:

\[ y(n) = x(n) + 0.3y(n-1) - 0.02y(n-2) \]

5) Compute the Transfer Function \( H_1(z) \) of this filter

6) Give the schematic of this filter

7) We want to accelerate this filter using the Scattered look-ahead pipelining. Give the new Transfer Function \( H_2(z) \) of this filter for a pipeline \( M=3 \)

4) Give the schematic of this new filter

5) We want **to evaluate the** number of operators corresponding to the calculation of \( 0.3y(n-1) \). If we consider that the coefficient (0.3) and \( y(n-1) \) are coded on 8 bits, give the schematic of this operation.
OUTLINE

I- Objectives of the Real Time Image Processing

II- Algorithm To Architecture Methodology

III- Digital filters applied to Image Processing

IV- Signal and Image Processing Algorithms Optimization

V- FPGA Implementation of Real Time Image Processing Algorithms using VHDL
V-1 VHDL Presentation

V-1-1 Introduction

-VHDL = VHSIC Hardware Description Language
-VHSIC = Very High Scale Integrated Circuit

-VHDL developed at the end of seventies by DOD
  (DOD=USA Departement of Defense)
-First Standard: IEEE 1076 in 1987
-Second Standard: IEEE 1164 in 1993 (multi-valuated Logic)

VHDL Advantages vs Schematic description:
- High level description: fast debug
- Flexibility: Intelectual Property (IP) Re-Use
- Choice of tools (Cadence, Mentor Graphics, …..)
- Choice of suppliers (FPGA: Xilinx, Altera….. and
  ASIC: STmicroelectronics, AMS, ….)
- VHDL block can be included inside a schematic
- Infinite logic gates (difficult in schematic > 10K gates)
3 Different description levels:

- VHDL level (High level):
  -> VHDL Behavioral: procedurale description
  -> VHDL structural or RTL (Register To Logic)

- Schematic level: description with logic cells

  a
  s
  b
  y

- Transistors level: layout
- Structure of VHDL language: Entity/Architecture pair:

-- Libraries declarations:
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- External logic block description:
entity component_name is
Port (list of input/output);
end component_name;

-- Internal logic block description:
architecture component_name_arch of component_name is
[external components declaration]
[types declaration]
[signals declaration]
begin
description area of the component
end component_name;
a) **Multiplexer 2 to 1**

<table>
<thead>
<tr>
<th>s</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>b</td>
</tr>
<tr>
<td>1</td>
<td>a</td>
</tr>
</tbody>
</table>

\[ y = a \cdot s + b \cdot \overline{s} \]

**mux21 1 bit:**

**mux21 8 bits:**
-- Multiplexer 2 to 1:

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity mux21_1 is
  Port ( a : in std_logic_vector(7 downto 0);
        b : in std_logic_vector(7 downto 0);
        s : in std_logic;
        y : out std_logic_vector(7 downto 0));
end mux21_1;

architecture mux2_1_arch of mux21_1 is
begin
  y<=a when (s='1') else b;
end mux2_1_arch;
b) Adder

-> Ripple Carry Adder:

Full Adder (FA) 1 bit

<table>
<thead>
<tr>
<th>$b_i$</th>
<th>$a_i$</th>
<th>$c_i$</th>
<th>$c_{i+1}$</th>
<th>$s_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

$s_i = a_i \text{xor} b_i \text{xor} c_i$

$c_{i+1} = c_i \cdot a_i \text{ or } b_i \cdot c_i \text{ or } b_i \cdot a_i$
Adder 4 bits:

-- Additionneur à propagation de retenue 4 bits

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity add4 is
  Port ( a : in std_logic_vector(3 downto 0);
         b : in std_logic_vector(3 downto 0);
         cin : in std_logic;
         s : out std_logic_vector(3 downto 0);
         cout : out std_logic);
end add4;
architecture add4_arch of add4 is
signal c: std_logic_vector(4 downto 0);
begin
  process(a,b,cin)
  begin
    c(0) <= cin;
    for i in 0 to 3 loop
      s(i) <= a(i) xor b(i) xor c(i);
      c(i+1) <= (c(i) and b(i)) or (c(i) and a(i)) or (b(i) and a(i));
    end loop;
    cout <= c(4);
  end process;
end add4_arch;
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity register is
  Port ( clk : in std_logic;
        D: in std_logic_vector(7 downto 0);
        Q : out std_logic_vector(7 downto 0));
end register;

architecture register_arch of register is
begin
  bascule3: process(clk)
  begin
    if rising_edge(clk) then  -- IEEE1164 function
      Q <= D;
    end if;
  end process;
end register_arch;
V-2 FPGA Implementation of Real Time Image Processing Algorithms using VHDL

V-2-1 FPGA description:
- Examples from Xilinx (www.Xilinx.com): Spartan and Virtex Families
- Xilinx tools (VHDL, Simulation, bitstream generation): Webpack and Modelsim (free)

V-2-2 Filter Implementation:

Example: \( y(n) = x(n) + \frac{5}{4}y(n-1) - \frac{3}{8}y(n-2) \) with \( x(n) \) coded on 8 bits and \( y(n) \) coded on 16 bits
a) Schematic solution

\[ x(n) \quad + \quad \frac{5}{4} \quad D \quad 5 \frac{7}{8} \quad D \quad + \quad x \quad - \quad \frac{3}{8} \quad \longrightarrow \quad y(n) \]

b) VHDL solution

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity filter is
  Port ( clk : in std_logic;
          x: in unsigned(7 downto 0);
          y : out unsigned(15 downto 0));
end filter;
```
architecture filtre_arch of filtre is
signal y1, y2, ytemp: unsigned (y’range);
begin
process(clk)
begin
if rising_edge(clk) then
    ytemp <= x + shr((y1+y1+y1+y1+y1),"100") - shr((y2+y2+y2),"1000");
    y2<=y1;
    y1<=ytemp;
end if;
end process;
y <= ytemp;
end filtre_arch;