Hes·so VALAIS WALLIS
School of Engineering

Hes·so
Haute Ecole Spécialisée de Suisse occidentale
Fachhochschule Westschweiz
University of Applied Sciences and Arts Western Switzerland

Degree Programme
Systems Engineering

Major Infotronics

Bachelor's thesis
Diploma 2019

Fracheboud Loïc

Video processing on SoC

Professor
Corthay François

Expert
Faure Pascal

Submission date of the report
23.08.2019

HES-SO Valais

SYND	ETE	TEVI
X	X	X

Données du travail de diplôme
Daten der Diplomarbeit

FO 1.2.02.07.CB
haj/11/01/2016

Filière / Studiengang **SYND**	Année académique / *Studienjahr* **2018/19**	No TD / *Nr. DA* **it/2019/84**
Mandant / *Auftraggeber* ☒ HES—SO Valais ☐ Industrie ☐ Etablissement partenaire *Partnerinstitution*	Etudiant / *Student* **Loïc Fracheboud** Professeur / *Dozent* **François Corthay**	Lieu d'exécution / *Ausführungsort* ☒ HES—SO Valais ☐ Industrie ☐ Etablissement partenaire *Partnerinstitution*
Travail confidentiel / *vertrauliche Arbeit* ☐ oui / ja [1] ☒ non / nein	Expert / *Experte (données complètes)* **Pascal Faure** Oculox Technologies SA, Via Industria 3, 6933 Muzzano	

Titre / *Titel*

Video processing on SoC

Description / *Beschreibung*

The aim of the diploma work is to develop the base for an image processing task on a Xilinx Zynq SoC. In this system, the ARM Cortex-A9 based Processing System (PS) will stream images to the Programmable Logic (PL) which will perform an image processing task on the fly.

Tasks to realize:

- Put together a Zynq design environment
- Develop a system where the PS can stream data to a component in the PL which does a very simple task and stream the result back to the PS
- Develop a more complex component implementing an image processing task in real-time
- Investigate the possibilities of Direct Memory Access (DMA) for the image transfers

Signature ou visa / *Unterschrift oder Visum*

Responsable de l'orientation / filière
Leiter der Vertiefungsrichtung / Studiengang:

.....

Etudiant / *Student* :

.....

Délais / *Termine*

Attribution du thème / *Ausgabe des Auftrags:*
13.05.2019

Présentation intermédiaire / *Zwischenpräsentation*
17.06.2019

Remise du rapport / *Abgabe des Schlussberichts:*
23.08.2019, 12:00

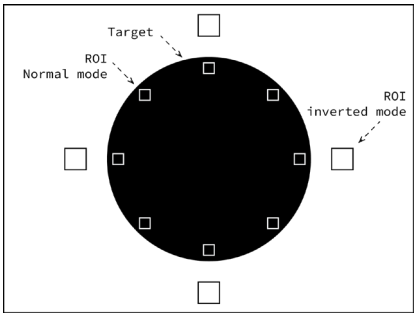
Expositions / *Ausstellungen der Diplomarbeiten:*
28, 29 – 30.08.2019

Défense orale / *Mündliche Verfechtung:*
02 – 05.09.2019

[1] *Par sa signature, l'étudiant-e s'engage à respecter strictement la directive DI.1.2.02.07 liée au travail de diplôme.*
Durch seine Unterschrift verpflichtet sich der/die Student/in, sich an die Richtlinie DI.1.2.02.07 der Diplomarbeit zu halten.

Rapport reçu le / *Schlussbericht erhalten am* ..... Visa du secrétariat / *Visum des Sekretariats* .....

Video processing on SoC



Graduate

Fracheboud Loïc

Objectives

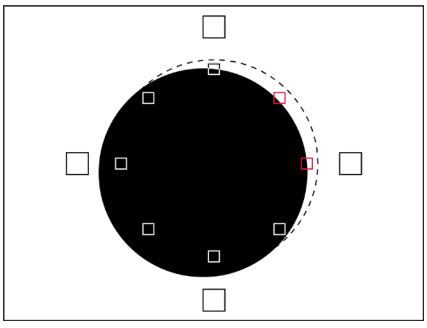
The goal of this project was to perform real-time video processing tasks on a Zynq SoC with a dedicated development environment embedded in a virtual machine.

Methods | Experiences | Results

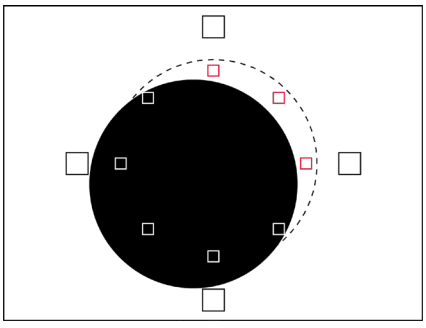
Eye tracking is a wide and emergent technology used in many fields such as medicine or marketing.
Traditionally, eye tracking technologies are typically based on direction, amplitude and location of the sight.

In this work, the movement was defined by tracking the limit between the white part of an eye (sclera) and the dark part, the iris. Those two elements benefits from the high colour contrast existing between them.
Practically, groups of pixels were defined in a target and their median value was monitored. As soon as this median value changed, a flag was raised to indicate a potential movement.
Difficulties of the project resided in the speed available in the system as all data must be received and processed in the small timeframe imposed by the high-speed camera.

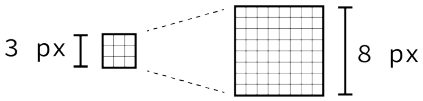
To reach this objective, a real-time video processing was designed and destinated to be embedded on a Zynq FPGA. The system has been then tested with several test benches and results showed that its tracking was efficient in several different conditions.



The eye (schematized as the black dot) can move slightly within a certain tolerance range without trigger a response.
Activated sensor are marked in red.



If the eye moves beyond the tolerance range, a response is triggered and a movement is detected.



Bachelor's Thesis
| 2019 |

Degree programme
Systems Engineering

Field of application
Major Infotronic

Supervising professor
Dr Corthay François
françois.corthay@hevs.ch

Hes·SO // VALAIS WALLIS

School of Engineering

Video processing on SoC

Bachelor thesis

REPORT

Abstract

The aim of the diploma work is to develop the base for an image processing task on a Xilinx Zynq SoC. In this system, the ARM Cortex-A9 based Processing System (PS) will stream images to the Programmable Logic (PL) which will perform an image processing task on the fly.

Student : Loic FRACHEBOUD

Teacher : François CORTHAY

Expert : Pascal FAURE

From : HES-SO Valais-Wallis

V1.0

26.08.2019

Contents

Figures and Tables

List of Figures

List of Tables

Listings

Preface

1 Acknowledgement

As I'm reaching the end of my thesis I would like to thank some people who were there during my work.

Thank to my professor Mr. François Corthay for his presence and help.

My coffee breaks would have not been the same without those lovely Marc and Eliot, thanks for that.
Speaking of breaks, Tristan and Adam, thanks for the ping-pong games.

Thanks to Juju for its priceless knowledge with Publisher, Amara for her expertise of LaTeX and Coach for its supportive words when times were hard.

A special thanks to the storm of the 11th of August, which gave me the opportunity to test my hardware under rough conditions.

Finally, a huge thanks to my supportive girlfriend, Lou, who took endless hours to review my thesis and spot my missing *s*.

2 Glossary

It is assumed that the reader has basic knowledge in programmable logic, C and hardware description in general. Acronyms used in this work and their signification are listed below to avoid any confusion.

3 Conventions

The present document is based on the following conventions.

- *Italic* indicates a signal
- SMALL CAPITALS identifies blocks, states or processes depending of the context

An itemized list is used when concepts are employed without any order while an enumeration indicates a temporal or hierarchical dependence.

Acronyms

AXI Advanced eXtensible Interface, A Xilinx AMBA (Advanced Microcontroller Bus Architecture) based bus.

DDR Double Data Rate.

DMA Direct Memory Access.

FIFO First In - First Out.

FPGA Field-Programmable Gate Array.

FPS Frames Per Second.

FPU Floating Point Unit.

GPIO General Purpose Input/Output.

HDL Hardware Description Language.

HDMI High-Definition Multimedia Interface.

HEI School of Engineering - *Haute Ecole d'Ingénieur*.

HES-SO University of Applied Sciences and Arts Western Switzerland - *Haute Ecole Spécialisée de Suisse occidentale*.

IP Intellectual Property, in this context it defines a block which contains hardware description for block design.

LED Light Emitting Diode.

LSB Least Significant Bit.

MSB Most Significant Bit.

PL Programmable Logic.

PS Processing System.

PWM Pulse Width Modulation.

ROI Region Of Interest.

SD-Card Secure Digital Card, a non-volatile type of memory card for.

SoC System on Chip.

TCL Tool Command Language.

USB Universal Serial Bus.

VGA Video Graphic Array, video format used as 640/480 pixels in greyscale in this work.

VHDL VHSIC (Very High Speed Integrated Circuits) Hardware Description Language.

VM Virtual Machine.

4 Introduction

Eye tracking is an evolving and challenging problematic. Used in many applications such as ophthalmology treatment, visual attention research or even in marketing, this technology covers a lot of applications with different needs.

According to the application, it could be desired to know where the subject is looking at a given time, how fast the eye moves, the amplitude of the moves or even their type or trajectory.

This diploma work takes place into a context where it is more important to know when an eye is moving instead of how and where, reactivity is a key feature.

To do so, a high-speed camera films an eye and a system receives this video flux.

Therefore, a real-time video processing task will be implemented in a Zynq SoC.
The high-speed induces a short time to do the computations and so on to know if the eye is moving the system focus on a special area.

To see if any movement occurs, it tracks the limits between sclera and iris, see figure 1.

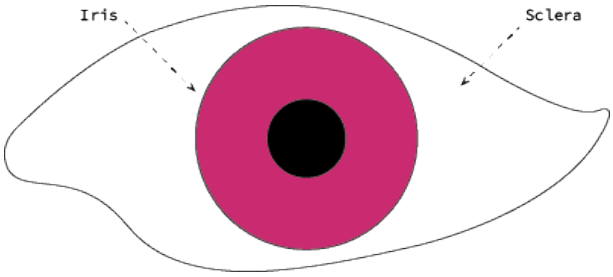


Figure 1: Simplified front view of an eye. By tracking the limit between the sclera (white part of an eye) and the iris (coloured part), movements could be spotted.

There are many methods to track the movements of an eye, one of them is to look at the limit between sclera and iris. This technique presents several advantages as it is focussed on the contrast created by the sclera and the iris (usually much darker [17][18]). Its weakness resides in the presence of the eyelid which could potentially hide a border region when the subject blinks. Methods which target the pupil only and its size are less precise due to the contraction and reactivity of the pupil to luminosity.

One of the main challenges of this work resides in the speed limit allowed by the system. The flux is given in a fast VGA and the operations units must be done in real-time. Hence, the time allowed for processing is limited.

Part I

Objectives and analysis

This diploma work was defined by the following goals:

1. Put together a Zynq design environment
2. Develop a system where the PS must interact the PL
3. Develop a more complex component implementing an image processing task in real-time
4. Investigate the possibilities of DMA for the image transfers

An initial analysis was done to divide the work in several main assignment and get an insight of the main tasks for each objective. This work is described in detail in appendix B.

First step was to get familiar with Zynq architecture by testing some basic tasks, a communication had been then implemented between the PS and the PL.

Then a flux was given to the PL from the PS and a few operations such as noticing luminosity changes were done.

5 Decomposition

5.1 Design environment

The first objective was to define a design environment and began by defining the development board.
During the thesis introduction meeting several boards were suggested and a choice has to be made.

It has been decided that the work would be done with the help of a VM. Working within a VM allocated an appropriate portability of the project as well a light working environment.

5.1.1 Environment setup

The evironment had been be composed mainly as following:

- based on Ubuntu
- Vivado to work with Xilinx FPGA
- HDL Designer for VHDL development
- Atom, git and such useful tools

The exact list of tools will be defined on setup and can be prone to evolution.

5.2 Basic operations

Basic operations were decomposed in a few milestones:

1. Being able to use the whole environment
2. Setup a simple communication between the PS and the PL

5.3 Real time image processing

A test bench had been be conceived. It was designed to provide at least those functionalities:

1. Creating a video-like data flux
2. Defining and transmitting information to do the tracking

The video flux is a VGA on 10-bits greyscale transmitted on 8 16-bits data lines.

To reach a real-time video processing, first development had been deployed on a small scale. Meaning, only one data line and a reduced image size has been used.
Then a second iteration will do the job on all data lines, which means 8 in parallel.

Working in real-time here required to do all the needed operations in a limited time-frame, the one created between two images. This time is defined by the video framerate.

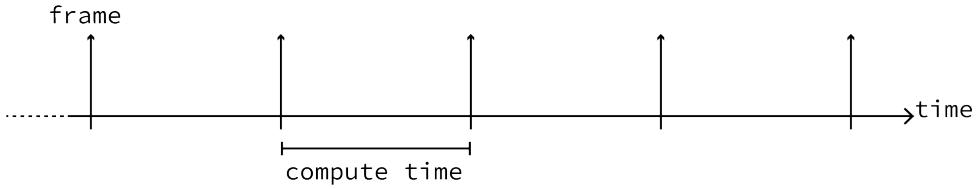


Figure 2: Each frame processing must be done within the time between two frames.

5.4 Specifications

As said, movements were tracked by looking at the limits between the iris and the sclera. It was accomplished by defining groups of pixels on the said border.
Visuals of the principle are listed in the subsection 5.4.1.

5.4.1 Sample image

The figure 11 gives a visual indication of how the border of the iris and the sclera was defined in the system.
The iris was simplified into a full black circle and the sclera was reduced to the white area surrounding it.

To detect movements, the median value of groups of pixels near the border was tracked. Therefore, when one median value tends to move away of the black value, it indicated that the target was moving.
To be sure that a movement was actually occurring, a minimum of median values must be out of the defined bounds.

Secondary, to be sure that the pixels groups inside in the dark area are valid, a few other pixels groups are placed outside the target and act as control.
By placing those pixels groups in a light area, it became possible to assess if problems occured (invalid video flux for example).

Figure 3 introduce the concept of ROI with the squares. It defines an interesting group of pixels. More information can be found in the subsection 5.4.2.

5.4.2 ROI

On this video flux, up to 32 ROI could be placed. One ROI is defined as a square of 3 to 8 pixels wide (see figure 4), a flag assigned to the ROI that would indicates its mode (inverted or not, explained below) and a X/Y position within the VGA frame. A ROI in its normal operation works in NORMAL MODE. It meant that the detection was based on a dark base deviating towards clear tones. The INVERTED MODE was defined as its exact opponent. The ROI was triggered by a deviation from clear to dark tones. The application of both modes is given in figure 11.

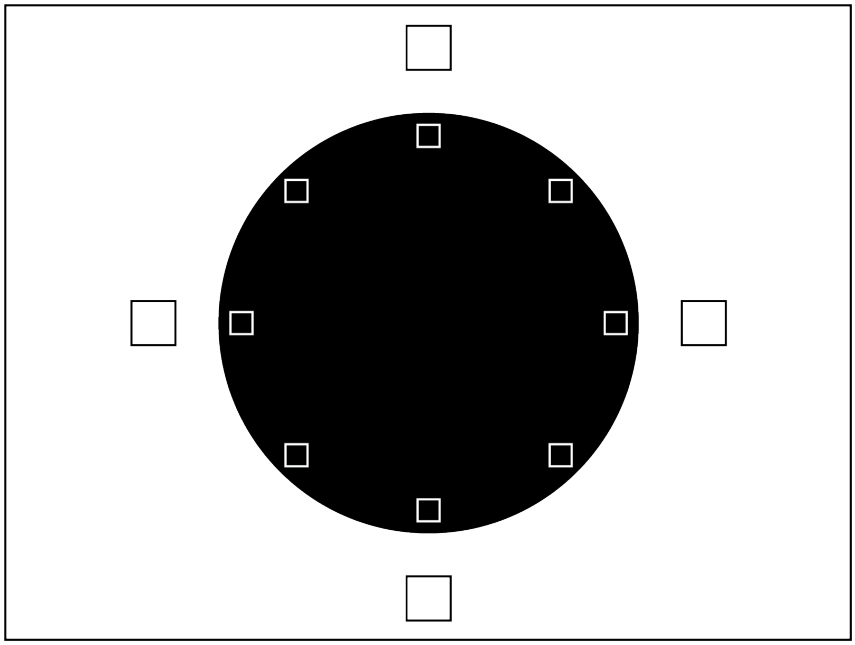


Figure 3: The final image processing was based on the following of several pixel groups. It intended to track the movements of the circular target by looking at the median luminosity of specific pixels groups, called ROI.

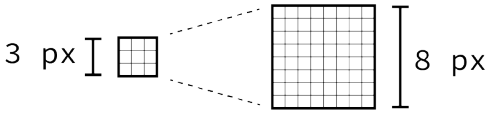


Figure 4: One ROI is a square between 3 to 8 pixels wide.

Attention must be drawn on the fact that the system must handle a ROI of only 1 pixel even if this is not encountered in real working conditions. This is a tested specification.

5.4.3 What to detect

The figure 5 shows how the movement was detected. A red ROI indicated a trigged one.

ROI are used to track movement when placed inside the dark circle and as a control group when placed outside, in the white area.

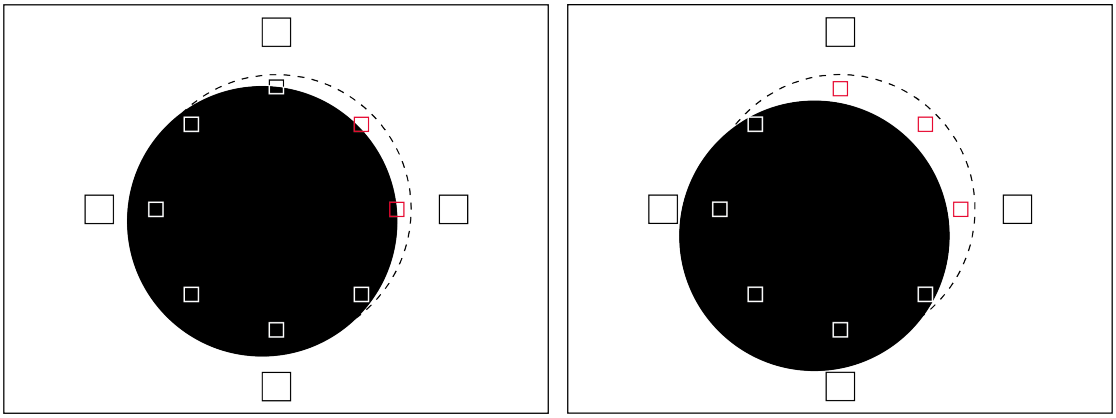
5.5 Principle of detection

An invalid ROI was defined by a too high medians value (or too low if it was in inverted mode). Mathematically, this was translated as the sum of all pixels within the ROI frame, divided by the number of pixels.

5.5.1 Video

The video flux was in VGA format (640/480 pixels), up to 500[FPS], grayscale 8-10 bits (parametrizable but defined on 10 bits for the thesis) on the LSB of a 16 bits value.

Alternatively, video flux could be provided by a standard Raspberry Pi camera which gave 1080 30[FPS], 720 60[FPS] or VGA up to 90[FPS] [15][16].
This version would give more time to work as framerate would be substantially reduced.



(a) When the target would move within the tolerance range, no response is flagged.
A red square indicates a trigged ROI.

(b) If the target crosses the tolerance range, a signal is triggered and a flag is raised.

Figure 5: Example of how the target movements were detected. The centred position of the target is indicated in dotted lines.

6 Further steps

An investigation of images transfers via DMA could lead to a more efficient usage of resources.

7 Planning

A simple planning was extrapolated of the analysis in appendix B.
Preparing the whole system and analysing it was planned on 6 weeks, including hands-on.
Working on basic task was estimated on 3 weeks.
Finally, implementing the real-time operation was estimated to take 6 weeks as it constituted the central task.

In temporal terms, this decomposition looks like the following Gantt-Diagram.

Week number current/in project Time in weeks		20/1	21/2	22/3	23/4	24/5	25/6	26/7	27/8	28/9	29/10	30/11	31/12	32/13	33/14	34/15
Initial analysis	3															
Hands-on	3															
PS-PL interactions	3															
Real-time operations	6															

Table 1: Initial planning.

The initial analysis intented to decompose the work into a handful[1] of main tasks and thinking about how to implement and achieve them.
It also contained the VM setup.

The hands-on one was dedicated to be comfortable with the development environment.

Finally, the main task was divided into two versions. One to work on a reduced and simplified system to outline the problematic and then a second with the real-time operations was implemented.

[1] https://xkcd.com/1070/

Part II

Design environment

The design environment was composed by the hardware and the software and the work done is described in following sections.

8 Hardware

Several boards could be used in this study.
As discussed during the introduction meeting, three boards met the needed requirements:

- FPGARackZynq[1], a board from HEI
- Snickerdoodle[2], a really compact board
- ZynqBerry[4], a board with a Raspberry Pi form-factor

All of them are based on Zynq 7000 family but are available with different types of core.
Similarities stand in a PS composed by one or two Cortex A9 with FPU and a PL equivalent to Artix-7 FPGA family.
More details are available in the Xilinx documentation, see appendix A.

8.1 Board choice

All three are based on a Zynq 7000 serie, therefore, their performances are similar. Slight difference appeared in peripherals and support.
The peripherals were not significant choice criteria as they were not important for this project. Except for the DDR which could be used in a particular use case that was not part of this work.
Having DDR could be useful if the system had to locally store a few images. In theory, the PS was fast enough to get the stream correctly.
If not, the PL must have had a direct access to a dedicated DDR.

The following table compares briefly the characteristics to choose the board.

		RackZynq	SnickerDoodle				ZynqBerry	
			Black	Prime LE	One	no-name		
	SoC	Z-7020-1C	Z-7020-3E	Z-7020-1C	Z-7010-1C		Z-7010	Z-7007s
			Cortex-A9 dual-core					Cortex-A9 single-core
	Frequency [MHz]	667	866	667	667		667	
PS	DRAM	1[GB]	1[GB]	512[MB]	512[MB]	1[GB]	64/128 - 512/512[MB] (Rev1/2/3)	512[MB]
	Type	DDR3L SDRAM (2 x 256[Mb] x 16, 32-bit data bus)	400[MHz] LPDDR2				DDR3L SDRAM	
	SRAM		256[kB]/28.4[Gbps]		256[kB]/36.9[Gbps]			
			Artix-7 FPGA					
	Cells	85K			28K			23K
PL	LUT	53200			17600			14400
	Flip-flops	106400			35200			28800
	Block RAM	4.9[Mb]			2.1[Mb]			1.8[Mb]
Other	Additional CPU		STM32F078 (cortex M0)					

Table 2: Characteristics of the suggested boards. A void cell indicates missing information from the manufacturer.

As no board disposes of separated DDR, the ZynqBerry, mounted with a 7Z007s, was chosen for its direct availability in the school.
Furthermore, this board was easy to handle and work with as it only needed a μUSB cable to power and program it.
Finally, its form-factor was a plus to easily add functionalities such as the official Raspberry camera instead of a dedicated one, direct access to storage via SD-Card-card, HDMI port and other standard as the hardware was designed in this direction.

9 Virtual Machine

To be able to work in a portable and dedicated environment, all the software components were installed in a VM. The chosen environment was composed by the following main components listed in table 3.

Type	Software	Version	Note
OS related	Ubuntu	18.4.3 LTS	
	VirtualBox	6.0.8	
Design	ModelSim	10.7c	
	HDL Designer	5.05-015	
	Vivado	2018.3	
Documentation	Typora	0.9.74	beta
	Atom	1.40.0	
	MediaWiki	1.32.1	
Version control	git	2.17.1	
	GitKraken	6.0.0	

Table 3: List of the VM components and version used.

The complete instructions are given in appendix C.

During the initial meeting it was defined to work on a Linux based system and Ubuntu was chosen because of its popularity.
VirtualBox was chosen because of its popularity and associated documentation was available .

Design tools were defined by school licences and hardware requirements.

Version control and documentation tools (Typora, Atom, MediaWiki) were described as essential and are popular amongst the informatic tools development community. Typora is a Markdown editor which give a nice what you see is what you get paired with an easy to use interface.
Finally MediaWiki was suggested in the initial meeting to host the VM configurations and generics information's.

9.1 Setup

Shortly, VirtualBox was used to host a near naked Ubuntu. Where additional tools could be introduced.
Instructions from Mondzeu Wiki[2] have been followed to install HDL Designer.
Then ModelSim by similar method[3] and finally Vivado.

9.2 Hands-on

To test the setup and get an insight of how the environment works, a simple example provided by Digi-Key[7] was followed. This design applies a PWM with a variable duty-cycle driven by a hardware timer on a GPIO and prints some words through serial port.

The VM did not let the serial transmission pass between the board and the environment. As this was not essential, no further investigations were made in VirtualBox configurations.

The facts that the GPIO was reacting and measured with an oscilloscope was sufficient to validate the development chain.

[2] http://mondzeu.ch/wikis/EDA/index.php?title=Install_HDL_Designer
[3] http://mondzeu.ch/wikis/EDA/index.php?title=Install_Modelsim

Part III

Loopback

To assess the communication between PS and PL, a form of loopback was implemented.
Taking the form of connecting LEDs and buttons to the processor and linking them via the PL before accessing through AXI to their belonging registers.
Figure 8 gives a simplified schematic view. The GPIO IP is configured with two channels each one dedicated to buttons, respectively LEDs.

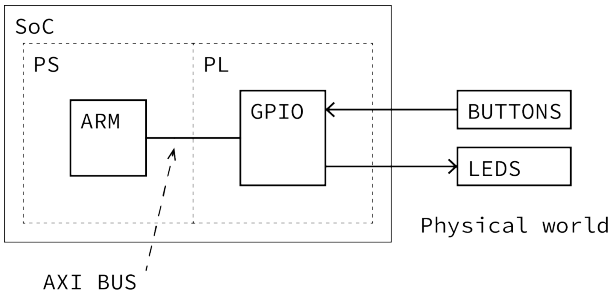


Figure 6: Simplified view of PS and PL interaction. The processor can access to buttons state via the logic and the AXI bus.

10 Development basis

Working with the hardware was facilited by a minimal working example easily editable in Vivado.
Based on the previously followed tutorial (see subsection 9.2) which make a LED fade-in and out with a hardware timer, it has served for the loopback task.

11 Implementation

Added to the basis, a GPIO IP was added from the Xilinx libraries. It was parametrized with two channels, one for LEDs and one for buttons, it allowed the PS to access the physical components through the PL.

The implementation was trivial as it consisted of reading GPIO dedicated to buttons and write the value into the registers of LEDs GPIO.

11.1 Specifications

GPIO This IP was configured with two channels, one for buttons and one for LEDs.
Channel 1 was dedicated to buttons, WIDTH parameter = 4 (one bit per button).
Channel 2 was dedicated to LEDs, WIDTH = 7 (one bit per LEDs but LED 0 was dedicated to the PWM).
Interruption was not used.

LEDs Physically, the connections were done as shown in figure 7. The first LED was driven by the PWM from the hands-on, then the next four LEDs were driven by one button per LED. The three remaining were not

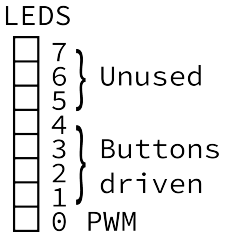


Figure 7: LEDs are driven by PWM, buttons 0-3 and remainings left unused.

used.
The wanted behaviour was button pressed leads to a LED ON then OFF when released.

11.2 Hardware design

To fulfil the desired requirements, the block design shown in figure 8 have been implemented.

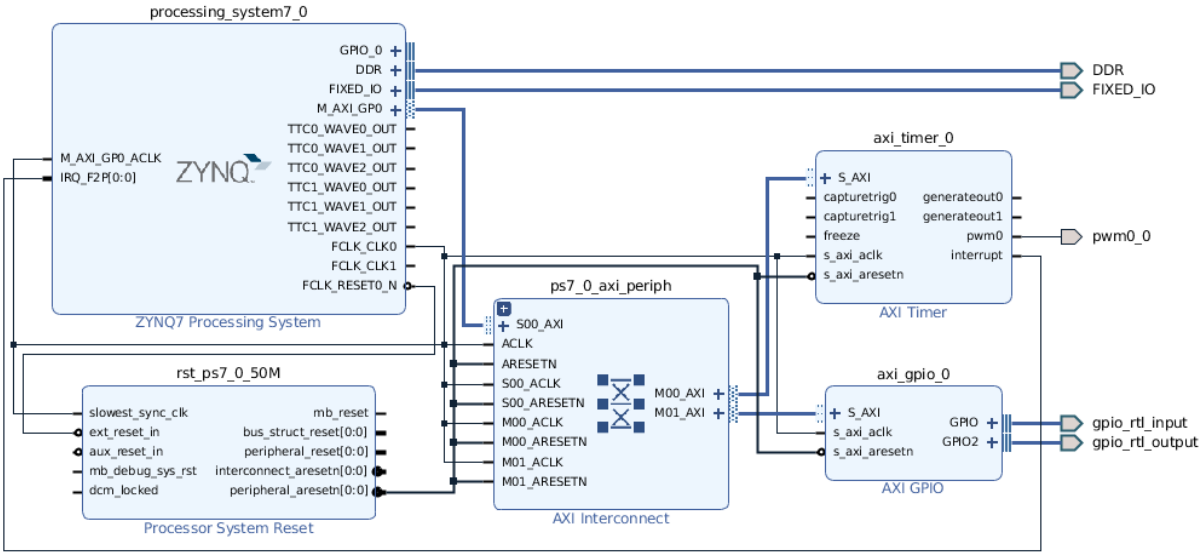


Figure 8: Block design implemented for working with LEDs and buttons. The block named AXI GPIO was the centre of this design by allowing the PS to interact with external hardware through PL. (source: Vivado block design explorer)

Consisting of two families of blocks, this design allowed an interaction of the PS with external hardware through PL.
The software within the ZYNQ 7 PROCESSING SYSTEM block reacted to the buttons push (more information in subsection 11.3).
Blocks ZYNQ 7 PROCESSING SYSTEM and PROCESSOR SYSTEM RESET were mandated to use the processor and AXI INTERCONNECT to use the AXI bus.
The only required action was to configure the PS as needed (and already done as this was a common part with the empty architecture realised previously in the hands-on, see subsection 9.2).

GPIO As explained in subsection 11.1, this IP was configured in dual-channels to read and write buttons, respectively LEDs.

11.3 Software

The software was constantly reading the GPIO buttons's registers to acknowledge when a button is pushed. Then it wrote the buttons status into the register dedicated to LEDs.
The entire code is available in appendix D.

Part IV

Real-time processing

12 Analysis

As explained in the initial analysis (see subsection 5.4), the tracking was based on following the state of ROI.

12.1 Processing goal

As said in the introduction, the objective was to track the movements of an eye, which could be schematized as a black target in a white environment as shown in figure 9.

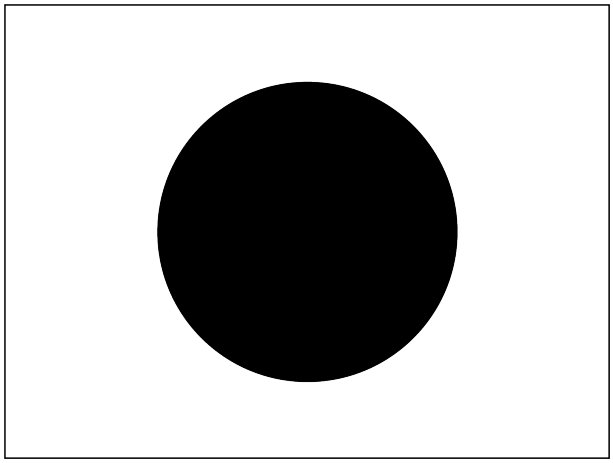


Figure 9: An eye could be seen as a black on white target.

12.2 Application

In reality, an eye is not perfectly black and white. Hence the real coloration of the iris and the sclera can be evaluated on a greyscale.
Factors such as patient himself, exposure time, illumination and angle of view could induce a lighter black (or darker white).
Detection threshold was a parameter proper to each ROI.
As well, ROIs could be placed on a clear environment to detect a decreasing luminosity. In this case, it was working in inverted mode.
Those parameters are shown in figure 10.

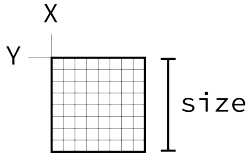


Figure 10: An ROI was composed by a position in X/Y and a size as shown here. A non-represented feature was the threshold to which the median value will be compared to, the mode and an id were the remaining characteristics.

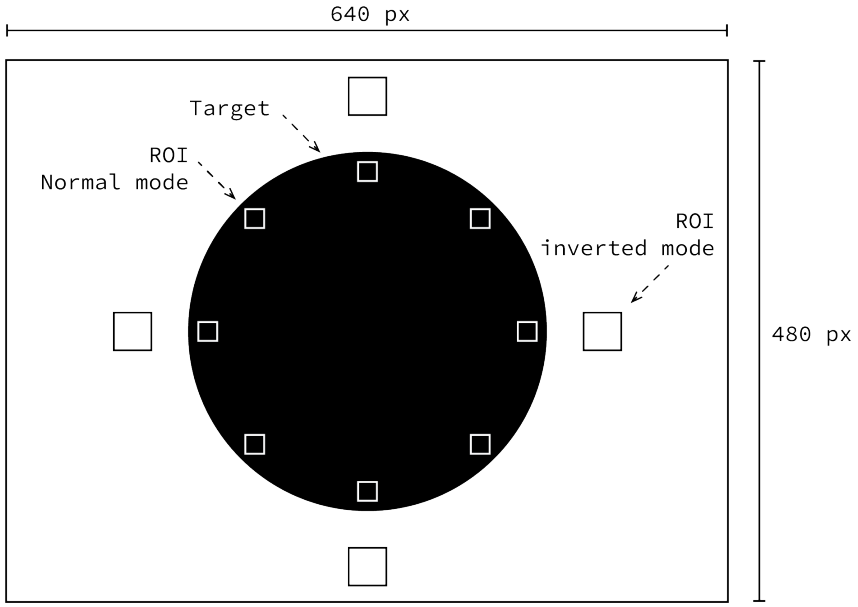


Figure 11: The final image processing was based on the tracking of ROIs. Their placement showed in this figure is an example. In reality, their positionning and number are user defined.

12.3 How to

In facts, the processing of each ROI was done in one dedicated bloc. It means that for n-ROIs, n-blocks were generated in parallel and data were dispatched by a monitor.

12.4 Optimisations

12.4.1 Computations

The calculations to get a ROI value included a division (as the median is defined as the sum of all pixels divided by their quantity).
As divisions were consuming high amount of logic and are inefficient operations in HDL the median was not calculated. Instead, thresholds were given multiplied by the ROI's pixels amount.

13 Processing

One block computes one ROI. Composed by a state-machine and a few processes, its interface is shown in figure 12. It takes ROI parameters and frame data to do the desired computations namely accumulate each pixel value and confronting it to the given threshold.

13.1 Description

The system is based on a state-machine which drives some counters to count pixels. Then a few processes make the effective ROI detection.

Two processes driven by a counter and the frame signals counted the pixels for a third process which took care of checking if the current pixel was in the ROI. If yes, the value was transmitted to an accumulator before being confronted to the threshold to determine validity.

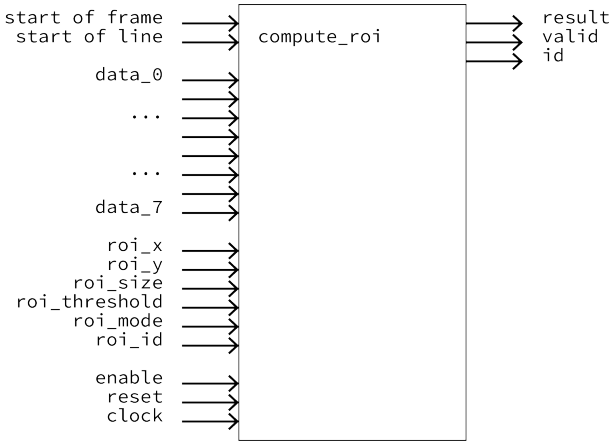


Figure 12: This block takes a ROI and return the according result. It is composed by subblocks which are dedicated to specific tasks.

14 Proof of concept

To validate the global design, a first version which worked on only one data line has been implemented. Using the test bench BASIC PROCESSING (see subsection 17), it was state-machine based.
The top level was the same as in figure 12 unlike there was only one data line.

The code for this version is contained within one file available in appendix D.6.

14.1 State machine

Each time a new frame is received, the computation will begin, and this event launches a state-machine which goes as shown in figure 13.

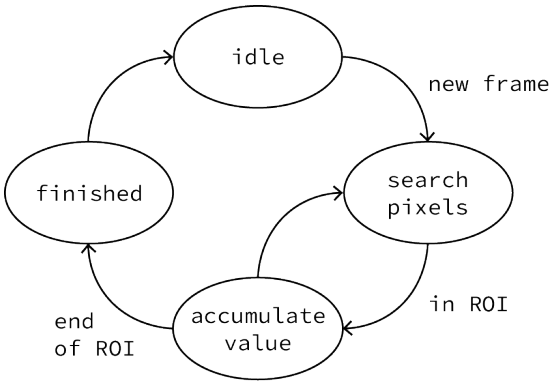


Figure 13: Proof of concept state-machine. A new frame launches the detection and then each time a ROI pixel is found, its value is accumulated. The cycle continue until each pixel has been accumulated.

In a few words, the system behaves as following.
Waiting a new frame is done in IDLE and when a *start of frame* occurs, it goes into SEARCH PIXELS. This state launches the search and wait for an indication of a pixel located in the ROI. Then the value is accumulated, and the cycle repeats itself until the last ROI's pixel.
At that moment, the output is set at the right value, depending of the ROI median value confronted to the threshold and the system is ready for a new frame after being pass through FINISHED.

A deeper explanation on how each state behaves and which process are involved is given in the following paragraphs.

IDLE This state wait for a *start of frame*.

SEARCH PIXEL Begin the search by enabling a main counter and waiting for a flag which indicates that a pixel located in ROI is detected to go in the next state. The search itself is delegated to a combination of processes (see further) driven by the state machine and the main counter.

ACCUMULATE VALUE When entering this state, a flag is raised to enable an accumulator located in a process. If the pixel is the last of the ROI, the state machine will go in the last state, else it will continue the search by returning to the previous state.

FINISHED When the entire ROI is scanned, this stage disables the main counter and indicates the validity of the outputs by rising *valid*.

14.2 Processes

The state machine uses a few processes to perform actions.
Three of them are dedicated to counting bits and pixels, one detects if the current pixel is in a ROI or not, one accumulates values of ROI's pixels and the last one verifies if the ROI is valid or not.

COUNTINPUTFLOW Described as main counter, this synchronous *reset/clock* process counts the incoming bits to synchronize the data flow with the pixel's counters.
Counts on a base of a constant defining the data duration (in clock cycles).
This counter reset its value when not receiving frame, or by a global reset.

COUNTPIXELX Asynchronous process driven by the main counter and the frame signals (*start of frame* and *start of line*) to count the pixels in x-axis.
Increment itself when the main counter reaches its half-period (arbitrary timing, parametrizable).
This counter reset its value when a global reset or a new frame occurs and on every new line.

COUNTPIXELY Similar to the x-axis counter but driven by *start of line* to increment itself.
This counter reset its value when a global reset or a new frame occurs.

DETECTPIXEL Synchronous *reset/clock* process which check if the actual pixel is in the ROI or not. This is done by confronting its coordinates to the counters x/y every time that a new pixel was incoming. This is known by checking changes on the x-axis counter.
When a wanted pixel is detected, its value is stored in a buffer and a flag is raised for the state machine.
It is also detecting when the actual pixel is the last to be buffered, there too by rising a flag intended to the state-machine.

ACCUMULATE_PROC Combinatorial process which accumulates given pixels values in a buffer.
When a new frame occurs, resets itself.

CHECKVALUE Combinatorial process which was triggered when the ROI's last pixel is detected. Confronts the accumulator buffer to the threshold and returns a logic response.

15 VGA processing

The main difference with previous data format occurs in transmission, as VGA format is given through eight parallel lines instead of one.
One frame is transmitted by batch of 8 pixels on 8 data lines, meaning that between two batches, each pixel must be checked to see if it is comprised into a ROI.

This leads to a more complex design which is not anymore hidden in one block but divided to get an easier and clearer view.

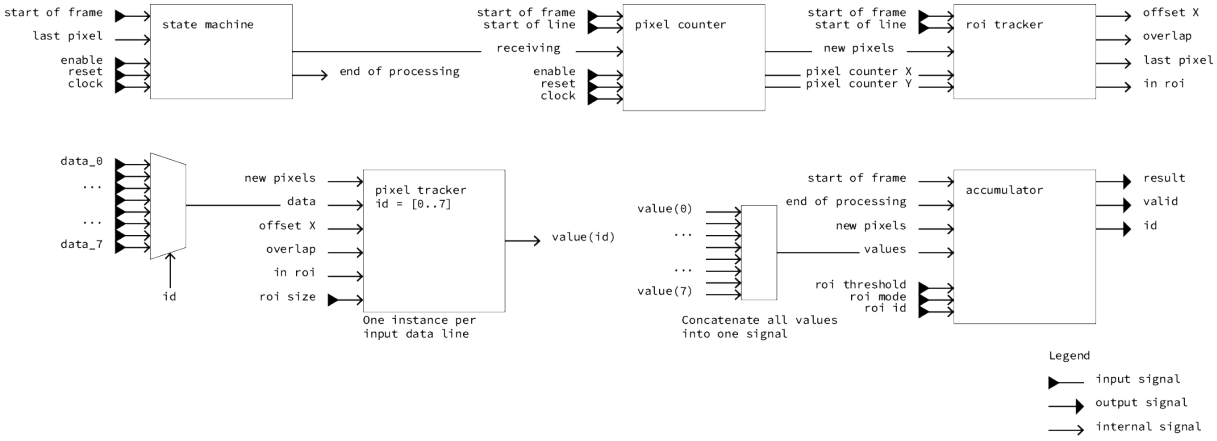


Figure 14: COMPUTE ROI is composed by a state-machine which drives the tracking, some counters to search pixels, several blocks to get the right values and finally an accumulator/comparator to deliver the result.

Figure 14 show the new repartition of tasks into the blocks.
Principle is the same, counting incoming data and extract from this pixel's values and coordinates, driven by a state-machine. Analysing if any of the pixels could be in ROI and then buffer interesting values.
Finally, an accumulator will add-up and check if the value is above or not of the given threshold.

15.1 State machine

In this second iteration, the state machine is reduced in three states as the accumulation is done by a dedicated sub-system.

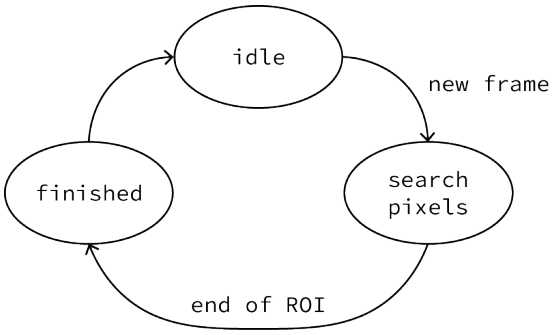


Figure 15: To work with VGA and its 8 input data lines, the state machine has been simplified as some of the work is done in driven subblocks.

The code is available in appendix D.11.

15.2 Pixel counters

To synchronize and receive correctly the incoming pixel flow, a block is dedicated to counting and indicating when values are ready to be read and what they correspond to.
Figure 16 show the chronogram of inputs and output signals.

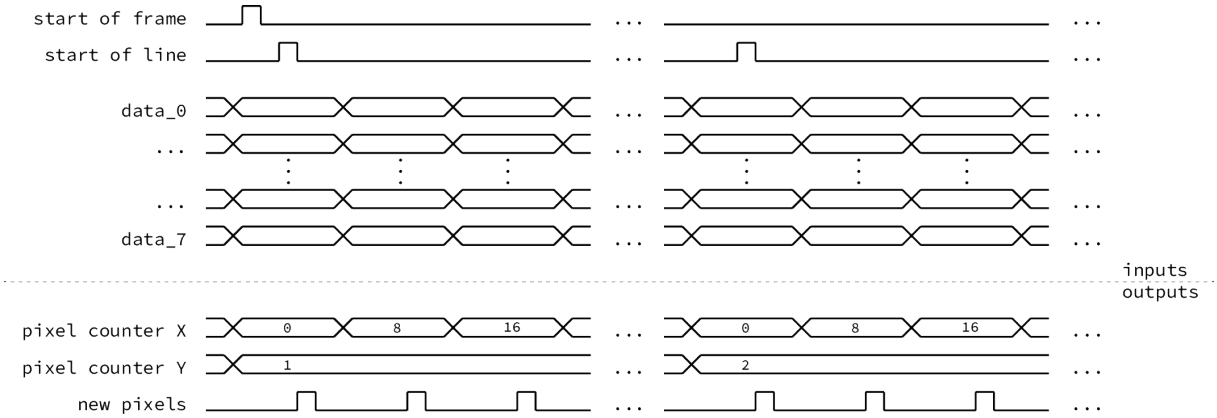


Figure 16: Chronogram of PIXEL COUNTERS. Taking as input data lines and frame signals, it delivers two counters to know where in the frame the incoming pixels are and when they are ready to be read. Clock is not represented as it is much faster than those signals.

On each new batch of pixels, the x-counter increments itself by step of data lines number (in this case 8) and on every *start of line* the y-counter does the same. Due to this sensibility to the signal, this second counter starts on 1 instead 0 as x-counter.
Another way to do could be to increment the counter in y-axis when the x-axis reaches the end of a line.
This has the main disadvantage not to be anymore resynchronized with the incoming flux by *start of line*.

Finally, *new pixels* indicate when a batch is ready to be read in the input buffers to be read by following blocks.

A *start of frame* reset all counters and flag to 0.

The code is available in appendix D.8.

15.3 ROI tracker

This block provides indication of when the incoming pixels contains some of the ones from the ROI to track and when the last ROI pixel is arrived. Those information's are encoded in signals *in ROI* and *last pixel*.

The two other signals (*offset x* and *overlap*) are employed to get the right pixels in each batch.
Figure 17 and 18 show the different situations of how a ROI could be positioned within the received pixels.
Next paragraphs detail the tasks of the implemented processes.

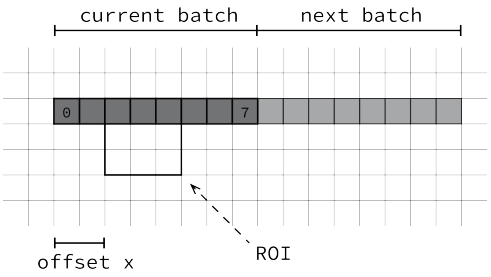


Figure 17: First situation, ROI is fully contained by a batch of pixels for each of its line. In this case, the offset x is defined as the number of the first pixel to match with ROI.

COMPUTEOFFSETS In the case shown in figure 17, each line of the ROI is fully contained by a batch of pixels. It means that an offset in x (and y but this axis is much more intuitive as it increments itself on each new line as soon as a ROI is detected until its end) must be computed to accumulate the right values further in the logic (see subsection 15.4).
Processing those offsets is the task of the process COMPUTEOFFSETS.

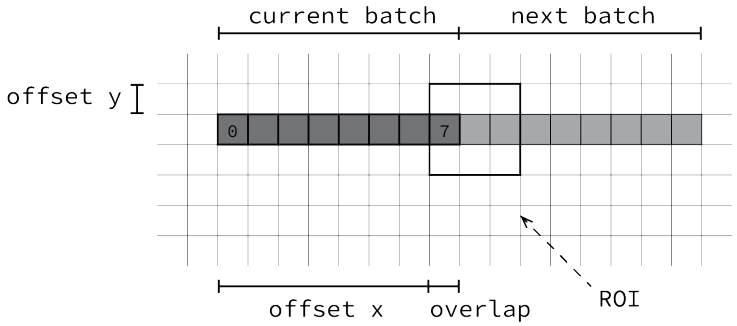


Figure 18: Second situation, ROI is partially over a second batch of pixels. It means that for the current batch all pixels after the offset must be taken into account in accumulator but also a few from the next batch. The remaining amount is given by *overlap* value.

A difference exists between *offset x* and *offset y*. The first one is used by other blocks (see subsection 15.4) but *offset y* is only an internal signal used by the ROI tracker processes.

DETECTOVERLAP The second case shown in figure 18, the current batch in now on the second ROI line and overlapping over two batches.
At this moment, the process DETECTOVERLAP is useful to indicate that an overlap is occuring and gives its size. To be exact, a signal *overlap* is set to the right value if this situation occurs, else it stays at 0.
Again, those information's are used further in the logic in the pixel tracker.

DETECTROI and DETECTROIEND The two remaining processes are pretty similar. They intend to detect the first, respectively the last, pixel of the ROI.

The code is available in appendix D.10.

15.4 Pixel Tracker

This block in generated once for each data line and is dedicated to check if its specific pixel value is to transmit to the accumulator. Figure 19 shows a simplified schematized view of how each pixel is read by the trackers. The *enable* signals comes from a trivial logic which use *offset x*, *inRoi* and *overlap* signals from ROI TRACKER.

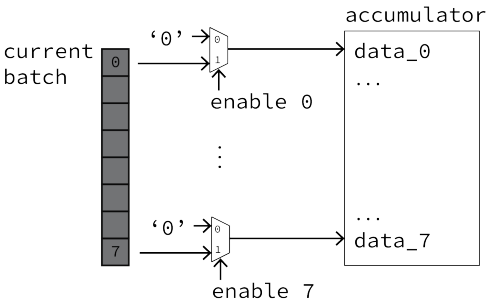


Figure 19: The logic functionality of the pixel tracker is a logic switch which transmits either the pixel value or a 0 if it is not a ROI.

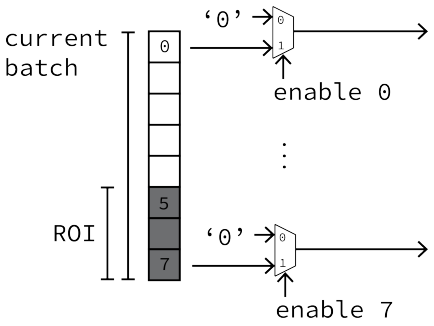


Figure 20: In this situation, the current batch contains three pixels in a ROI, 5 to 7 shown in dark grey. It means that the *enable* from trackers 5 to 7 are activated.

Figure 20 show an example of how the trackers transmit or not the pixels values to the accumulator.

The code is available in appendix D.9.

15.5 Accumulator

The accumulator is in charge of taking the buffered pixels values and sums them up into another buffer.
The maximal value that could occurs is defined as in equation 1.

$$accumulator_{maximal} = ROI^2_{size} \cdot pixel_{bitdepth} = 8^2 \cdot 2^{16} = 2^{22} \rightarrow 22bits \tag{1}$$

Practically, the pixel value is encoded on 10 bits which are then stored in 16 bits values. Hence, it limits the maximal value to 16 bits as shown in equation 2.

$$accumulator_{maximal} = ROI^2_{size} \cdot pixel_{bitdepth} = 8^2 \cdot 2^{10} = 2^{16} \rightarrow 16bits \tag{2}$$

Based on a FOR LOOP architecture which generates an accumulation of each input buffer in a variable, it will, after those operations, confronts the accumulated value to the given threshold.
Comparison is quite simple, either the ROI is in normal or reversed and hence, the final result is a binary response.

ACCUMULATOR This process accumulates the value of its input data line each time a new batch of pixels is ready. The previous blocks (pixel trackers), transmit to the accumulator a 0 by default or the pixel value if it is a pixel from a ROI.
Hence, this process is trivial as it could always accumulate all its inputs. In facts, it is some expanded logic with a FOR LOOP.

CHECKTHRESHOLD When an *end of processing* is thrown, the process compares the accumulated value to the given threshold. ROI mode (inverted or normal) are also considerated.

The code is available in appendix D.7.

16 Libraries

All codes used in this work originating from the libraries defined by LIBRARY IEEE, especially STD_LOGIC_1164 and NUMERIC_STD.
In parallel, a package was written to define some constants and data types. The package is available in appendix D.12.

Part V

Test benches

Three tests benches were conceived. One for basic processing, one for final tests and one to unit test the accumulator.

17 Basic processing

A simplified test bench is implemented to perform a proof of concept.
It generates a picture on only one data line, feeding a single ROI tracker and have been used it to validate a proof of concept.

17.1 Objectives

This test bench was designed to fulfill these objectives:

- Generating some data to simulate a frame and feeding of the tested block
- Be comfortable with the VGA signals
- Creating a base for more complicated test benches

17.2 Implementation

It was based on a state machine which drives a counter to generate data in the form of a greyscale.

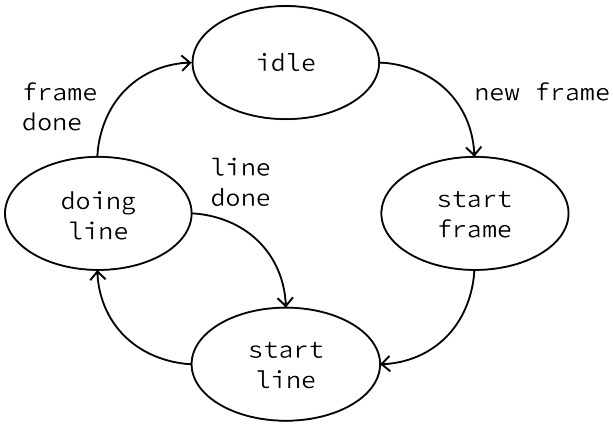


Figure 21: Frame generation is driven by a state machine which draws lines until reaching the desired amount.

GENERATEDATA The data are generated by a counter and boundaries are defined by generics, as well as the step between two pixels.
Practically, the greyscale is a 10 bits ramp. The pixels are counted line by line and a second counter stops the generation when their requested number is reached.

In parallel of this, a few parameters and signals are generated such as ROI attributes (position in x/y, threshold and mode) and frame signals (*start of frame* and *start of line*) as shown in figure 22.

Written code is available in appendix D.2.

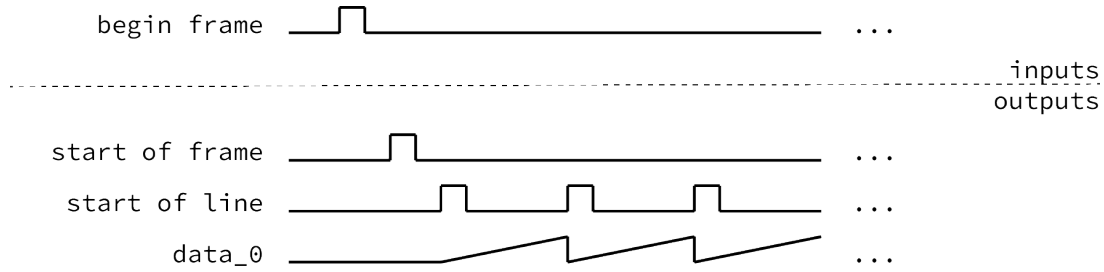


Figure 22: The basic test bench generates a 10 bits ramp over one data line, the frame signals and set ROI parameters (not shown).

18 VGA processing

18.1 Data generation for global test

To validate this version, the developed test bench must be reworked a bit to concord with the following changes:

- Data are splitted on 8 parallels lines
- Each data is a pixel, meaning that each time a new batch of pixels comes, 8 new pixels are coming in series with the last ones

Main changes where done on the way of generating data. Counters counts 8 by 8 (whether the number of data lines) and each line is incremented to match with its relative position in the batch.
Figure 23 show the 8 data lines of this test bench iteration.

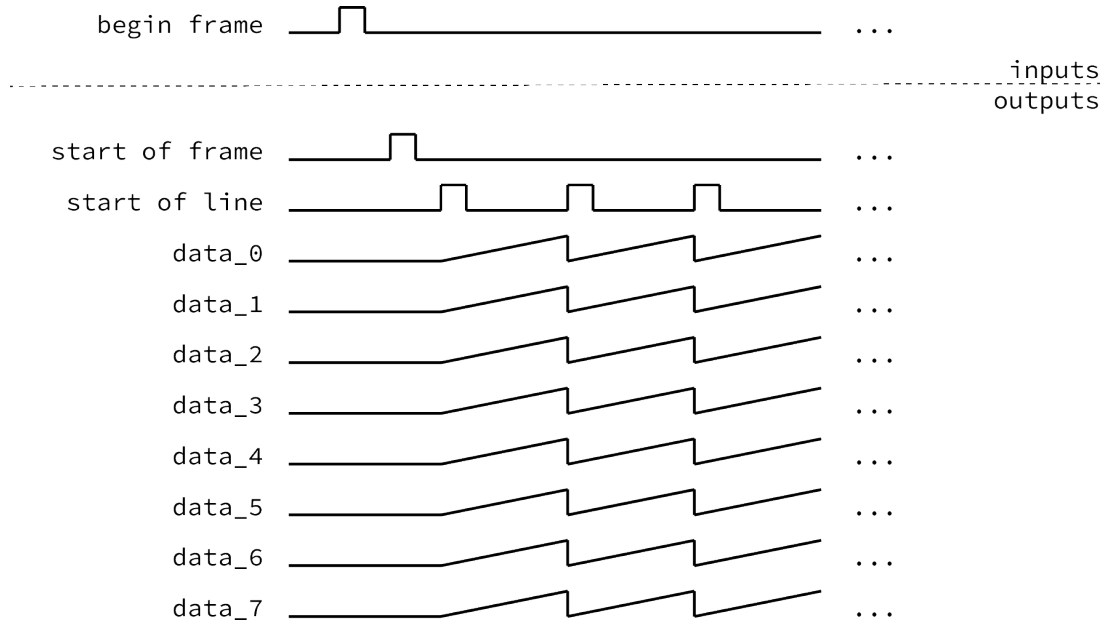


Figure 23: The test bench for VGA format generates a 10 bits ramps for each 8 data lines, the frame signals and set ROI parameters (not shown).

The code written is available in appendix D.3.

18.2 Unit tests

Not all blocks are individually tested as it takes a lot of time to conceive the tests or it is not useful as their behaviour is trivial and thus verified in global tests.
Ideally, it would be better to tests each block individually but as time was a constraint, choice was made not to implement the tests.

18.2.1 Accumulator

To make tests easier and faster, the accumulator was tested independently with a dedicated test bench.
The criteria needed to declared a test as successful are described below:

- transmitting a given ID
- comparing the accumulated value with the threshold
- and telling when the result is valid

Transmitting a given ID is a copy of ROI ID to the output. It is later used to sort all the individual results from each compute ROI blocks.

Testing this block alone asks a certain amount of tests given by formula 3 which must be summed with following behaviours:

- when a *start of frame* occurs, accumulator must reset
- an *end of processing* leads to confronting the accumulator value against the threshold and so on an indication on output by a binary *result*. A *valid* tell when the result is ready.

Those two behaviours can be tested in only one situation.

$$base = mode \cdot threshold = 2 \cdot 3 = 6 \tag{3}$$

Thus, six tests would be performed to validate each case and one more to check both behaviours.

Implementation Consisting of seven cases covering each possibility, the pattern is simple and repetitive.
Parameters have been set accordingly to the test case and a loop drive the accumulation. When it ends, the *end of processing* signal is thrown to test the last accumulator stage, the check against the threshold.
The code is available in appendix D.4.

Part VI

Tests

To validate the system, several tests were performed using the test benches described in part V.

Testing each possibility involves a lot of individuals tests. As a reminder, the main tested characteristics were the following:

- various sizes, from 1 to 8
- both modes, normal and inverted
- median value above, under and equal to the threshold
- position near frame borders
- overlap of ROI on multiple incoming pixels batches

Modifying the lines in the listing 1 allows to test the system with the given parameters presented in the abstract bellow.

```
-- set parameters
roi_x <= to_unsigned(632, roi_x'length);
roi_y <= to_unsigned(471, roi_y'length);
roi_mode <= '0';
roi_size <= to_unsigned(8, roi_size'length);
roi_threshold <= to_unsigned(1000, roi_threshold'length);
```

Listing 1: The five lines to edit to test multiple ROI positions. Value are an arbitrary example.

The amount of combinations is given by formula 4. Each label encodes the number of possibilities.

$$combinations = size \cdot mode \cdot threshold \cdot overlap \cdot position = 8 \cdot 2 \cdot 3 \cdot 2 \cdot 4 = 384 \tag{4}$$

As said in subsection 18.2.1, a unit test performed on the accumulator reduces greatly the amount of tests to do, as shown in formula 5 it eliminate cases.

$$combinations = size \cdot overlap \cdot position = 8 \cdot 2 \cdot 4 = 64 \tag{5}$$

Then, only a certain amount of theses combinations must be performed. Remaining tasks would be to test ROI properties (size, position and overlap) but many are redundant.
By example, testing the position near border is in no way related to the ROI size. Indeed, the border could lead to strange behaviour if a ROI is exactly placed on the last pixel, but this appears independantly of the ROI size.
The real amount of test is brought down to 20 (see formula 6).

$$combinations = size \cdot overlap + position = 8 \cdot 2 + 4 = 20 \tag{6}$$

19 Results

For readability reasons, the whole results table is available in appendix D.5.
All tests have been successfully passed. Overall, the results show that with different initial conditions, the system reacted positively and in an expected manner.

Part VII

Further work and improvements

20 Remaining tasks

20.1 Implementation

The whole system must be converted and integrated into the Zynq to be fed by a real video flux (whether it is originating from a camera or computer).
Due to a discussion with Mr. Corthay it has been decided that integrating the whole system into the Zynq hardware was not necessary as it does not show a real interest in this work.

21 System improvements

21.1 Accumulator

The accumulator is a simple FOR LOOP over the incoming pixel batch leading to a cascade of accumulator as shown in figure 24.
The problem is that the path to get to the final result is as long as the number of values to add (complexity O(n) where n is the number of values to sum).

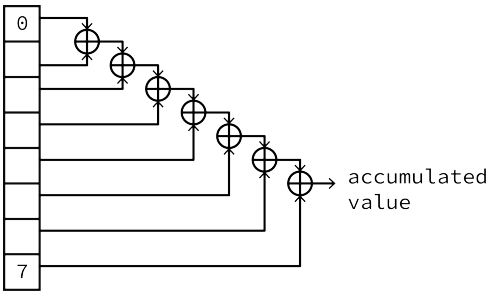


Figure 24: A cascade accumulator work with a serie of additionner.

Balanced tree A possible improvement could be an implementation of a balanced tree.
It is interesting as it reduces the number of additions to $\log(n)$ where n is the number of value to sum.
Figure 25 show its architecture.

21.2 Test bench

The implementation of the tests benches was a bit rudimentary and showed little flexibility.
A real improvement could be to use the current state-machine based system and drive it with a few parameters such as:

new frame requests a new frame generation, could be paired with a few parameters such as image type (white/black, pattern (greyscale, noise))

roi set ROI parameters

Then an automatized result validation could be applied to test many combinations without a manual check.

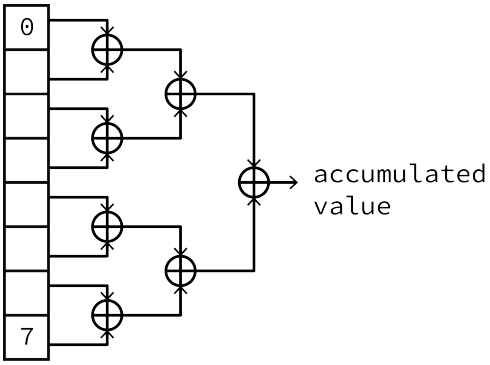


Figure 25: A balanced tree reduce the length of the path needed to do the full operation and bring parallelisation to the operations.

22 Others improvements

22.1 ROI

By improving the system, one should be able to process rectangle ROI. This could allow a fine placement (and less pixels to monitor) of ROI along the border who could be seen as near linear as shown in figure 26.

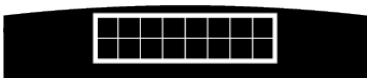


Figure 26: A rectangle ROI to get a better fit on the edge.

22.2 Using DMA

To make faster data transfer without overloading the system, the DMA could be used to transfer frames between the tracking system and the processor in the case where the processor needs to receive a video flux.
This consideration should be taken into account when the current system will be integrated into a higher-level architecture.

Part VIII

Conclusion

As described in part I, this diploma work was defined by the following goals:

1. Put together a Zynq design environment
2. Develop a system where the PS can stream data to a component in the PL which does a very simple task and stream back to the PS
3. Develop a more complex component implementing an image processing task in real-time
4. Investigate the possibilities of DMA for the image transfers

Globally, the defined goals and their evolution were reached.

Design environment The design environment is functional, ready to use and portable via its VM.

Stream from PS to PL and back Streaming data between PL and PS has been modified into an interaction between both parts.
A user interface taking form of LEDs and buttons is connected to the PL and the PS reacts to this interface.
Hence, the interaction is located in the register read/write operations.

Real time image processing The system is capable of receiving a video flux and analysing the comportment of a given ROI.

Further work The real-time operations are fully functional in simulation but needs to be synthetized and tested on hardware.
As setting up a whole system is a time consuming task, it has been decided not to implement this feature in this work. See subsection 20.1 to get more details.

A better implementation of the accumulator could give better performance as it shortens the longest path, see subsection 21.1 for more information.

Designing and implementing a higher-level architecture to handle more ROI per frame by generating the needed amount of computing blocks is the next main step.

Finally, implementing a bunch of automatized test benches must be done to validate faster the improvements done.

Loic FRACHEBOUD

Sion, le 26.08.2019

Part IX

Bibliography

[1] *Hardware/FPGARackZynqADDAV1 - UIT*. URL: http://wiki.hevs.ch/uit/index.php5/Hardware/FPGARackZynqADDAV1 (visited on 05/13/2019).

[2] *snickerdoodle | krtkl*. URL: https://krtkl.com/snickerdoodle/ (visited on 05/13/2019).

[3] Jann Kumann. *TE0720 TRM - Public Docs - Trenz Electronic Wiki*. Sept. 2018. URL: https://wiki.trenz-electronic.de/display/PD/TE0720+TRM (visited on 05/16/2019).

[4] Jann Kumann. *TE0726 - ZynqBerry - Public Docs - Trenz Electronic Wiki*. Wiki. Sept. 2018. URL: https://wiki.trenz-electronic.de/display/PD/TE0726+-+ZynqBerry (visited on 05/13/2019).

[5] Shahul Akthar. *Block RAM and Distributed RAM in Xilinx FPGA*. en-US. Oct. 2014. URL: https://allaboutfpga.com/block-ram-and-distributed-ram-in-xilinx-fpga/ (visited on 05/16/2019).

[6] KRTKL. *krtkl wiki - boards specifications*. Sept. 2018. URL: https://wiki.krtkl.com/index.php?title=Main_Page (visited on 05/16/2019).

[7] *Getting Started with the ZynqBerry - Motley Electronic Topics - eewiki*. URL: https://www.digikey.com/eewiki/display/Motley/Getting+Started+with+the+ZynqBerry#GettingStartedwiththeZynqBerry-HardwareDesign (visited on 05/21/2019).

[8] *Pre-Harvest: Getting Started with the Zynqberry in Vivado 2018.2*. en-US. URL: https://www.knitronics.com/the-zynqberry-patch/getting-started-with-the-zynqberry-in-vivado-2018-2 (visited on 05/24/2019).

[9] Altera. "Understanding Metastability in FPGAs". en. In: (), p. 6.

[10] Eduardo Sanchez. *VGA video signal generation*. en. URL: http://lslwww.epfl.ch/pages/teaching/cours_lsl/ca_es/VGA.pdf (visited on 07/19/2019).

[11] *Vivado Design Suite User Guide: Using Tcl Scripting (UG894)*. en. 2018. (Visited on 08/07/2019).

[12] "Graphical Editors User Manual". en. In: (2008), p. 482.

[13] "HDL Designer Series User Manual". en. In: (2008), p. 602.

[14] Peter J Ashenden. *Designer's Guide to VHDL*. eng. Morgan Kaufmann, 2008. ISBN: 0-12-088785-1.

[15] *Camera Module - Raspberry Pi Documentation*. URL: https://www.raspberrypi.org/documentation/hardware/camera/ (visited on 08/20/2019).

[16] *IMX219PQ | Sony Semiconductor Solutions*. en. URL: https://www.sony-semicon.co.jp/new_pro/april_2014/imx219_e.html (visited on 08/20/2019).

[17] *Eye color*. en. Page Version ID: 910825107. Aug. 2019. URL: https://en.wikipedia.org/w/index.php?title=Eye_color&oldid=910825107 (visited on 08/22/2019).

[18] *The World's Population By Eye Color Percentages*. en. URL: https://www.worldatlas.com/articles/which-eye-color-is-the-most-common-in-the-world.html (visited on 08/22/2019).

Part X

Appendix

A Datasheet

To keep a light bibliography, used datasheet are listed here.
Xilinx provide the following documentation to use their environment:

- Zynq-7000 SoC Data Sheet: Overview, DS190 (v1.11.1) July 2, 2018
- Zynq-7000 All Programmable SoC Software Developers Guide, UG821 (v12.0) September 30, 2015
- The Zynq Book, 1st Edition, produced in association with Xilinx by the University of Strathclyde, Glasgow

Others are available at `http://www.xilinx.com/products/silicon-devices/soc/zynq-7000.html#documentation`

B Specifications and initial analysis

A document generated at the project genesis to analyse the given specifications and how the project is dissolved in various tasks.

Hes·SO // VALAIS WALLIS

School of Engineering

Video processing on SoC

Bachelor thesis

SPECIFICATIONS AND INITIAL ANALYSIS

Abstract

The aim of the diploma work is to develop the base for an image processing task on a Xilinx Zynq SoC. In this system, the ARM Cortex-A9 based Processing System (PS) will stream images to the Programmable Logic (PL) which will perform an image processing task on the fly.

Student : Loic FRACHEBOUD

Teacher : François CORTHAY

Expert : Pascal FAURE

From : HES-SO Valais-Wallis

V2.0

13.06.2019

Contents

Introduction

This document intend to detail requirements of bachelor thesis. It is a compenent of the main report and does not constitute a whole independant document itself.
Please refer to the report for any missing information.

Objectives

Basis goals are defined by thesis description:

1. Put together a Zynq design environment
2. Develop a system where the PS can stream data to a component in the PL wich does a very simple task and stream back to the PS
3. Develop a more complex component implementing an image processing task in real-time
4. Investigate the possibilities of Direct Memory Access (DMA) for the image transfers

From this each goal is decomposed in a few tasks to get an idea of how the project will take place.

Decomposition

1 Design environment

The first objective is to define a design environment and it begin by defining the development board.
During the thesis introduction meeting three boards were suggested:

- FPGARackZynq[1], a board from HEVS developped by Charles Papon
- Snickerdoodle[2], a really compact board
- ZynqBerry[3], a board with a Raspberry Pi form-factor

All three are based on a Zynq 7000 serie, placing them on a similar performance basis.
Differences appears in peripherals and support but nothing really big.

After a discussion with Mr. Faure, the ZynqBerry with a 7Z007s available at school will be sufficient.

1.1 Environment setup

The work will be done within a VM. It allocate a nice portability of the project and a light working environment in term of just having what is needed.

It will be composed mainly as following:

1. Ubuntu
2. Vivado
3. HDL Designer
4. Atom, git and such useful tools

The exact list of tools will be defined on setup and especially while working with it.

[1] http://wiki.hevs.ch/uit/index.php5/Hardware/FPGARackZynqADDAV1
[2] https://krtkl.com/snickerdoodle/
[3] https://wiki.trenz-electronic.de/display/PD/TE0726+-+ZynqBerry

2 Basic processing

This will be decomposed in a few milestones:

A. Setup a simple stream from the PS to the PL

B. In the PL, forward the stream back to the PS

C. Do a simple operation on the stream before sending back

2.1 System architecture

The first step takes as input a top level design from Mr. Faure wich gives a few IP to fastly get a working design as shown in figure 1.
This define an architecture where the PS stream a data flux to the PL and require to specificate how and in wich format the data is transmitted.

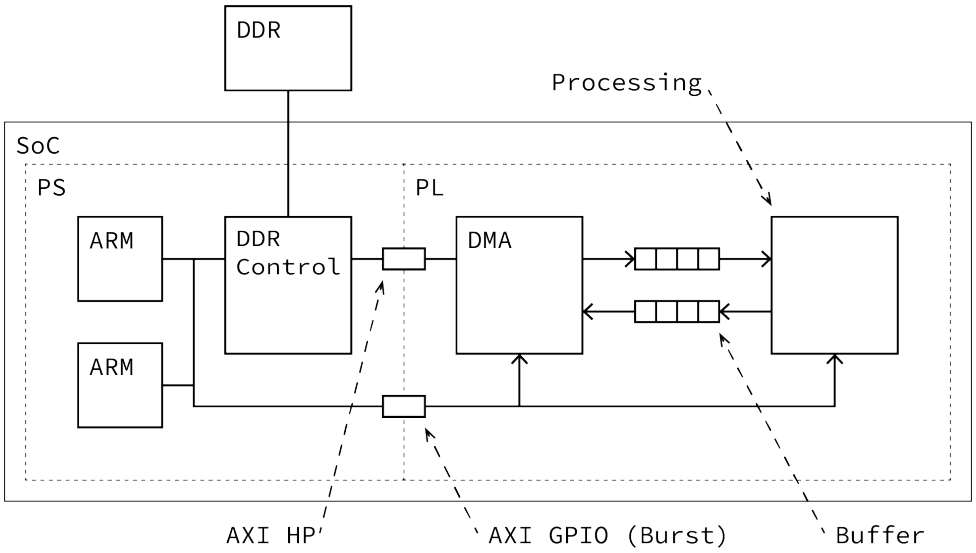


Figure 1: System architecture, the block named *Processing* contain more IPs and especially the one in which my system will take place.

Then from the PL point of view, the stream must be recovered and forwarded back to the ARM. This validate the communication between PS and PL.

Finally, a simple operation such as bit inversion will be performed on the stream.

2.2 Feedback flux

To give a feedback, a slower flux must be given back to the PS.
This could consitute a nice work to do at this moment because the data flows are proved working with really simple operation.
Generating this feedback consist in reducing the framerate on a lower value.
This value could be a fixed parameter or dynamically processed from the original framerate to obtain the wanted one.

3 Real time image processing

Later a more complex operation on the stream will be performed.

A. Start by a simple operation in real time

B. Develop a more complex processing to perform image processing

Working in real time with image processing is the main objective.
To reach it, real time operations must be implemented. At this point, odds are high that the simple operation implemented previously is working in real-time.

Finally an image processing operation will be conceived to be applied on the stream.
Working in real-time here ask to do all needed operations in a limited time-frame, the one between two images.

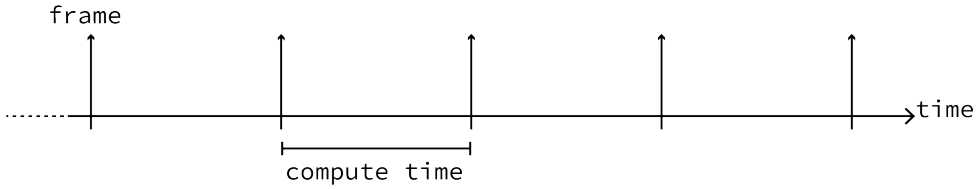


Figure 2: Each frame processing must be done within the time between two frames.

3.1 Specifications

Video The video flux is in VGA format (640/480 pixels), up to 500[fps], grayscale 8-10 bits on the LSB of a 16 bits value.
Some informations are available in registers, others are given as hard parameters. This will be more preciselly defined when the architecture will be in my hands.

ROI On this flux, up to 32 ROI could be placed. One ROI is defined as a square of 3 to 8 pixels wide (see figure 3), a flag who indicates its mode (inverted or not, explained below) and a X/Y position inside the VGA frame. A ROI in it's normal operation works in *normal mode*. It means that the detection is based on a dark base deviating towards clear tones. The *inverted mode* is it's exact opponent. The ROI is trigged by a deviation from clear to dark tones. The application of both modes is given below.

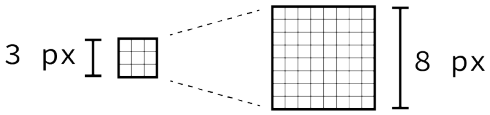


Figure 3: One ROI is a square between 3 to 8 pixels wide.

Later it could be nice to improve the system to be able to process rectangle ROI. This to allow a fine placement (and less pixels to monitor) of ROI along the border who could be seen as near linear as shown in figure 4.

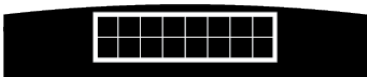


Figure 4: A rectangle ROI to get a better fit on the edge.

Be aware that the system must handle a ROI of only 1 pixel even if this is not encounter in working conditions. This is a tested specification.

Sample image The figure 5 gives a visual indication of how it looks like.
In the center, the circle is the target to follow. If it moves, this has to be indicated. The circle is really dark, close to full black.
To detect movements, ROI are placed inside the target along the border. So when one ROI median value tends to move away of black, it indicates that the target is moving.
Validation of a movement needs a minimum amount of ROI trigged, given as a parameter.

Secondary, to be sure that the ROIs in the dark are valids, a few inverted ones are placed outside the target. As this area is well lit, if they are not bright it means that a problem occurs (invalid video flux for example) and it must be indicated as well.

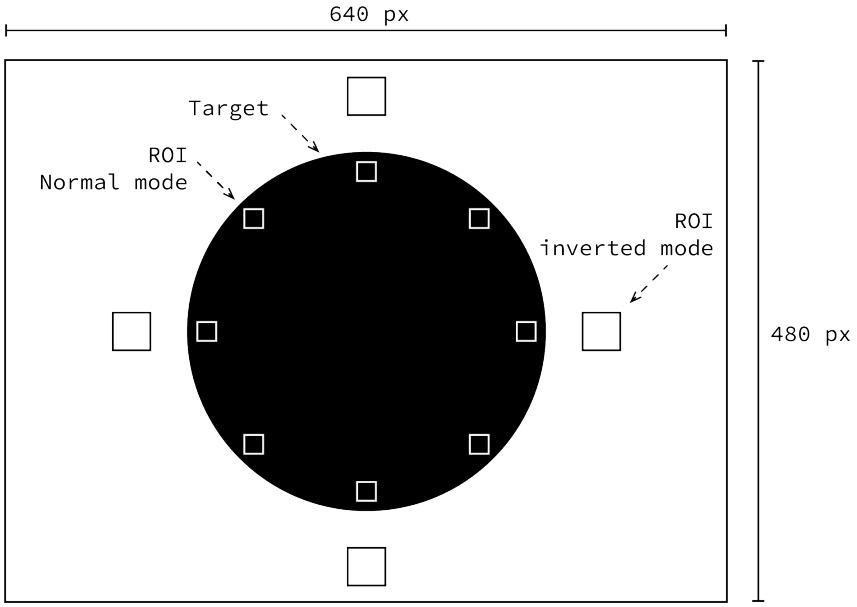
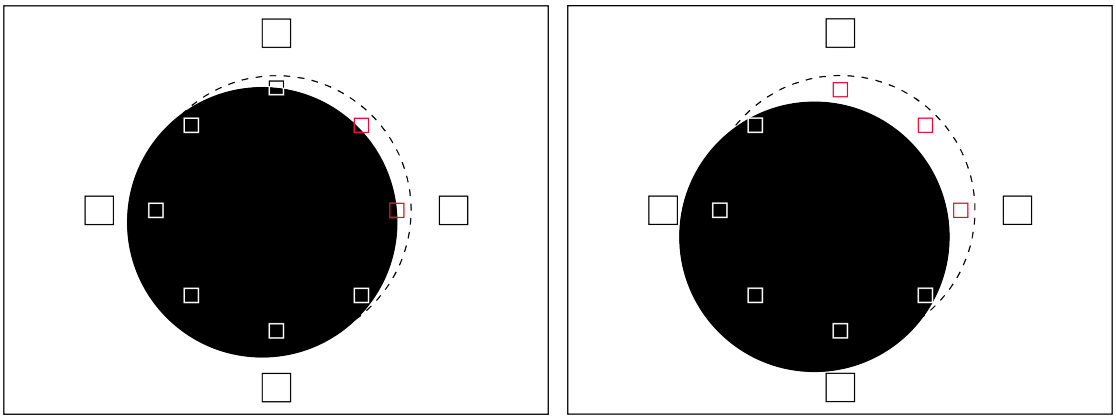


Figure 5: The final image processing is based on the following of thoses ROI. It intend to track the movements of the circular target by looking at the median luminousity of the previously placed ROI.

What to detect The figure 6 shows how the movement is detected. A red ROI indicates a trigged one.



(a) When moving a little, ROI are detecting it (in red are the triggered ones) but the threshold is not overpassed (in this example there is not enough ROI triggered to indicate a movement).

(b) If the target is moving too much, the threshold is over and yet the system must halt.

Figure 6: Example of how the target movements are detected. In dotted the target centered position is indicated.

3.2 Principle of detection

An invalid ROI is defined by a too high median's value (or too low if it's in inverted mode). Mathematically this is translated as the sum of all pixels within the ROI frame, divided by the number of pixels.

A monitoring system will then take as input the median value of all ROI and if their number exceed the threshold, it will be indicated.

4 Further steps

An investigation of images transfers via DMA could lead to a more efficient usage of ressources. This need to be thinked when the system is proved.

5 Planning

A fast planning is extrapolated of this analysis. A much more detailled will be generated and available with the report.
Preparing the whole system and analysing it is planned on 5 weeks.
Working with the flux and doing the simple loop is estimated on 2 weeks.
Finally, implementing the real-time operation must take 7 weeks as it constitute the central task.

Loic FRACHEBOUD

Sion, le 13.06.2019

C VM setup

Preparing a design environment

Begin by installing an Ubuntu as a VM. Disk space must be at least 50[GB] (Vivado is quite big).

Continue with a few useful tools such as git and Atom.

A shared folder between Windows and Ubuntu could be nice. When needed it could mounted with the third command below (automount disabled to keep portability as high as possible with very little configuration when runned by another HW).

```
mkdir ~/shared
sudo adduser bachelor vboxsf
sudo mount -t vboxsf vm_share ~/shared
```

Install HDL Designer

Follow http://mondzeu.ch/wikis/EDA/index.php?title=Install_HDL_Designer

`HDS_2018.1_ixl.exe` is given by M. Corthay

Library files weren't copied

HDL Designer is called directly in

```
/usr/opt/HDS_2018.1/bin/hdl_designer
```

A few things were added to `/etc/profile`

```
#------------------------------------------------------------------------------
# EDA tools
#
export PATH=$PATH:/usr/opt/HDS_2018.1/bin:/usr/opt/Modelsim/modeltech/bin
export
LM_LICENSE_FILE=$LM_LICENSE_FILE:27001@mentorlm.hevs.ch:2100@xilinxlm.hevs.ch
```

Configure HDL Designer

A project is created based on one of the 6th semester lab. A few configurations occurs in the *.bash files.

First take the `/Scripts` and `/Syntax` folder from one of them and the `project_name.bash` file. Granting a 775 access is needed (full power to the owner and the group (Read-Write-Execute) but only RE to other) for the project folder and the `/Scripts`.

Rename all to the desired name and especially replace all old occurence of the old project name in the `project_name.hdp` file located in `project_name/Prefs`. This file make the link between folders and libraries.

Be aware that if HDL Designer installation folder is different of `HDS`, it must be changed in `hdl_designer.bash` file located in `/Scripts`. This could be improved in the long-run by setting a variable who define the folder exact name for the entire script.

Install ModelSim

Follow http://mondzeu.ch/wikis/EDA/index.php?title=Install_Modelsim

A difference occurs at the "Install after HDL designer" step. I've done the following instead.

```
sudo ./install.aol/mgc_install &
```

Testing ModelSim was done from the /bin directory

```
sudo ./usr/opt/Modelsim/modeltech/bin/vsim &
```

If one of the PATH variable is not found, export it.

Install Vivado

The installation take place with Design Edition in `/usr/opt/Xilinx` without packages for Zynq Ultrascale.

Vivado need Gtk package:

```
sudo apt install libcanberra-gtk-module
```

And must be launch as root

```
sudo -H /usr/opt/Xilinx/Vivado/2018.3/bin/vivado
```

Configure a Vivado project

Those two following links give a nice overview on how start and work with the environment :

https://www.knitronics.com/the-zynqberry-patch/getting-started-with-the-zynqberry-in-vivado-2018-2

https://www.digikey.com/eewiki/display/Motley/Getting+Started+with+the+ZynqBerry

Basically consisting of downloading board files and creating a project based on this board.

Steps for the first run

Being by following the DigiKey tutorial (with a Z7007s chip in my case) until *Standalone Software Design*. Here jump to the Knitronics tutorial.

Then some code lines must be commented out in `main.c` (301-311 in my case), edit DDR base address to 0x0 on line 35 of `xparameters.h` (pointed by line 296). Optional but recommended for debugging, the lines 291-294.

Line 383 is replaced by :

```
BootModeRegister &= JTAG_MODE;
```

After, the tutorial recommend a few modification in the Hello World program but I won't use it, so just ignore it and fly back to the DigiKey one on *Application Code*.

Take the application code and replace all the content of `main.c` with it.

This tutorial then give a few instruction on UART (Modify BSP settings), as it's not used, ignore it.

Set the run configurations as explained just above the given application code and especially, set `Run ps7_init` and `Run ps7_post_config` as sometimes the system is held in reset.

From there, it must apply a dynamic PWM in the sense of a variable going forth and back from 0 to 100% on the GPIO7, visible on oscilloscope.

Steps for a new iteration

- Apply again the configuration file to ZYNQ PS block and keep desired peripherals (in this case UART 1, MIO 48-49)
- Run synthesis, implementation and generate bitstream
- Export Hardware (file -> export -> export HW) **with bitstream**
- Launch SDK
- Run configuration which do a full system reset

MediaWiki

Follow http://mondzeu.ch/wikis/mediaWiki under *Installing* to have a full install procedure.

The folder in wich the Wiki will be installed must be editable by user.

```
chown -R bachelor:users www
```

Then the `index.html` must be loaded before the PHP one. This is configurable in `lighttpd.conf`.

Appendix

Resize partition

As the program need around 40[GB] I had to resize the VM. I choose 100[GB] to have a bit of spare space.

```
cd C:\Program Files\ Oracle\VirtualBox
VBoxManage modifiyhd "C:\Users\loic.frachebo\VirtualBox
VMs\bachelor\bachelor.vdi" --resize 100000
```

A GParted VM has been set to resize the partition. This second VM take the *.vdi from the first one as a SATA device.

Then just run GParted, extend the partition and apply.

D Codes

D.1 LEDs and buttons application

```
/******************************************************************************
*
* Copyright (C) 2009 - 2014 Xilinx, Inc.  All rights reserved.
*
* Permission is hereby granted, free of charge, to any person obtaining a copy
* of this software and associated documentation files (the "Software"), to deal
* in the Software without restriction, including without limitation the rights
* to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
* copies of the Software, and to permit persons to whom the Software is
* furnished to do so, subject to the following conditions:
*
* The above copyright notice and this permission notice shall be included in
* all copies or substantial portions of the Software.
*
* Use of the Software is limited solely to applications:
* (a) running on a Xilinx device, or
* (b) that interact with a Xilinx device through a bus or interconnect.
*
* THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
* IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
* FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL
* XILINX  BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY,
* WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF
* OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
* SOFTWARE.
*
* Except as contained in this notice, the name of the Xilinx shall not be used
* in advertising or otherwise to promote the sale, use or other dealings in
* this Software without prior written authorization from Xilinx.
*
******************************************************************************/

/*
 * helloworld.c: simple test application
 *
 * This application configures UART 16550 to baud rate 9600.
 * PS7 UART (Zynq) is not initialized by this application, since
 * bootrom/bsp configures it to baud rate 115200
 *
 * ------------------------------------------------
 * | UART TYPE   BAUD RATE                        |
 * ------------------------------------------------
 *   uartns550   9600
 *   uartlite    Configurable only in HW design
 *   ps7_uart    115200 (configured by bootrom/bsp)
 */

#include <stdio.h>
#include "platform.h"
#include "xil_printf.h"
#include "xgpio.h"
#include "sleep.h"
#include "xtmrctr.h"
#include "xscugic.h"
#include "xil_exception.h"

// Timer variables
#define TMR_INTR_ID     XPAR_FABRIC_AXI_TIMER_0_INTERRUPT_INTR
#define PWM_CONFIG      ((1<<9) | (1<<2) | (1<<1))
#define TMR0_RELOAD     20000
#define TMR1_RELOAD     10000

int status;
u32 control_reg;
u32 pwm;
```

```
66 s8 step = 4;
67
68 // Structures
69 XTmrCtr Timer;
70 XTmrCtr_Config *TimerConPtr;
71 XScuGic Intc;
72 XScuGic_Config *IntcConPtr;
73
74 // User interface variables
75 #define LED_CHANNEL 2
76 #define PUSH_BTN_CHANNEL 1
77
78 #define L1N 0x01
79 #define L2N 0x01
80 #define L3N 0x02
81 #define L4N 0x03
82 #define L5N 0x04
83 #define L6N 0x05
84 #define L7N 0x06
85 #define L8N 0x07
86
87 // all the PWM code is directly from Digikey example
88 void Timer0Callback(void *Inst)
89 {
90   // Set duty cycle
91   pwm += step;
92   if(pwm >= TMR0_RELOAD || pwm == 0)
93   {
94     step = -step;
95   }
96
97   // Load duty cycle into Timer 1
98   XTmrCtr_SetResetValue(&Timer, 1, pwm);
99
100  // Clear interrupt flag (bit 8 in the Control & Status reg)
101  control_reg = XTmrCtr_GetControlStatusReg(TimerConPtr->BaseAddress, 0)|(1<<8);
102  XTmrCtr_SetControlStatusReg(TimerConPtr->BaseAddress, 0, control_reg);
103 }
104 void TimerSetup(void)
105 {
106   // Initialize timer
107   TimerConPtr = XTmrCtr_LookupConfig(XPAR_AXI_TIMER_0_DEVICE_ID);
108   XTmrCtr_CfgInitialize(&Timer, TimerConPtr, TimerConPtr->BaseAddress);
109
110   // Configure timers for PWM usage
111   control_reg = XTmrCtr_GetControlStatusReg(TimerConPtr->BaseAddress, 0)
112    | PWM_CONFIG;
113   XTmrCtr_SetControlStatusReg(TimerConPtr->BaseAddress, 0, control_reg);
114   control_reg = XTmrCtr_GetControlStatusReg(TimerConPtr->BaseAddress, 1)
115    | PWM_CONFIG;
116   XTmrCtr_SetControlStatusReg(TimerConPtr->BaseAddress, 1, control_reg);
117
118   // Set starting values for timers
119   XTmrCtr_SetResetValue(&Timer, 0, TMR0_RELOAD);
120   XTmrCtr_SetResetValue(&Timer, 1, TMR1_RELOAD);
121
122   // Assign function Timer0Callback to be called when interrupt occurs
123   XTmrCtr_SetHandler(&Timer, (XTmrCtr_Handler)Timer0Callback, &Timer);
124
125   // Enable interrupts for Timer 0 only
126   XTmrCtr_EnableIntr(TimerConPtr->BaseAddress, 0);
127 }
128 int IntcSetup(void)
129 {
130   // Enable exceptions in the ARM
131   Xil_ExceptionEnable();
132
133   // Configure Interrupt Controller
134   IntcConPtr = XScuGic_LookupConfig(XPAR_PS7_SCUGIC_0_DEVICE_ID);
135   status = XScuGic_CfgInitialize(&Intc, IntcConPtr, IntcConPtr->CpuBaseAddress);
```

```
136   if(status != XST_SUCCESS)
137   {
138     return XST_FAILURE;
139   }
140
141   // Connect to hardware
142   Xil_ExceptionRegisterHandler(XIL_EXCEPTION_ID_INT,
143     (Xil_ExceptionHandler) XScuGic_InterruptHandler, &Intc);
144   XScuGic_Connect(&Intc, TMR_INTR_ID,
145     (Xil_ExceptionHandler) Timer0Callback, &Timer);
146   XScuGic_Enable(&Intc, TMR_INTR_ID);
147   return XST_SUCCESS;
148 }
149
150 int main()
151 {
152   init_platform();
153
154   // PWM stuff
155   TimerSetup();
156   IntcSetup();
157   print("Hello World!\n\r");
158   XTmrCtr_Start(&Timer, 0);
159   XTmrCtr_Start(&Timer, 1);
160   print("Timers started.\n\r");
161
162   // leds and buttons stuff
163   XGpio user_interface;
164   u32 buttons = 0;
165   u32 slave_leds = 0;
166
167   // Initialize GPIO
168   XGpio_Initialize(&user_interface, XPAR_AXI_GPIO_0_DEVICE_ID);
169
170   while(1) {
171     // constantly read buttons and write their status into LEDs register
172     buttons = XGpio_DiscreteRead(&user_interface, PUSH_BTN_CHANNEL);
173     XGpio_DiscreteWrite(&user_interface, LED_CHANNEL, buttons);
174   }
175
176   cleanup_platform();
177   return 0;
178 }
```

Listing 2: Program written to handle LEDs and buttons, based on what was done in the hands-on (see subsection 9.2)

D.2 Test bench basic

```
ARCHITECTURE test OF loopback_tester IS

  constant clockFrequency: real := 66.0E6;
  constant clockPeriod: time := (1.0/clockFrequency) * 1 sec;
  constant pixelPeriod: time := (3.0/clockFrequency) * 1 sec;
  signal sClock: std_uLogic := '1';
  signal sData_0: unsigned(grayscaleBitNb-1 downto 0) := (others => '0');
  signal sSOL: std_uLogic := '0';
  signal sSOF: std_uLogic := '0';
  signal counterEn: std_uLogic := '0';
  signal lineCounter: integer := 0;
  signal frameDone: std_uLogic;   -- indicate the end of a frame generation
  signal beginFrame: std_uLogic;  -- used to start a frame generation
  signal lineDone: std_uLogic;    -- indicate that a line is in generation
  signal counter: unsigned(paramBitNb-1 downto 0);
  signal pixelXcounter: unsigned(dataBitNb-1 downto 0);

  ------------------------------------------------------------------------------
  -- State Machine
  ------------------------------------------------------------------------------
  -- possible states
  type STATE_TYPE is (
    idle,
    start_frame,
    start_line,
    doing_line
  );

  -- Declare current and next state signals
  signal current_state : STATE_TYPE;
  signal next_state : STATE_TYPE;

begin
  ------------------------------------------------------------------------------
                                                        -- clock and reset
  sClock <= not sClock after clockPeriod/2;
  clock <= transport sClock after clockPeriod*9/10;
  reset <= '1', '0' after 2*clockPeriod;

  ------------------------------------------------------------------------------
                                                      -- enable peripheral
  en <= '0', '1' after 400 ns;

  -- set to generate frames
  beginFrame <= '0', '1' after 600 ns;--, '0' after 800 ns;
  -- todo work on a system to request the generation of n pictures


  -- set parameters
  roi_x <= to_unsigned(3, paramBitNb);
  roi_y <= to_unsigned(3, paramBitNb);
  roi_size <= to_unsigned(3, paramBitNb);
  roi_threshold <= to_unsigned(900, paramBitNb);

  ------------------------------------------------------------------------------
                                                        -- send a picture

  ------------------------------------------------------------------------------
  -- State Machine
  ------------------------------------------------------------------------------
  -- Process name : clocked_proc
  -- Description : state's flip-flop handling
  ------------------------------------------------------------------------------
  clocked_proc : process(sClock, reset)
  begin
    if (reset = '1') then
      current_state <= idle;
    elsif rising_edge(sClock) then
```

```
69      current_state <= next_state;
70    end if;
71  end process clocked_proc;
72
73  ------------------------------------------------------------------------------
74  -- Process name : nextstate_proc
75  -- Description : determine the next state according to :
76  --   1) actual state
77  --   2) incomming events
78  ------------------------------------------------------------------------------
79  nextstate_proc : process(current_state, beginFrame, frameDone, lineDone)
80  begin
81    case current_state is
82      when idle =>
83        -- wait until start of frame
84        if beginFrame = '1' then
85          next_state <= start_frame;
86        end if;
87
88      -- start a frame and so begin by a line
89      when start_frame =>
90        next_state <= start_line;
91
92      -- start a line (send SOL) and wait completion in doing_line
93      when start_line =>
94        next_state <= doing_line;
95
96      -- wait for the end of a line
97      when doing_line =>
98        if frameDone then
99          next_state <= idle;
100
101       elsif lineDone then
102         -- create one more line if needed and last one terminated
103         next_state <= start_line;
104
105       end if;
106       -- else continue generation
107
108     when others =>
109       next_state <= idle;
110   end case;
111 end process nextstate_proc;
112
113 ------------------------------------------------------------------------------
114 -- Process name : output_in_proc
115 -- Description : action on state entry
116 ------------------------------------------------------------------------------
117 output_in_proc : process(current_state, lineCounter)
118 begin
119  -- Default Assignment
120   frameDone <= '0';
121   sSOF <= '0';
122   sSOL <= '0';
123
124  -- Combined Actions
125  case current_state is
126   when idle =>
127     -- do nothing
128
129   when start_frame =>
130     -- start counter and send star of frame signal
131     sSOF <= '1';
132
133   when start_line =>
134     -- send start of line signal
135     counterEn <= '1';
136     sSOL <= '1';
137
138   when doing_line =>
```

```
      -- checking if more lines are needed
      if lineCounter = imageHeigth - 1 then
        frameDone <= '1';
        counterEn <= '0';
      end if;

    when others =>
      null;
  end case;
end process output_in_proc;



-------------------------------------------------------------------------------
-- Process name : generateData
-- Description : counter driven by state machine to generate data
-------------------------------------------------------------------------------
generateData : process(sClock)
begin
  if rising_edge(sClock) then
    lineDone <= '0';

    -- if counter is enabled, increment output on a lower speed than clock
    if counterEn = '1' then
      counter <= counter + 1;

      -- there is no synchronization signal, so duration must be known
      if counter = dataDuration - 1 then
        counter <= (others => '0');
        pixelXcounter <= pixelXcounter + 1;
        sData_0 <= sData_0 + grayscaleStep;

        -- upper limit is not 1023 as step is not a divisor
        if pixelXcounter = imageWidth - 1 then
          -- one line is drawn
          lineDone <= '1';
          lineCounter <= lineCounter + 1;
          sData_0 <= (others => '0');
          pixelXcounter <= (others => '0');
        end if;
      end if;

    -- if not counting, reset signals
    else
      sData_0 <= (others => '0');
      counter <= (others => '0');
      pixelXcounter <= (others => '0');
      lineCounter <= 0;
    end if;
  end if;
end process;

-- output signals
data_0 <= resize(sData_0, dataBitNb);
start_of_frame <= sSOF;
start_of_line <= sSOL;

END ARCHITECTURE test;
```

Listing 3: Test bench written to test basic functionalities

D.3 Test bench VGA

```
-- VHDL Entity ImageProcessing_test.loopback_tester.interface
--
-- Created:
--          by - bachelor.bachelor (bachelor-vm)
--          at - 11:33:30 08/13/19
--
-- Generated by Mentor Graphics' HDL Designer(TM) 2018.1 (Build 12)
--
LIBRARY ieee;
  USE ieee.std_logic_1164.all;
  USE ieee.numeric_std.ALL;

LIBRARY ImageProcessing;
  USE ImageProcessing.general.ALL;

ENTITY loopback_tester IS
   GENERIC(
      dataBitNb     : positive := 16;
      paramBitNb    : positive := 16;
      imageWidth    : positive := 40;
      imageHeigth   : positive := 30;
      grayscaleStep : positive := 25;
      dataDuration  : positive := 8;
      nbrInputData  : positive := 8
   );
   PORT(
      id              : IN     unsigned (dataBitNb-1 DOWNTO 0);
      result          : IN     std_ulogic;
      valid           : IN     std_ulogic;
      clock           : OUT    std_ulogic;
      data_0          : OUT    unsigned (dataBitNb-1 DOWNTO 0);
      data_1          : OUT    unsigned (dataBitNb-1 DOWNTO 0);
      data_2          : OUT    unsigned (dataBitNb-1 DOWNTO 0);
      data_3          : OUT    unsigned (dataBitNb-1 DOWNTO 0);
      data_4          : OUT    unsigned (dataBitNb-1 DOWNTO 0);
      data_5          : OUT    unsigned (dataBitNb-1 DOWNTO 0);
      data_6          : OUT    unsigned (dataBitNb-1 DOWNTO 0);
      data_7          : OUT    unsigned (dataBitNb-1 DOWNTO 0);
      en              : OUT    std_ulogic;
      reset           : OUT    std_ulogic;
      roi_mode        : OUT    std_ulogic;
      roi_size        : OUT    unsigned (paramBitNb-1 DOWNTO 0);
      roi_threshold   : OUT    unsigned (paramBitNb-1 DOWNTO 0);
      roi_x           : OUT    unsigned (paramBitNb-1 DOWNTO 0);
      roi_y           : OUT    unsigned (paramBitNb-1 DOWNTO 0);
      start_of_frame  : OUT    std_ulogic;
      start_of_line   : OUT    std_ulogic
   );

-- Declarations

END loopback_tester ;
```

Listing 4: Test bench entity

```
ARCHITECTURE test OF loopback_tester IS

  constant clockFrequency: real := 66.0E6;
  constant clockPeriod: time := (1.0/clockFrequency) * 1 sec;
  constant pixelPeriod: time := (3.0/clockFrequency) * 1 sec;
  signal sClock: std_uLogic := '1';
  signal sData_0: unsigned(data_0'range) := (others => '0');
  signal sData_2: unsigned(data_1'range) := (others => '0');
  signal sData_1: unsigned(data_2'range) := (others => '0');
  signal sData_3: unsigned(data_3'range) := (others => '0');
  signal sData_4: unsigned(data_4'range) := (others => '0');
  signal sData_5: unsigned(data_5'range) := (others => '0');
  signal sData_6: unsigned(data_6'range) := (others => '0');
```

```
  signal sData_7: unsigned(data_7'range) := (others => '0');
  signal sSOL: std_uLogic := '0';
  signal sSOF: std_uLogic := '0';
  signal counterEn: std_uLogic := '0';
  signal lineCounter: integer := 0;
  signal frameDone: std_uLogic;   -- indicate the end of a frame generation
  signal beginFrame: std_uLogic;  -- used to start a frame generation
  signal lineDone: std_uLogic;    -- indicate that a line is in generation
  signal counter: unsigned(paramBitNb-1 downto 0);
  signal pixelXcounter: unsigned(dataBitNb-1 downto 0);

  ------------------------------------------------------------------------
  -- State Machine
  ------------------------------------------------------------------------
  -- possible states
  type STATE_TYPE is (
    idle,
    start_frame,
    start_line,
    doing_line
  );

  -- Declare current and next state signals
  signal current_state : STATE_TYPE;
  signal next_state : STATE_TYPE;

begin
  ------------------------------------------------------------------------
                                                    -- clock and reset
  sClock <= not sClock after clockPeriod/2;
  clock <= transport sClock after clockPeriod*9/10;
  reset <= '1', '0' after 2*clockPeriod;

  ------------------------------------------------------------------------
                                                  -- enable peripheral
  en <= '0', '1' after 400 ns;

  -- set to generate frames
  beginFrame <= '0', '1' after 600 ns;--, '0' after 800 ns;
  -- todo work on a system to request the generation of n pictures


  -- set parameters
  roi_x <= to_unsigned(632, roi_x'length);
  roi_y <= to_unsigned(471, roi_y'length);
  roi_mode <= '0';
  roi_size <= to_unsigned(8, roi_size'length);
  roi_threshold <= to_unsigned(1000, roi_threshold'length);

  ------------------------------------------------------------------------
                                                    -- send a picture

  ------------------------------------------------------------------------
  -- State Machine
  ------------------------------------------------------------------------
  -- Process name : clocked_proc
  -- Description : state's flip-flop handling
  ------------------------------------------------------------------------
  clocked_proc : process(sClock, reset)
  begin
    if (reset = '1') then
      current_state <= idle;
    elsif rising_edge(sClock) then
      current_state <= next_state;
    end if;
  end process clocked_proc;

  ------------------------------------------------------------------------
  -- Process name : nextstate_proc
  -- Description : determine the next state according to :
```

```
--  1) actual state
--  2) incomming events
-------------------------------------------------------------------------------
nextstate_proc : process(current_state, beginFrame, frameDone, lineDone)
begin
  case current_state is
    when idle =>
      -- wait until start of frame
      if beginFrame = '1' then
        next_state <= start_frame;
      end if;

    -- start a frame and so begin by a line
    when start_frame =>
      next_state <= start_line;

    -- start a line (send SOL) and wait completion in doing_line
    when start_line =>
      next_state <= doing_line;

    -- wait for the end of a line
    when doing_line =>
      if frameDone then
        next_state <= idle;

      elsif lineDone then
        -- create one more line if needed and last one terminated
        next_state <= start_line;

      end if;
      -- else continue generation

    when others =>
      next_state <= idle;
  end case;
end process nextstate_proc;

-------------------------------------------------------------------------------
-- Process name : output_in_proc
-- Description : action on state entry
-------------------------------------------------------------------------------
output_in_proc : process(current_state, lineCounter)
begin
 -- Default Assignment
  frameDone <= '0';
  sSOF <= '0';
  sSOL <= '0';

 -- Combined Actions
 case current_state is
  when idle =>
    -- do nothing

  when start_frame =>
    -- start counter and send star of frame signal
    sSOF <= '1';

  when start_line =>
    -- send start of line signal
    counterEn <= '1';
    sSOL <= '1';

  when doing_line =>
    -- checking if more lines are needed
    if lineCounter = imageHeigth - 1 then
      frameDone <= '1';
      counterEn <= '0';
    end if;

  when others =>
```

```
      null;
  end case;
end process output_in_proc;



-------------------------------------------------------------------------------
-- Process name : generateData
-- Description : counter driven by state machine to generate data
-------------------------------------------------------------------------------
generateData : process(sClock)
begin
  if rising_edge(sClock) then
    lineDone <= '0';

    -- if counter is enabled, increment output on a lower speed than clock
    if counterEn = '1' then
      counter <= counter + 1;

      -- there is no synchronization signal, so duration must be known
      if counter = dataDuration - 1 then
        counter <= (others => '0');
        pixelXcounter <= pixelXcounter + nbrInputData;
        sData_0 <= sData_0 + grayscaleStep*nbrInputData;
        sData_1 <= sData_0 + grayscaleStep*nbrInputData + 1;
        sData_2 <= sData_0 + grayscaleStep*nbrInputData + 2;
        sData_3 <= sData_0 + grayscaleStep*nbrInputData + 3;
        sData_4 <= sData_0 + grayscaleStep*nbrInputData + 4;
        sData_5 <= sData_0 + grayscaleStep*nbrInputData + 5;
        sData_6 <= sData_0 + grayscaleStep*nbrInputData + 6;
        sData_7 <= sData_0 + grayscaleStep*nbrInputData + 7;

        -- upper limit is not 1023 as step is not a divisor
        -- and this counter counts 8 by 8 so ends on imageWidth
        if pixelXcounter = imageWidth then
          -- one line is drawn
          lineDone <= '1';
          lineCounter <= lineCounter + 1;
          -- reset data for next one
          sData_0 <= (others => '0');
          sData_1 <= to_unsigned(1, sData_0'length);
          sData_2 <= to_unsigned(2, sData_0'length);
          sData_3 <= to_unsigned(3, sData_0'length);
          sData_4 <= to_unsigned(4, sData_0'length);
          sData_5 <= to_unsigned(5, sData_0'length);
          sData_6 <= to_unsigned(6, sData_0'length);
          sData_7 <= to_unsigned(7, sData_0'length);
          pixelXcounter <= (others => '0');
        end if;
      end if;

    -- if not counting, reset signals
    else
      sData_0 <= (others => '0');
      sData_1 <= to_unsigned(1, sData_0'length);
      sData_2 <= to_unsigned(2, sData_0'length);
      sData_3 <= to_unsigned(3, sData_0'length);
      sData_4 <= to_unsigned(4, sData_0'length);
      sData_5 <= to_unsigned(5, sData_0'length);
      sData_6 <= to_unsigned(6, sData_0'length);
      sData_7 <= to_unsigned(7, sData_0'length);
      counter <= (others => '0');
      pixelXcounter <= (others => '0');
      lineCounter <= 0;
    end if;
  end if;
end process;

-- output signals
data_0 <= sData_0;
```

```
224   data_1 <= sData_1;
225   data_2 <= sData_2;
226   data_3 <= sData_3;
227   data_4 <= sData_4;
228   data_5 <= sData_5;
229   data_6 <= sData_6;
230   data_7 <= sData_7;
231   start_of_frame <= sSOF;
232   start_of_line <= sSOL;
233 
234 END ARCHITECTURE test;
```

Listing 5: Test bench written to test system with a VGA frame format

D.4 Test bench accumulator

```
-- VHDL Entity ImageProcessing_test.accumulator_tb.symbol
--
-- Created:
--          by - francois.corthay.UNKNOWN (WEA30906)
--          at - 14:48:16 25.02.2019
--
-- Generated by Mentor Graphics' HDL Designer(TM) 2018.1 (Build 12)
--


ENTITY accumulator_tb IS
-- Declarations

END accumulator_tb ;
```

Listing 6: Test bench entity for accumulator

```
--
-- VHDL Architecture ImageProcessing_test.accumulator_tester.tb
--
-- Created:
--          by - bachelor.bachelor (bachelor-vm)
--          at - 15:52:14 08/13/19
--
-- using Mentor Graphics HDL Designer(TM) 2018.1 (Build 12)
--
LIBRARY ieee;
  USE ieee.std_logic_1164.all;
  USE ieee.numeric_std.ALL;

LIBRARY ImageProcessing;
  USE ImageProcessing.general.ALL;

ENTITY accumulator_tester IS
  GENERIC(
    constant dataBitNb: positive := 16;
    constant paramBitNb: positive := 16;
    constant grayscaleBitNb: positive := 10;
    constant nbrInputData: positive := 8;
    constant maxRoiSize: positive := 8
  );
   PORT(
      id              : IN     unsigned (dataBitNb-1 DOWNTO 0);
      result          : IN     std_ulogic;
      valid           : IN     std_ulogic;
      endOfProcessing : OUT    std_logic;
      newPixels       : OUT    std_ulogic;
      roi_id          : OUT    unsigned (dataBitNb-1 DOWNTO 0);
      roi_mode        : OUT    std_ulogic;
      roi_threshold   : OUT    unsigned (paramBitNb-1 DOWNTO 0);
      start_of_frame  : OUT    std_ulogic;
      value           : OUT    data_array
   );

-- Declarations

END accumulator_tester ;

--
ARCHITECTURE tb OF accumulator_tester IS
  constant t_pulse: time := 30 ns;
  constant t_data: time := 100 ns;
  constant t_reset: time := 50 ns;
  constant t_betweenTests: time := 200 ns;
BEGIN
  ------------------------------------------------------------------------
                                                    -- test general behaviour
  process
```

```
52  begin
53    wait for t_reset;
54    start_of_frame <= '0', '1' after 10 ns, '0' after t_pulse;
55
56    -- test a value under threshold, normal mode
57    roi_id <= to_unsigned(16, roi_id'length);
58    roi_mode <= '0';
59    roi_threshold <= to_unsigned(100, roi_threshold'length);
60    endOfProcessing <= '0';
61
62    -- generate values to accumulate, expected accumulation is 7! * 10 * 7
63    -- first batch is used to see if accumulator reset on SOF
64    for i in 0 to maxRoiSize-1 loop
65      for j in 0 to nbrInputData-1 loop
66        value(j) <= to_unsigned(j*10, value(j)'length);
67      end loop;
68      newPixels <= '1', '0' after t_pulse;
69      wait for t_data;
70    end loop;
71
72    endOfProcessing <= '1', '0' after t_pulse;
73
74  ------------------------------------------------------------------------
75  -- test normal mode, accumulation is set at 2240
76  --    - above threshold
77  --    - under threshold
78  --    - equal to threshold
79  ------------------------------------------------------------------------
80    wait for t_betweenTests;
81    start_of_frame <= '0', '1' after 10 ns, '0' after t_pulse;
82    roi_id <= to_unsigned(16, roi_id'length);
83    roi_mode <= '0';
84    roi_threshold <= to_unsigned(100, roi_threshold'length);
85    endOfProcessing <= '0';
86    -- make accumulation
87    wait for t_data;
88    for i in 0 to maxRoiSize-1 loop
89      newPixels <= '1', '0' after t_pulse;
90      wait for t_data;
91    end loop;
92    endOfProcessing <= '1', '0' after t_pulse;
93
94
95    wait for t_betweenTests;
96    start_of_frame <= '0', '1' after 10 ns, '0' after t_pulse;
97    roi_threshold <= to_unsigned(3000, roi_threshold'length);
98    endOfProcessing <= '0';
99    -- make accumulation
100   wait for t_data;
101   for i in 0 to maxRoiSize-1 loop
102     newPixels <= '1', '0' after t_pulse;
103     wait for t_data;
104   end loop;
105   endOfProcessing <= '1', '0' after t_pulse;
106
107
108   wait for t_betweenTests;
109   start_of_frame <= '0', '1' after 10 ns, '0' after t_pulse;
110   roi_threshold <= to_unsigned(2240, roi_threshold'length);
111   endOfProcessing <= '0';
112   -- make accumulation
113   wait for t_data;
114   for i in 0 to maxRoiSize-1 loop
115     newPixels <= '1', '0' after t_pulse;
116     wait for t_data;
117   end loop;
118   endOfProcessing <= '1', '0' after t_pulse;
119
120 ------------------------------------------------------------------------
121 -- test inverted mode, accumulation is set at 1960
```

```
122   --    - above threshold
123   --    - under threshold
124   --    - equal to threshold
125   ------------------------------------------------------------------------------
126     wait for t_betweenTests;
127     roi_mode <= '1';
128     start_of_frame <= '0', '1' after 10 ns, '0' after t_pulse;
129     roi_threshold <= to_unsigned(100, roi_threshold'length);
130     endOfProcessing <= '0';
131     -- make accumulation
132     wait for t_data;
133     for i in 0 to maxRoiSize-1 loop
134       newPixels <= '1', '0' after t_pulse;
135       wait for t_data;
136     end loop;
137     endOfProcessing <= '1', '0' after t_pulse;
138
139
140     wait for t_betweenTests;
141     start_of_frame <= '0', '1' after 10 ns, '0' after t_pulse;
142     roi_threshold <= to_unsigned(3000, roi_threshold'length);
143     endOfProcessing <= '0';
144     -- make accumulation
145     wait for t_data;
146     for i in 0 to maxRoiSize-1 loop
147       newPixels <= '1', '0' after t_pulse;
148       wait for t_data;
149     end loop;
150     endOfProcessing <= '1', '0' after t_pulse;
151
152
153     wait for t_betweenTests;
154     start_of_frame <= '0', '1' after 10 ns, '0' after t_pulse;
155     roi_threshold <= to_unsigned(2240, roi_threshold'length);
156     endOfProcessing <= '0';
157     -- make accumulation
158     wait for t_data;
159     for i in 0 to maxRoiSize-1 loop
160       newPixels <= '1', '0' after t_pulse;
161       wait for t_data;
162     end loop;
163     endOfProcessing <= '1', '0' after t_pulse;
164
165     -- END SIMULATION
166     wait for t_betweenTests;
167     assert false
168       report "End of simulation"
169       severity failure;
170     wait;
171
172 end process;
173
174 END ARCHITECTURE tb;
```

Listing 7: Test bench written to test the accumulator

D.5 Tests results

roi						expected			observed			passed	note
x	y	size	mode	threshold	overlap	accumulated value	result	valid	accumulated value	result	valid		
10	10	3	normal	1000	no	99	1	1	99	1	1	yes	under threshold, must return valid
10	10	3	normal	10	no	99	0	1	99	0	1	yes	above threshold, must return invalid
10	10	3	normal	99	no	99	0	1	99	0	1	yes	equal to threshold, must return invalid
10	10	3	inverted	1000	no	99	0	1	99	0	1	yes	under threshold, must return invalid
10	10	3	inverted	10	no	99	1	1	99	1	1	yes	above threshold, must return valid
10	10	3	inverted	99	no	99	0	1	99	0	0	yes	equal to threshold, must return invalid
10	10	1	normal	100	not applicable	10	1	1	10	1	1	yes	
10	10	1	normal	1	not applicable	10	0	1	10	0	1	yes	
10	10	1	normal	10	not applicable	10	0	1	10	0	1	yes	
10	10	1	inverted	1	not applicable	10	1	1	10	1	1	yes	
10	10	1	inverted	10	not applicable	10	0	1	10	0	1	yes	
10	10	1	inverted	100	not applicable	10	0	1	10	0	1	yes	
10	10	2	inverted	100	no	42	0	1	42	0	1	yes	not testing all case as 1 and 3 are ok, assuming between is too
10	10	4	normal	100	no	184	0	1	184	0	1	yes	
10	10	5	normal	1000	no	300	1	1	300	1	1	yes	
9	10	6	normal	1000	no	414	1	1	414	1	1	yes	
9	10	7	normal	1000	no	588	1	1	588	1	1	yes	
16	10	8	normal	1000	no	1248	0	1	1248	0	1	yes	
12	10	8	normal	1000	yes	992	1	1	992	1	1	yes	Good repartition to next pixel batch
12	10	7	normal	1000	yes	735	1	1	735	1	1	yes	
13	10	7	normal	1000	yes	784	1	1	784	1	1	yes	
14	10	7	normal	1000	yes	833	1	1	833	1	1	yes	
15	10	7	normal	1000	yes	882	1	1	882	1	1	yes	
16	10	7	normal	1000	yes	931	1	1	931	1	1	yes	
17	10	7	normal	1000	yes	980	1	1	980	1	1	yes	
0	0	8	normal	1000	yes	224	1	1	224	1	1	yes	Okay when in 0/0
0	471	8	normal	1000	yes	224	1	1	224	1	1	yes	Okay for next corner
632	0	8	normal	1000	yes	40672	0	1	40672	0	1	yes	Okay for next corner
632	471	8	normal	1000	yes	40672	0	1	40672	0	1	yes	Okay for last corner

D.6 Compute ROI

```
--
-- VHDL Architecture ImageProcessing.compute_roi.behaviour
--
-- Created:
--          by - bachelor.bachelor (bachelor-vm)
--          at - 11:23:14 07/19/19
--
-- using Mentor Graphics HDL Designer(TM) 2018.1 (Build 12)
--
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.all;

ENTITY compute_roi IS
  GENERIC(
    dataBitNb       : positive := 16;
    paramBitNb      : positive := 16;
    imageWidth      : positive := 640;
    imageHeigth     : positive := 480;
    grayscaleBitNb  : positive := 10;
    grayscaleMax    : positive := 1024;
    grayscaleStep   : positive := 1;
    dataDuration    : positive := 8
  );
  PORT(
    clock           : IN     std_ulogic;
    data_0          : IN     unsigned (dataBitNb-1 DOWNTO 0);
    data_1          : IN     unsigned (dataBitNb-1 DOWNTO 0);
    data_2          : IN     unsigned (dataBitNb-1 DOWNTO 0);
    data_3          : IN     unsigned (dataBitNb-1 DOWNTO 0);
    data_4          : IN     unsigned (dataBitNb-1 DOWNTO 0);
    data_5          : IN     unsigned (dataBitNb-1 DOWNTO 0);
    data_6          : IN     unsigned (dataBitNb-1 DOWNTO 0);
    data_7          : IN     unsigned (dataBitNb-1 DOWNTO 0);
    en              : IN     std_ulogic;
    reset           : IN     std_ulogic;
    roi_size        : IN     unsigned (paramBitNb-1 DOWNTO 0);
    roi_threshold   : IN     unsigned (paramBitNb-1 DOWNTO 0);
    roi_mode        : IN     std_ulogic;
    roi_x           : IN     unsigned (paramBitNb-1 DOWNTO 0);
    roi_y           : IN     unsigned (paramBitNb-1 DOWNTO 0);
    start_of_frame  : IN     std_ulogic;
    start_of_line   : IN     std_ulogic;
    id              : OUT    unsigned (dataBitNb-1 DOWNTO 0);
    result          : OUT    std_ulogic;
    valid           : OUT    std_ulogic
  );

  -- Declarations

END ENTITY compute_roi;

--
ARCHITECTURE behaviour OF compute_roi IS
  -- counters
  signal counter: unsigned(dataBitNb-1 downto 0); -- way too big, must only count until
    dataDuration
  signal pixelCounter_x: unsigned(dataBitNb-1 downto 0); -- count pixels in X
  signal pixelCounter_y: unsigned(dataBitNb-1 downto 0); -- count pixels in Y
  signal pixelCounter_x_old: unsigned(dataBitNb-1 downto 0);

  -- flags
  signal receiving: std_uLogic;
  signal pixelInRoiFound: std_uLogic;
  signal accumulate: std_uLogic;
  signal lastPixel: std_uLogic;

  -- buffers and variables
```

```
68   signal currentPixelValue: unsigned(dataBitNb-1 downto 0); -- store value for accumulator
69   signal accumulator: unsigned(dataBitNb-1 downto 0) := (others => '0');
70
71   -- coordinates
72   -- offsets to ROI x/y position
73   signal roi_offset_x: unsigned(paramBitNb-1 downto 0);
74   signal roi_offset_y: unsigned(paramBitNb-1 downto 0);
75
76   -- encode the number of input data lines
77   constant nbrInputData: positive := 8
78
79   ------------------------------------------------------------------------------
80   -- State Machine
81   ------------------------------------------------------------------------------
82   -- possible states
83   type STATE_TYPE is (
84     idle,
85     search_pixels,
86     accumulate_value,
87     finished
88   );
89
90   -- Declare current and next state signals
91   signal current_state : STATE_TYPE;
92   signal next_state : STATE_TYPE;
93
94 BEGIN
95
96   ------------------------------------------------------------------------------
97   -- State Machine
98   ------------------------------------------------------------------------------
99   -- Process name : clocked_proc
100  -- Description : state's flip-flop handling
101  ------------------------------------------------------------------------------
102  clocked_proc : process(clock, reset)
103  begin
104    if reset = '1' then
105      current_state <= idle;
106    elsif rising_edge(clock) then
107      current_state <= next_state;
108    end if;
109  end process clocked_proc;
110
111  ------------------------------------------------------------------------------
112  -- Process name : nextstate_proc
113  -- Description : determine the next state according to :
114  --  1) actual state
115  --  2) incomming events
116  ------------------------------------------------------------------------------
117  nextstate_proc : process(current_state, start_of_frame, lastPixel, pixelInRoiFound)
118  begin
119    case current_state is
120      -- when a new frame occurs, begin the analysis
121      when idle =>
122        if start_of_frame then
123          next_state <= search_pixels;
124        end if;
125
126      -- search is only ended by the arriving of a new frame
127      when search_pixels =>
128        if pixelInRoiFound then
129          next_state <= accumulate_value;
130        end if;
131
132      -- tell the accumulator to add a new value, if it was the last, finish
133      when accumulate_value =>
134        if lastPixel then
135          next_state <= finished;
136        else
137          next_state <= search_pixels;
```

```
138         end if;
139
140       when others =>
141         next_state <= idle;
142     end case;
143   end process nextstate_proc;
144
145   ------------------------------------------------------------------------------
146   -- Process name : output_in_proc
147   -- Description : action on state entry
148   ------------------------------------------------------------------------------
149   output_in_proc : process(current_state)
150   begin
151    -- Default Assignment
152    valid <= '0';
153    accumulate <= '0';
154
155    -- Combined Actions
156    case current_state is
157     when idle =>
158       -- do nothing
159
160     -- search desired pixels by starting counters
161     when search_pixels =>
162       receiving <= '1';
163
164     when accumulate_value =>
165       accumulate <= '1';
166
167     -- indicate to output that output data are valids
168     when finished =>
169       receiving <= '0';
170       valid <= '1';
171     when others =>
172       null;
173    end case;
174   end process output_in_proc;
175
176   ------------------------------------------------------------------------------
177   -- Process name : countInputFlow
178   -- Description : counts incomming bit flow for synchronization
179   ------------------------------------------------------------------------------
180   countInputFlow : process(clock, reset)
181   begin
182     if reset = '1' then
183       counter <= (others => '0');
184
185     elsif rising_edge(clock) then
186       if receiving = '1' and counter < dataDuration - 1 then
187         counter <= counter + 1;
188       else
189         counter <= (others => '0');
190       end if;
191     end if;
192   end process countInputFlow;
193
194   ------------------------------------------------------------------------------
195   -- Process name : countPixelX
196   -- Description : count pixels x to further know if the actual pixel is in ROI
197   ------------------------------------------------------------------------------
198   countPixelX : process(counter, reset, start_of_frame, start_of_line)
199   begin
200     -- reset counters in case of a reset or a new frame incomming or sync to SOL
201     if reset = '1' or start_of_frame = '1' or start_of_line = '1' then
202       pixelCounter_x <= (others => '0');
203     end if;
204
205     -- triggering on half duration allow to read value time-centered
206     if counter = dataDuration/2 then
207       -- increment x counter as pixels come 1 per input data line
```

```
208       pixelCounter_x <= pixelCounter_x + nbrInputData;  -- maybe -1, need simul. to validate
209     end if;
210   end process countPixelX;
211
212   ------------------------------------------------------------------------------
213   -- Process name : countPixelY
214   -- Description : count pixels y to further know if the actual pixel is in ROI
215   ------------------------------------------------------------------------------
216   countPixelY : process(reset, start_of_frame, start_of_line)
217   begin
218     -- reset counters in case of a reset or a new frame incomming
219     if reset = '1' or start_of_frame = '1' then
220       pixelCounter_y <= (others => '0');
221     end if;
222
223     -- if a new line occurs, increment y counter and reset x counter
224     if start_of_line = '1' then
225       pixelCounter_y <= pixelCounter_y + 1;
226     end if;
227   end process countPixelY;
228
229   ------------------------------------------------------------------------------
230   -- Process name : detectPixel
231   -- Description : sync process who check if actual pixel is in ROI and the last
232   ------------------------------------------------------------------------------
233   detectPixel : process(reset, clock)
234     -- local aliases to keep an easyto read code
235     variable pixel_0: unsigned(dataBitNb-1 downto 0) := 0;
236     variable pixel_7: unsigned(dataBitNb-1 downto 0) := 0;
237     variable currentRoi_x: unsigned(paramBitNb-1 downto 0) := 0;
238     variable currentRoi_y: unsigned(paramBitNb-1 downto 0) := 0;
239
240   begin
241     if reset = '1' then
242       pixelInRoiFound <= '0';
243       lastPixel <= '0';
244       pixel_0 <= (others => '0');
245       pixel_7 <= (others => '0');
246
247     elsif rising_edge(clock) then
248       -- if new pixels are incomming
249       if pixelCounter_x_old /= pixelCounter_x then
250         -- update pixels coordinates and local aliases
251         pixel_0 <= pixelCounter_x - nbrInputData;
252         pixel_7 <= pixelCounter_x;
253         currentRoi_x <= roi_x + roi_offset_x;
254         currentRoi_y <= roi_y + roi_offset_y;
255
256         -- if current pixels
257         if pixel_0 <= currentRoi_x and pixel_7 > currentRoi_x then
258           -- y-axis verification
259           if pixelCounter_y >= roi_y and pixelCounter_y < roi_y + roi_size then
260             -- copy value in buffer and indicate it
261             currentPixelValue <= data_0;
262             pixelInRoiFound <= '1';
263             roi_offset_x <= roi_offset_x + 1;
264           else
265           end if;
266         else
267         end if;
268
269         pixelCounter_x_old <= pixelCounter_x;
270
271       -- reset flag as the actual is already indicated (or not at all in ROI)
272       else
273         pixelInRoiFound <= '0';
274       end if;
275
276       -- detect if all ROI pixel's values are accumulated
277       if pixelCounter_x = roi_x + roi_size - 1 and pixelCounter_y = roi_y + roi_size - 1 then
```

```
278         lastPixel <= '1';
279       else
280         lastPixel <= '0';
281       end if;
282     end if;
283
284   end process detectPixel;
285
286   ------------------------------------------------------------------------------
287   -- Process name : accumulate_proc
288   -- Description : accumulate values when current pixel is in scope
289   ------------------------------------------------------------------------------
290   accumulate_proc : process(accumulate, start_of_frame)
291   begin
292     -- current pixel value is valid only if the flag is raised
293     if accumulate = '1' then
294       accumulator <= accumulator + currentPixelValue;
295
296     -- reset accumulator on new frame
297     elsif start_of_frame = '1' then
298       accumulator <= (others => '0');
299     end if;
300
301   end process accumulate_proc;
302
303   ------------------------------------------------------------------------------
304   -- Process name : checkValue
305   -- Description : confront accumulator when all ROI pixel's values are known
306   ------------------------------------------------------------------------------
307   checkValue : process(pixelInRoiFound, currentPixelValue, lastPixel)
308   begin
309
310     if lastPixel = '1' then
311       if accumulator < roi_threshold then
312         result <= '1';
313
314       else
315         result <= '0';
316       end if;
317     end if;
318   end process checkValue;
319
320 END ARCHITECTURE behaviour;
```

Listing 8: Entity and architecture of compute ROI

D.7 Accumulator

```
--
-- VHDL Architecture ImageProcessing.accumulator.behaviour
--
-- Created:
--          by - bachelor.bachelor (bachelor-vm)
--          at - 13:04:23 08/08/19
--
-- using Mentor Graphics HDL Designer(TM) 2018.1 (Build 12)
--
LIBRARY ieee;
  USE ieee.std_logic_1164.all;
  USE ieee.numeric_std.ALL;

LIBRARY ImageProcessing;
  USE ImageProcessing.general.ALL;

ENTITY accumulator IS
  GENERIC(
    constant dataBitNb: positive := 16;
    constant paramBitNb: positive := 16;
    constant grayscaleBitNb: positive := 10;
    constant nbrInputData: positive := 8
  );
   PORT(
      endOfProcessing : IN     std_logic;
      newPixels       : IN     std_ulogic;
      roi_id          : IN     unsigned (dataBitNb-1 DOWNTO 0);
      roi_mode        : IN     std_ulogic;
      roi_threshold   : IN     unsigned (paramBitNb-1 DOWNTO 0);
      start_of_frame  : IN     std_ulogic;
      value           : IN     data_array;
      id              : OUT    unsigned (dataBitNb-1 DOWNTO 0);
      result          : OUT    std_ulogic;
      valid           : OUT    std_ulogic
   );

-- Declarations
END accumulator ;

--
ARCHITECTURE behaviour OF accumulator IS
  -- the maximal value is defined as roi_size^2 * grayscale max value
  signal accumulatedValue: unsigned(grayscaleBitNb + 6 - 1 downto 0);
  signal sResult: std_ulogic;
  signal sValid: std_ulogic;
BEGIN
  ----------------------------------------------------------------------------
  -- Process name : accumulator
  -- Description : accumulate the actual values from pixels
  ----------------------------------------------------------------------------
  accumulator : process(newPixels, start_of_frame, value)
    -- used to accumulate vales before giving the value to the nex process
    variable internalAccumulator: unsigned(accumulatedValue'range) := (others => '0');
  begin
    -- reset the accumulator on new frame
    if start_of_frame = '1' then
      internalAccumulator := (others => '0');

    -- each time a new pixel is detected, accumulate bufferised values
    elsif newPixels = '1' then
      for i in 0 to nbrInputData-1 loop
        internalAccumulator := internalAccumulator + value(i);
      end loop;
    else
    end if;

    -- write on output
    accumulatedValue <= internalAccumulator;
```

```
69  end process accumulator;
70
71  ------------------------------------------------------------------------------
72  -- Process name : checkThreshold
73  -- Description : confront accuulator against given threshold
74  ------------------------------------------------------------------------------
75  checkThreshold : process(endOfProcessing, accumulatedValue, roi_mode,
76    roi_threshold)
77  begin
78    if endOfProcessing = '1' then
79      if (accumulatedValue < roi_threshold and roi_mode = '0') or
80        (accumulatedValue > roi_threshold and roi_mode = '1') then
81        sResult <= '1';
82        sValid <= '1';
83      else
84        sResult <= '0';
85        sValid <= '1';
86      end if;
87    else
88      sResult <= '0';
89      sValid <= '0';
90    end if;
91  end process checkThreshold;
92
93  -- outputs
94  id <= roi_id;
95  result <= sResult;
96  valid <= sValid;
97  END ARCHITECTURE behaviour;
```

Listing 9: Entity and architecture of accumulator

D.8 Pixels counter

```
--
-- VHDL Architecture ImageProcessing.pixelsCounter.behaviour
--
-- Created:
--          by - bachelor.bachelor (bachelor-vm)
--          at - 11:24:22 08/05/19
--
-- using Mentor Graphics HDL Designer(TM) 2018.1 (Build 12)
--
LIBRARY ieee;
  USE ieee.std_logic_1164.all;
  USE ieee.numeric_std.ALL;

ENTITY pixelsCounter IS
  GENERIC(
    constant dataBitNb: positive := 16;
    constant dataDuration: positive := 8;
    constant nbrInputData: positive := 8
  );
   PORT(
      clock          : IN     std_ulogic;
      en             : IN     std_ulogic;
      receiving      : IN     std_logic;
      reset          : IN     std_ulogic;
      start_of_frame : IN     std_ulogic;
      start_of_line  : IN     std_ulogic;
      newPixels      : OUT    std_ulogic;
      pixelCounter_x : OUT    unsigned (dataBitNb-1 DOWNTO 0);
      pixelCounter_y : OUT    unsigned (dataBitNb-1 DOWNTO 0)
   );

-- Declarations

END pixelsCounter ;

--
ARCHITECTURE behaviour OF pixelsCounter IS
-- counters
signal counter: unsigned(dataBitNb-1 downto 0);
signal sPixelCounter_x: unsigned(pixelCounter_x'range); -- count pixels in X
signal sPixelCounter_y: unsigned(pixelCounter_y'range); -- count pixels in Y
signal pixelCounter_x_old: unsigned(pixelCounter_x'range);

-- flags
signal sNewPixels: std_ulogic;
BEGIN

-------------------------------------------------------------------------------
  -- Process name : countInputFlow
  -- Description : counts incomming bit flow for synchronization
  -----------------------------------------------------------------------------
  countInputFlow : process(clock, reset)
  begin
    if reset = '1' then
      counter <= (others => '0');

    elsif rising_edge(clock) then
      if receiving = '1' and counter < dataDuration - 1 then
        counter <= counter + 1;
      else
        counter <= (others => '0');
      end if;
    end if;
  end process countInputFlow;

  -----------------------------------------------------------------------------
  -- Process name : countPixelX
  -- Description : count pixels x to further know if the actual pixel is in ROI
```

```
69  ------------------------------------------------------------------------------
70  countPixelX : process(counter, start_of_frame, start_of_line, receiving)
71  begin
72    -- reset counters in case of a reset, a new frame incoming or sync to SOL
73    if start_of_frame = '1' or start_of_line = '1' or receiving = '0' then
74      sPixelCounter_x <= (others => '0');
75
76    -- triggering on half duration allow to read value time-centered
77    elsif counter = dataDuration/2 then
78      -- increment x counter by nbrInputData as pixels come 1 per input line
79      sPixelCounter_x <= sPixelCounter_x + nbrInputData;
80    end if;
81  end process countPixelX;
82
83  ------------------------------------------------------------------------------
84  -- Process name : countPixelY
85  -- Description : count pixels y to further know if the actual pixel is in ROI
86  ------------------------------------------------------------------------------
87  countPixelY : process(start_of_frame, start_of_line, receiving)
88  begin
89    -- reset counters in case of a reset or a new frame incoming
90    if start_of_frame = '1' or receiving = '0' then
91      sPixelCounter_y <= (others => '0');
92
93    -- if a new line occurs, increment y-counter and reset x-counter
94    elsif start_of_line = '1' then
95      sPixelCounter_y <= sPixelCounter_y + 1;
96    end if;
97  end process countPixelY;
98
99  ------------------------------------------------------------------------------
100 -- Process name : detectPixel
101 -- Description : sync process which check if new pixels are ready
102 ------------------------------------------------------------------------------
103 detectPixel : process(reset, clock)
104 begin
105   if reset = '1' or start_of_line = '1' or start_of_frame = '1' then
106     sNewPixels <= '0';
107     pixelCounter_x_old <= (others => '0');
108
109   elsif rising_edge(clock) then
110     -- if new pixels are incomming, indicate it
111     if pixelCounter_x_old /= pixelCounter_x then
112       sNewPixels <= '1';
113       pixelCounter_x_old <= pixelCounter_x;
114
115     -- reset flag
116     else
117       sNewPixels <= '0';
118     end if;
119   end if;
120
121 end process detectPixel;
122
123 -- output signals
124 newPixels <= sNewPixels;
125 pixelCounter_x <= sPixelCounter_x;
126 pixelCounter_y <= sPixelCounter_y;
127
128 END ARCHITECTURE behaviour;
```

Listing 10: Entity and architecture of pixel counter

D.9 Pixel tracker

```
--
-- VHDL Architecture ImageProcessing.roiTracker.behaviour
--
-- Created:
--          by - bachelor.bachelor (bachelor-vm)
--          at - 15:09:46 08/05/19
--
-- using Mentor Graphics HDL Designer(TM) 2018.1 (Build 12)
--
LIBRARY ieee;
  USE ieee.std_logic_1164.all;
  USE ieee.numeric_std.ALL;

ENTITY pixelTracker IS
  GENERIC(
    constant dataBitNb: positive := 16;
    constant paramBitNb: positive := 16
  );
   PORT(
      data_n          : IN     unsigned (dataBitNb-1 DOWNTO 0);
      inRoi           : IN     std_ulogic;
      newPixels       : IN     std_ulogic;
      offset_x        : IN     unsigned (paramBitNb-1 DOWNTO 0);
      overlap         : IN     signed (paramBitNb-1 DOWNTO 0);
      roi_size        : IN     unsigned (paramBitNb-1 DOWNTO 0);
      trackerId       : IN     unsigned (paramBitNb-1 DOWNTO 0);
      value_n         : OUT    unsigned (dataBitNb-1 DOWNTO 0)
   );

-- Declarations

END pixelTracker ;

--
ARCHITECTURE behaviour OF pixelTracker IS
  -- internals
  signal sValue: unsigned(value_n'range);

  -- flags
  signal lastPixel: std_uLogic;

BEGIN
  ----------------------------------------------------------------------------
  -- Process name : detectPixel
  -- Description : process which check if actual pixel is in ROI
  ----------------------------------------------------------------------------
  detectPixel : process(newPixels, inRoi, offset_x, trackerId, roi_size,
    data_n, overlap)
  begin
    -- reset all between pixels
    if newPixels = '0' then
      sValue <= (others => '0');

    -- do this only inside a ROI
    elsif newPixels = '1' and inRoi = '1' then
      -- ensure a positive result as otherwise it is not a interesting condition
      if trackerId >= offset_x and overlap = 0 then
        -- test if the watched pixel could be in ROI
        if trackerId - offset_x < roi_size then
          sValue <= data_n;
        else
        end if;

      elsif overlap /= 0 then
        if overlap > signed(trackerId) then
          sValue <= data_n;
        else
          sValue <= (others => '0');
```

```
69          end if;
70
71        else
72          sValue <= (others => '0');
73        end if;
74
75      else
76        sValue <= (others => '0');
77      end if;
78
79    end process detectPixel;
80
81    -- outputs
82    value_n <= sValue;
83  END ARCHITECTURE behaviour;
```

Listing 11: Entity and architecture of pixel tracker

D.10 ROI tracker

```
--
-- VHDL Architecture ImageProcessing.roiTracker.behaviour
--
-- Created:
--          by - bachelor.bachelor (bachelor-vm)
--          at - 15:09:46 08/05/19
--
-- using Mentor Graphics HDL Designer(TM) 2018.1 (Build 12)
--
LIBRARY ieee;
  USE ieee.std_logic_1164.all;
  USE ieee.numeric_std.ALL;

ENTITY roiTracker IS
  GENERIC(
    constant dataBitNb: positive := 16;
    constant paramBitNb: positive := 16;
    constant nbrInputData: positive := 8
  );
   PORT(
      newPixels      : IN     std_ulogic;
      pixelCounter_x : IN     unsigned (dataBitNb-1 DOWNTO 0);
      pixelCounter_y : IN     unsigned (dataBitNb-1 DOWNTO 0);
      roi_size       : IN     unsigned (paramBitNb-1 DOWNTO 0);
      roi_x          : IN     unsigned (paramBitNb-1 DOWNTO 0);
      roi_y          : IN     unsigned (paramBitNb-1 DOWNTO 0);
      start_of_frame : IN     std_ulogic;
      start_of_line  : IN     std_ulogic;
      inRoi          : OUT    std_ulogic;
      lastPixel      : OUT    std_ulogic;
      offset_x       : OUT    unsigned (paramBitNb-1 DOWNTO 0);
      overlap        : OUT    signed (paramBitNb-1 DOWNTO 0)
   );

-- Declarations

END roiTracker ;

--
ARCHITECTURE behaviour OF roiTracker IS
  -- internals
  signal sOffset_x: unsigned(offset_x'range);
  signal sOffset_y: unsigned(offset_x'range);
  signal sOverlap: signed(overlap'range);
  signal sLastPixel: std_ulogic;
BEGIN
  ----------------------------------------------------------------------------
  -- Process name : computeOffsets
  -- Description : process which check if actual pixels contains ROI beginning
  ----------------------------------------------------------------------------
  computeOffsets : process(newPixels, start_of_frame, start_of_line, roi_x,
    roi_y, roi_size, pixelCounter_x, pixelCounter_y, inRoi)
  begin
    -- reset all between pixels
    if start_of_frame = '1' then
      sOffset_x <= (others => '0');
      sOffset_y <= (others => '0');

    elsif inRoi = '1' then
      -- if current line match to a ROI, check further coordinates
      if roi_y + sOffset_y = pixelCounter_y-1 then
        -- set offset_x depending of where the ROI is in the 8 current pixels
        for i in 0 to nbrInputData-1 loop
          -- check which pixel correspond to the one in the tracked data line
          -- minus 1 to pixelCounter_x as it counts pixels 8 by 8!
          if roi_x = pixelCounter_x - 1 - i then
            -- assign the offset, counted from maximal down to 0
            sOffset_x <= to_unsigned((nbrInputData-1) - i, sOffset_x'length);
```

```
69          else
70          end if;
71        end loop;
72      else
73      end if;
74
75    -- on a new line, check if the y offset needs an increment
76    elsif start_of_line = '1' then
77      -- minus 2 as counter y start at 1
78      if roi_y + sOffset_y = pixelCounter_y-2 and sOffset_y < roi_size-1 then
79        sOffset_y <= sOffset_y + 1;
80      else
81      end if;
82    else
83    end if;
84  end process computeOffsets;
85
86  ------------------------------------------------------------------------------
87  -- Process name : detectRoi
88  -- Description : process which check if current pixels are in ROI
89  ------------------------------------------------------------------------------
90  detectRoi : process(pixelCounter_x, pixelCounter_y, roi_x, roi_y, roi_size, overlap)
91  begin
92    -- check if current pixels are in the current line
93    if pixelCounter_y-1 >= roi_y and pixelCounter_y-1 < roi_y + roi_size then
94      -- if yes, check wich batch contains ROI pixels
95      if pixelCounter_x > roi_x and pixelCounter_x - (nbrInputData-1) <= roi_x then
96        inRoi <= '1';
97
98      elsif pixelCounter_x - (nbrInputData) = roi_x then
99        inRoi <= '1';
100
101     -- if an overlap occurs
102     elsif overlap /= 0 then
103       inRoi <= '1';
104
105     -- if not in ROI, reset flag
106     else
107       inRoi <= '0';
108     end if;
109
110   -- default, reset flags
111   else
112     inRoi <= '0';
113   end if;
114 end process detectRoi;
115
116 ------------------------------------------------------------------------------
117 -- Process name : detectRoiEnd
118 -- Description : process which check if all ROI pixels are found
119 ------------------------------------------------------------------------------
120 detectRoiEnd : process(newPixels, start_of_frame, pixelCounter_x,
121   pixelCounter_y, roi_x, roi_y, roi_size)
122 begin
123   -- reset all between pixels
124   if start_of_frame = '1' then
125     sLastPixel <= '0';
126
127   elsif newPixels = '1' then
128     -- if x-counter is bigger than roi position, means that roi is over
129     if pixelCounter_x > roi_x + roi_size and pixelCounter_y = roi_y + roi_size then
130       sLastPixel <= '1';
131     end if;
132
133   else
134     sLastPixel <= '0';
135   end if;
136
137 end process detectRoiEnd;
138
```

```
139  ------------------------------------------------------------------------------
140  -- Process name : detectOverlap
141  -- Description : check if the ROI is between two consecutives pixels rows
142  ------------------------------------------------------------------------------
143   detectOverlap : process(newPixels, pixelCounter_x, pixelCounter_y, roi_x, roi_size, offset_x
       )
144   begin
145     if newPixels = '1' then
146       -- do the check only when a line contains ROI
147       if pixelCounter_y-1 >= roi_y and pixelCounter_y-1 < roi_y + roi_size then
148         -- occurs only if offset is not null, then test the next batch of pixels !!!!! WARNING
      could lead to problems when near borders
149         if offset_x /= 0 and pixelCounter_x - (nbrInputData-1) >= roi_x and pixelCounter_x -
      2*(nbrInputData-1) <= roi_x then
150           sOverlap <= signed(roi_size - (nbrInputData - offset_x));
151         else
152           sOverlap <= (others => '0');
153         end if;
154       else
155         sOverlap <= (others => '0');
156       end if;
157     else
158       sOverlap <= (others => '0');
159     end if;
160   end process detectOverlap;
161
162   -- outputs
163   offset_x <= sOffset_x;
164   overlap <= sOverlap;
165   lastPixel <= sLastPixel;
166 END ARCHITECTURE behaviour;
```

Listing 12: Entity and architecture of ROI tracker

D.11 State machine

```
--
-- VHDL Architecture ImageProcessing.stateMachine.behaviour
--
-- Created:
--          by - bachelor.bachelor (bachelor-vm)
--          at - 13:44:34 08/05/19
--
-- using Mentor Graphics HDL Designer(TM) 2018.1 (Build 12)
--
LIBRARY ieee;
  USE ieee.std_logic_1164.all;
  USE ieee.numeric_std.ALL;

ENTITY stateMachine IS
   PORT(
      clock           : IN     std_ulogic;
      en              : IN     std_ulogic;
      lastPixel       : IN     std_ulogic;
      reset           : IN     std_ulogic;
      start_of_frame  : IN     std_ulogic;
      endOfProcessing : OUT    std_logic;
      receiving       : OUT    std_logic
   );

-- Declarations

END stateMachine ;

--
ARCHITECTURE behaviour OF stateMachine IS
  -------------------------------------------------------------------
  -- State Machine
  -------------------------------------------------------------------
  -- possible states
  type STATE_TYPE is (
    idle,
    search_pixels,
    finished
  );

  -- Declare current and next state signals
  signal current_state : STATE_TYPE;
  signal next_state : STATE_TYPE;

BEGIN
  -------------------------------------------------------------------
  -- State Machine
  -------------------------------------------------------------------
  -- Process name : clocked_proc
  -- Description : state's flip-flop handling
  -------------------------------------------------------------------
  clocked_proc : process(clock, reset)
  begin
    if reset = '1' then
      current_state <= idle;
    elsif rising_edge(clock) then
      current_state <= next_state;
    end if;
  end process clocked_proc;

  -------------------------------------------------------------------
  -- Process name : nextstate_proc
  -- Description : determine the next state according to :
  --  1) actual state
  --  2) incomming events
  -------------------------------------------------------------------
  nextstate_proc : process(current_state, start_of_frame, lastPixel)
  begin
```

```
    case current_state is
      -- when a new frame occurs, begin the analysis
      when idle =>
        if start_of_frame then
          next_state <= search_pixels;
        end if;

      -- search is only ended by the arriving of a new frame
      when search_pixels =>
        if lastPixel then
          next_state <= finished;
        end if;

      when finished =>
        next_state <= idle;

      when others =>
        next_state <= idle;
    end case;
  end process nextstate_proc;

  ------------------------------------------------------------------------
  -- Process name : output_in_proc
  -- Description : action on state entry
  ------------------------------------------------------------------------
  output_in_proc : process(current_state)
  begin
   -- Default Assignment
   endOfProcessing <= '0';

   -- Combined Actions
   case current_state is
    when idle =>
      -- do nothing

    -- search desired pixels by starting counters
    when search_pixels =>
      receiving <= '1';

    -- indicate to output that output data are valids
    when finished =>
      receiving <= '0';
      endOfProcessing <= '1';
    when others =>
      null;
   end case;
  end process output_in_proc;

END ARCHITECTURE behaviour;
```

Listing 13: Entity and architecture of state-machine

D.12 General package

```
1  --
2  -- VHDL Package Header ImageProcessing.general
3  --
4  -- Created:
5  --          by - bachelor.bachelor (bachelor-vm)
6  --          at - 16:00:08 08/08/19
7  --
8  -- using Mentor Graphics HDL Designer(TM) 2018.1 (Build 12)
9  --
10 LIBRARY ieee;
11 USE ieee.std_logic_1164.all;
12 USE ieee.numeric_std.all;
13
14 PACKAGE general IS
15   constant dataBitNb: positive := 16;
16   constant paramBitNb: positive := 16;
17   constant nbrInputData: positive := 8;
18
19   type data_array is array (nbrInputData-1 downto 0) of unsigned (dataBitNb-1 downto 0);
20   type param_array is array (nbrInputData-1 downto 0) of unsigned (paramBitNb-1 downto 0);
21 END general;
```

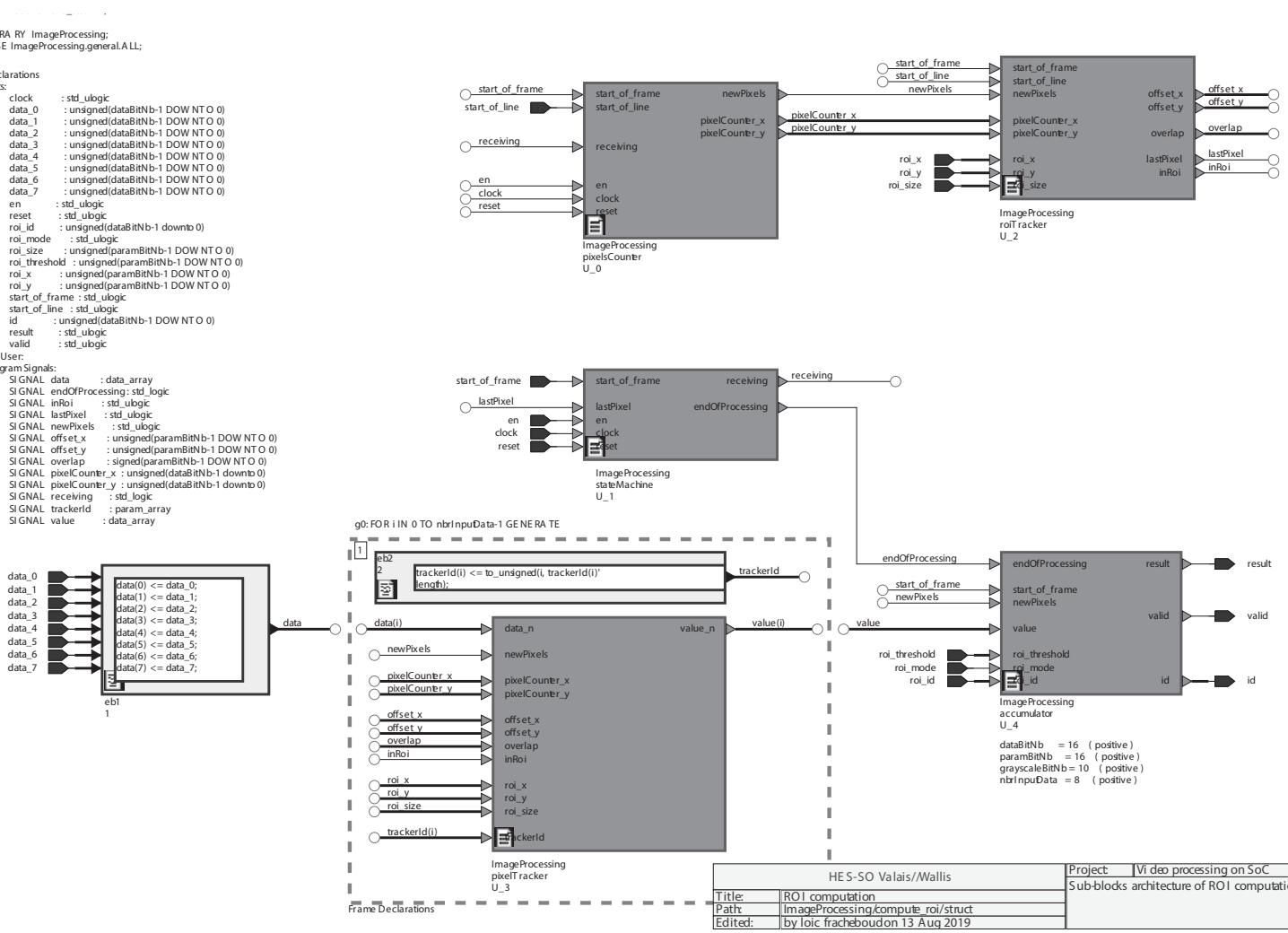
Listing 14: Package written to define some type and constants

E Schematics

Here stands the three main schematics.

```
LIBRARY ImageProcessing;
  USE ImageProcessing.general.ALL;

Declarations
Ports:
        clock          : std_ulogic
        data_0         : unsigned(dataBitNb-1 DOWNTO 0)
        data_1         : unsigned(dataBitNb-1 DOWNTO 0)
        data_2         : unsigned(dataBitNb-1 DOWNTO 0)
        data_3         : unsigned(dataBitNb-1 DOWNTO 0)
        data_4         : unsigned(dataBitNb-1 DOWNTO 0)
        data_5         : unsigned(dataBitNb-1 DOWNTO 0)
        data_6         : unsigned(dataBitNb-1 DOWNTO 0)
        data_7         : unsigned(dataBitNb-1 DOWNTO 0)
        en             : std_ulogic
        reset          : std_ulogic
        roi_id         : unsigned(dataBitNb-1 downto 0)
        roi_mode       : std_ulogic
        roi_size       : unsigned(paramBitNb-1 DOWNTO 0)
        roi_threshold  : unsigned(paramBitNb-1 DOWNTO 0)
        roi_x          : unsigned(paramBitNb-1 DOWNTO 0)
        roi_y          : unsigned(paramBitNb-1 DOWNTO 0)
        start_of_frame : std_ulogic
        start_of_line  : std_ulogic
        id             : unsigned(dataBitNb-1 DOWNTO 0)
        result         : std_ulogic
        valid          : std_ulogic
Pre User:
Diagram Signals:
        SIGNAL data          : data_array
        SIGNAL endOfProcessing : std_logic
        SIGNAL inRoi         : std_ulogic
        SIGNAL lastPixel     : std_ulogic
        SIGNAL newPixels     : std_ulogic
        SIGNAL offset_x      : unsigned(paramBitNb-1 DOWNTO 0)
        SIGNAL offset_y      : unsigned(paramBitNb-1 DOWNTO 0)
        SIGNAL overlap       : signed(paramBitNb-1 DOWNTO 0)
        SIGNAL pixelCounter_x : unsigned(dataBitNb-1 downto 0)
        SIGNAL pixelCounter_y : unsigned(dataBitNb-1 downto 0)
        SIGNAL receiving     : std_logic
        SIGNAL trackerId     : param_array
        SIGNAL value         : data_array
```



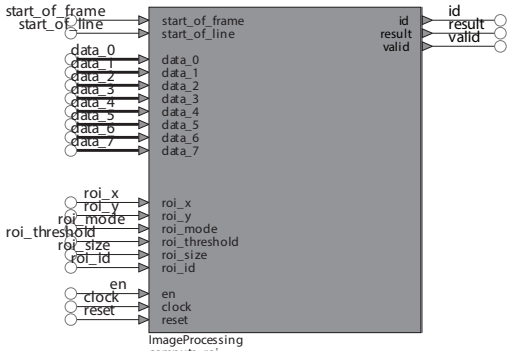
HES-SO Valais//Wallis		Project:	Video processing on SoC
Title:	ROI computation	Sub-blocks architecture of ROI computation	
Path:	ImageProcessing/compute_roi/struct		
Edited:	by loic fracheboud on 13 Aug 2019		

ImageProcessing_test/loopback_tb/struct

Package List
LIBRARY ieee;
 USE ieee.std_logic_1164.all;
 USE ieee.numeric_std.ALL;

LIBRARY ImageProcessing;
 USE ImageProcessing.general.ALL;

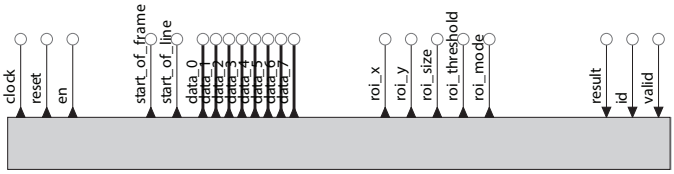
Declarations
Ports:
Pre User:
```
    constant dataBitNb: positive := 16;
    constant paramBitNb: positive := 16;
    constant imageWidth: positive := 640;
    constant imageHeigth: positive := 480;
    constant grayscaleBitNb: positive := 10;
    constant grayscaleMax: positive := 1024;
    constant grayscaleStep: positive := 1;
    constant dataDuration: positive := 8;
    constant nbrInputData: positive := 8;
```
Diagram Signals:
```
    SIGNAL clock        : std_ulogic
    SIGNAL data_0       : unsigned(dataBitNb-1 DOWNTO 0)
    SIGNAL data_1       : unsigned(dataBitNb-1 DOWNTO 0)
    SIGNAL data_2       : unsigned(dataBitNb-1 DOWNTO 0)
    SIGNAL data_3       : unsigned(dataBitNb-1 DOWNTO 0)
    SIGNAL data_4       : unsigned(dataBitNb-1 DOWNTO 0)
    SIGNAL data_5       : unsigned(dataBitNb-1 DOWNTO 0)
    SIGNAL data_6       : unsigned(dataBitNb-1 DOWNTO 0)
    SIGNAL data_7       : unsigned(dataBitNb-1 DOWNTO 0)
    SIGNAL en           : std_ulogic
    SIGNAL id           : unsigned(dataBitNb-1 DOWNTO 0)
    SIGNAL reset        : std_ulogic
    SIGNAL result       : std_ulogic
    SIGNAL roi_id       : unsigned(dataBitNb-1 downto 0)
    SIGNAL roi_mode     : std_ulogic
    SIGNAL roi_size     : unsigned(paramBitNb-1 DOWNTO 0)
    SIGNAL roi_threshold : unsigned(paramBitNb-1 DOWNTO 0)
    SIGNAL roi_x        : unsigned(paramBitNb-1 DOWNTO 0)
    SIGNAL roi_y        : unsigned(paramBitNb-1 DOWNTO 0)
    SIGNAL start_of_frame: std_ulogic
    SIGNAL start_of_line : std_ulogic
    SIGNAL valid        : std_ulogic
```



ImageProcessing
compute_roi
I0

```
dataBitNb     = 16   ( positive )
paramBitNb    = 16   ( positive )
grayscaleBitNb = 10  ( positive )
dataDuration  = 8    ( positive )
nbrInputData  = 8    ( positive )
```

```
dataBitNb    = 16   ( positive )
paramBitNb   = 16   ( positive )
imageWidth   = 640  ( positive )
imageHeigth  = 480  ( positive )
grayscaleStep = 1   ( positive )
dataDuration = 8    ( positive )
nbrInputData = 8    ( positive )
```



ImageProcessing_test
loopback_tester
I1

HES-SO Valais//Wallis		Project:	Video processing on SoC
		Perform global tests on the block	
Title:	ROI computation test bench		
Path:	<<-- more -->>		
Edited:	by loic fracheboud on 13 Aug 2019		

ImageProcessing_test/accumulator_tb/struct

Package List

```
LIBRARY ieee;
  USE ieee.std_logic_1164.all;
  USE ieee.numeric_std.ALL;

LIBRARY ImageProcessing;
  USE ImageProcessing.general.ALL;
```

Declarations

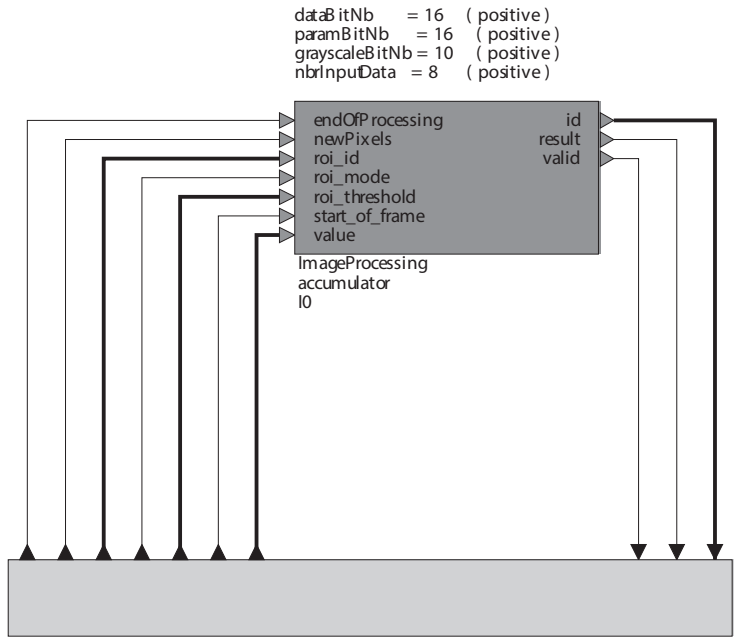
Ports:

Pre User:

```
    constant dataBitNb: positive := 16;
    constant paramBitNb: positive := 16;
    constant grayscaleBitNb: positive := 10;
    constant nbrInputData: positive := 8;
    constant maxRoiSize: positive := 8;
```

Diagram Signals:

```
    SIGNAL endOfProcessing : std_logic
    SIGNAL id              : unsigned(dataBitNb-1 DOWNTO 0)
    SIGNAL newPixels       : std_ulogic
    SIGNAL result          : std_ulogic
    SIGNAL roi_id          : unsigned(dataBitNb-1 DOWNTO 0)
    SIGNAL roi_mode        : std_ulogic
    SIGNAL roi_threshold   : unsigned(paramBitNb-1 DOWNTO 0)
    SIGNAL start_of_frame  : std_ulogic
    SIGNAL valid           : std_ulogic
    SIGNAL value           : data_array
```



HES-SO Valais//Wallis		Project:	Video processing on SoC
Title:	Accumulator test bench	Perform unit test on accumulator, which is a part of ROI computation block	
Path:	<<-- more -->>		
Edited:	by loic fracheboud on 13 Aug 2019		