Indoor positioning system

1. Members

- Henry Rodrigues da Silva
- Rodrigo Travessini

2. About the project

The project aims to develop a solution for mapping the localization people in a room using a video camera. It will be capable of identify the relative position of these people in the room based on the camera’s location, because of this, the camera has to be in a position where it has maximum visibility of the room. The system connected to the camera has to receive the frames captured by the camera and provide a set of coordinates indicating the position of each person in the room.

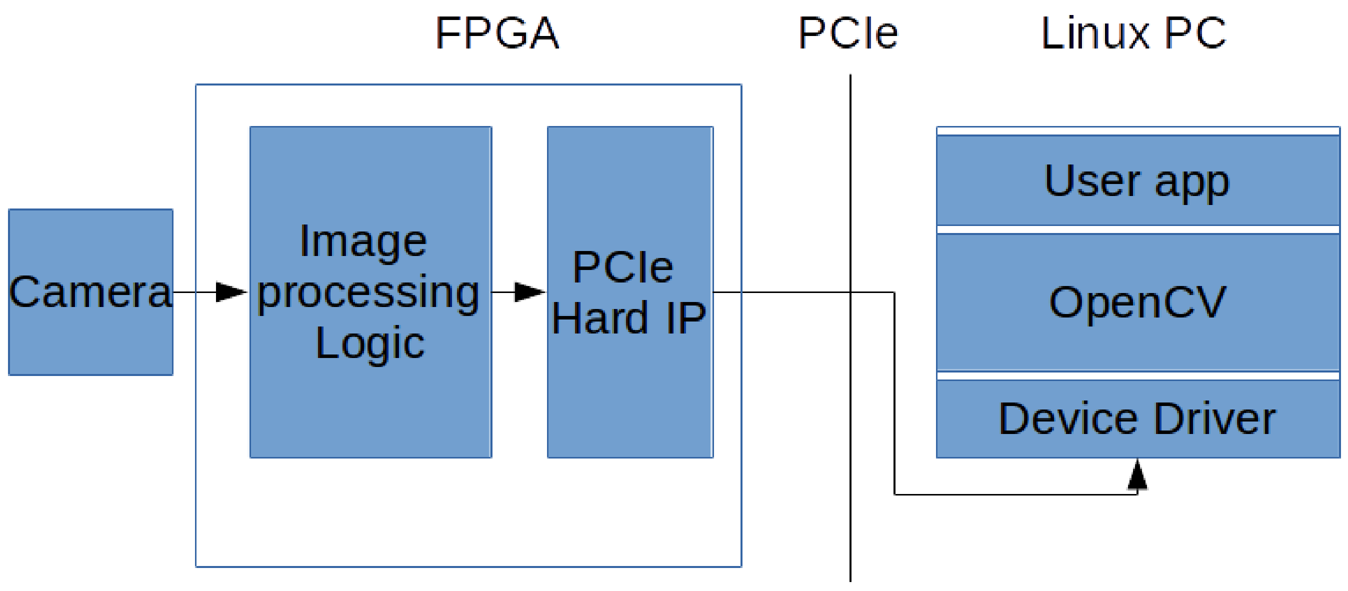
The project will be composed by both the development of hardware and software. The hardware part will be necessary to improve the performance of the image processing component of this project, when compared to a solution entirely in software. An ideal platform for a first prototype should be an FPGA, where through the use of a hardware description language (HDL), a digital system to perform the hardware part can be synthesized, and also a softcore can be used to run the software. Another option would be the use of a development board with both an FPGA and a general purpose processor, possibly improving the overall performance.

Due to the laws and regulations of several countries, this project has to guarantee that the images captured by the camera will not be utilized for other porpouses. One of the option is try to connect the digital system developed as close as possible to the camera’s CCD(Charge-coupled device), thus allowing a safer image processing and eliminating the risk of images being “leaked” to other systems and malicious softwares.

3. Motivation

In the building automation area is of great interest to have a way to identify how many people are inside a room and also to determine their position at each instant. With this kind of information a system may be able to improve the configuration of cooling systems such as air conditioners, possibly also improving the energy efficiency of those systems.

4. Model



5. Non functional model

5.1. Performance Requirements

The frames captured by the camera can be represented in the following resolution and frame rate:

- *Configuration A - 1920x1080 - 10 fps*
- *Configuration B - 1280x720 - 10 fps*
- *Configuration C - 640x480 - 10 fps*

Many algorithms used in the process of movement detection perform its actions in a pixel-base way. The algorithms usually make use of two nested loops, one traversing the lines and the other the colums. This way the total number of iterations executed for each frame is proportional to the resolution. Processing the frames in real time requires that the time processing one frame should not be higher than the interval between consecutive captured frames. Based on this information, it is possible to calculate how many iterations must be executed per second in each configuration provided by the camera:

- Configuration A - 20.736.000 Iterations per second
- Configuration B - 9.216.000 Iterations per second
- Configuration C - 3.072.000 Iterations per second

In a sequential harware, with no paralelism, where each iteration is executed in a single clock, would result in clock rates between 5Mhz and 23Mhz. This simple model is unattainable. The movement detection is a complex process, and the entire chain of filters applied will probably impose a long critical path, not allowing these clock rates. Besides that, some of the filters must wait the previous filter be applied to the entire frame, making necessary separated loops. This way, a good way to approach the problem is through the use of pipeline, isolating different filters in different pipeline stages.

5.2. Memory Requirements

The memory needed is highly influenced by the resolution being used and the amount of frames that must be in memory simultaneously. Other factor, is that some frames in memory must be in the RGB representation, others only the luma component, or even only a binary representation (e.g. masks). Since most of the time the luma is the prefered representation, it is possible to suppose that is necessary a byte to represent each pixel.

- Configuration A - 2MB per frame in memory
- Configuration B - 0.9MB per frame in memory
- Configuration C - 0.3MB per frame in memory

6. Hardware Modules

- FPGA development board
- Camera module (preferably a stereo camera or 2 cams)

7. Algorithm overview

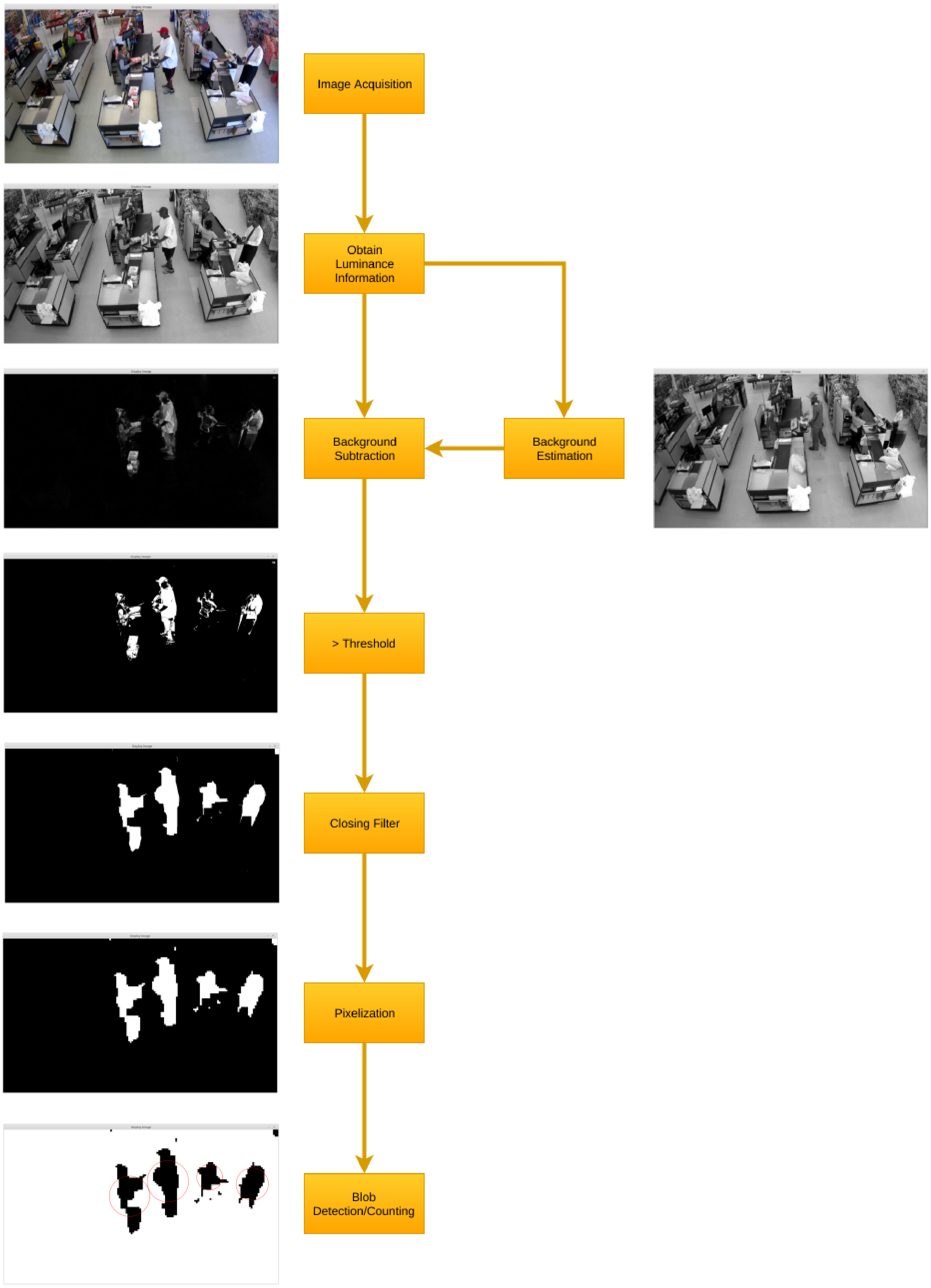


Image Acquisition
Obtain Luminance Information
Background Subtraction
Background Estimation
> Threshold
Closing Filter
Pixelization
Blob Detection/Counting

7.1. Obtain Luminance Information

Video sequence is captured by the camera in the RGB color space. The problem with the RGB color space is that it has great sensitivity to sensor noise and changes of lighting conditions. Instead of using the RGB information, it is preferred to only use the luminance component during the algorithm. The formula to obtain the luma component from RGB information is shown below:

$$Y' = 0.299\,R' + 0.587\,G' + 0.114\,B'$$

7.2. Background Estimation

For a very simple algorithm, the background could be chosen as the FIRST FRAME obtained by the camera.

PROBLEM 1 - The first frame should only have static objects.
PROBLEM 2 - Static objects in posterior frames would be recognized as movement.

Because of those two problems, it is necessary that the background be continuously updated. One possible way to perform this update, is through the use of a moving average filter. Each frame processed is added to the background with a small weight. This way non static objects present in the first frame will be removed from the background a few frames later, and static objects added in posterior frames are recognized as being part of the background. The moving average formula is presented below:

$$new\_background = \alpha * old\_background + (1 - \alpha) * current\_frame$$

$$0 < \alpha < 1$$

PROBLEM 3 - Movement pixels that shouldn't be in the background are still being added during the moving average filter.

To improve this, it is necessary to find a logic that respects the following rule:

$$new\_background = \alpha * old\_background + (1 - \alpha) * current\_frame \text{ (FOR STATIC PIXELS)}$$

$$new\_background = old\_background \text{ (FOR MOVEMENT PIXELS)}$$

This way, only the pixels that really do not represent any kind of movement are in the background. To achive this goal, this project use the following logic:

1. Create a matrix and store the pixel by pixel absolute difference between the current frame and the frame before (t - 1), let's call this ABS_1. Do the same for the current frame and the frame (t - 2), ABS_2.
2. For both matrix, obtain the mean (MEAN_1 and MEAN_2) and standard deviation (STD_1 and STD_2) between the differences.
3. Having this information, we can say that a pixel (i,j) is only considered static if both ABS_1(i,j) < MEAN_1 + STD_1 and ABS_2(i,j) < MEAN_2 + STD_2 are true.

7.3. Background Subtraction

During this step, the algorithm simply subtract the current frame (version with only luma information), from the estimated background. This way it is possible to identify the differences between the current frame and the background.

7.4. Threshold

Since there are lots of random noise and light changes on the image recorded by the camera, only the significant pixel differences must be kept for the remaining of the algorithm. Besides that, we are not interested in the exact difference value, but only the information regarding the pixel having an significant difference or not, making a binary image enough to represent.

7.5. Closing Filter

Closing is an image processing operation used to remove small holes from the objects in the image. In our project, it is really useful because the movement pixels usually are only located in the object boundary, and though the use of this filter, we can fill the object interior, so that it can be more easily recognized/detected in the next steps.

7.6. Pixelization

In this step we scale down the resolution of the frame. This step is useful to finish removing small holes in the objects, and by having a smaller resolution, the next steps of the algorithm can run faster.

7.7. Blob Counting

The final step of the algorithm consist in detecting each individual object in the frame, and identify its center. Since this step requires some complex algorithms, we are using the Blob Detector provided by the OpenCV Library.

8. Dataflow Model of Computation

In the dataflow model, the decision as to when to react can reside with each individual actor, and is less centralized. Reactions are dependent on one another, the dependence is due to the flow of data. If a reaction of actor A requires data produced by a reaction of actor B, then the reaction of A must occur after the reaction of B.

In dataflow models, the signals providing communication between the actors are sequences of message, where each message is called a token. Each actor in the dataflow model has a firing rule specifying the number of tokens required on each input port in order to fire the actor.

Since the firing of the actors is asynchronous, a token sent from one actor to another must be buffered; it needs to be saved until the destination actor is ready to consume it. Since one of the actors may fire on a higher rate, there must be a way to guarantee that the buffer will not overflow (one common problem in the dataflow model). Another common problem arises when there are feedback networks in the model, generating possible deadlocks, one actor in the loop has insufficient tokens to fire, but it will not receive any more token unless it fires because of the loop.

8.1. Synchronous Dataflow (SDF)

SDF is a constrained form of dataflow where for each actor, every firing consumes a fixed number of input tokens on each input port and produces a fixed number of output tokens on each output port.

Considering two actor A and B with a single connection between them. The output of A is connected to the input of B. A fires at a rate qA, and B at a rate qB. A produces M tokens on each firing, and B consume N tokens. This way, a balance can be achieved by:

$$qA*N = qB*M$$

If this equation can be satisfied, it is possible to find a schedule that guarantees with a bounded buffer between the actors an unbounded execution.

8.2. Dataflow Model in our Project

The indoor positioning system can be described as a set of filters and manipulations applied on the frames received by the camera, in a way that in the end, the result is a matrix showing the locations of each person in the room.

In this way, each of this filters/manipulations applied in the frames can be seen as a separate actor. The idea on the project is to have all those actors implemented in hardware through the use of hardware description language. Since we are working with image frames, a matrix, we can define as one token, the value of one matrix position (eg. one pixel color).

In the current status of the project, we already can see, that some of the image filters (actors), can work on each received token (pixel), and others need to have one entire frame. The same happens to the output. Filters such as the grayscale, react on each received pixel, and produces a grayscale version of that pixel. A blob analysis filter may need a block of pixels (eg.: 8x8, 4x4) and produces a block with corresponding size.

In this sense the dataflow model fits really well in our project. Each actor filter waits the previous filter produce enough data before acting. In our project we also need to have a synchronous dataflow, in a way that if one filter work by pixel, it must operate in a rate number_of_pixels times the rate of an filter that operates on frames.

The only actor in our project that may discard data is our first filter, since not all frames produced by the camera need to be processed for our purpose. If the camera is working at 30fps, we can easily process only 1 every 3-6 frames and still have good results on detecting movement and counting people.

8.3. Problems

The problems faced during the implementation of this model in our project are:

- **Overflow of transport buffers between blocks** - The overflow in buffers can be eliminated by implementing flags that prevents the production of data by the previous block. This flag would be only set by the current block and read by the previous one, so that, while the buffer is full the previous block cannot produce more data. Assuming, of course, that we can safely stall the pipeline without losing a deadline;
- **Frequency of production/consumption of data** - When the blocks are developed only in hardware, the risk of overflow by overproduction or lack of consumption of data can be reduced during the separation of blocks by organizing the components that will compose each block so that the total time required for a block to process the data is not bigger than the time necessary to the previous block to produce it. If the even distribution of time between the blocks is not possible, or for some reason, is necessary to wait until all the previous processing is done, then the buffer inside the current block must be big enough to store all data received until the condition necessary to start its consumption be fulfilled;
- **Deadlocks** - In a parallel environment with feedback, we have the risk of creating a deadlock by letting two parallel components try to access the same resource and being blocked waiting indefinitely by each other, in the Indoor positioning system we have only one situation where this could happen, the background estimation, however our implementation is enough to prevent this. The background estimation doesn't need a copy from any pixel other the one the block is handling, and each block only handles one pixel at the time, so we never have more than one parallel component accessing the same resource thus eliminating the risk of deadlock.

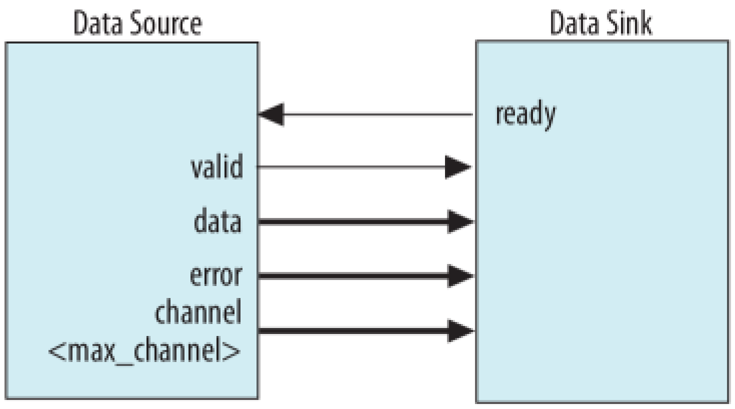
9. FPGA Prototype

Our idea for this project, is to have the initial filters synthesized in an FPGA. Basically we want all the blocks before the closing filter in the FPGA, since they have non complex logic, and can be implemented without the use of OpenCV library.

For this implementation we end up trying three different approaches.

1. Develop all the different IPs for each stage of the algorithm, and develop the way they communicate. The problem with this strategy, is that our IPs would be only useful for this project, and future projects would need to respect the protocol we used in this implementation. Besides that we would be reinventing the wheel, since there are lots of things already done that we could use.
2. In the second approach, we used the software Xilinx HLS, to generate HDL (Verilog or VHDL) from the C++ models. The communication between the blocks would use the protocol AXI 4 Streaming. This is an interesting approach, but since our FPGA is from Altera, we wouldn't be taking advantage of the IPs already provided by Altera for Video and Image Processing, since this library uses a different communication protocol.
3. Finaly we discovered the existence of the Altera Image and Video Processing Suite. This suite already provides IP for connecting with cameras, and others such as a Frame Buffer. Since this suite makes use of the Altera Avalon Streaming Protocol, we had to study and understand how this protocol works.

9.1. Altera Avalon Streaming



The Avalon Streaming protocol is a way to implement the dataflow model of computation. This protocol implements the communication betweent two blocks, one is the producer (Source) and the other is the consumer (Sink). The protocol has three mandatory signals:

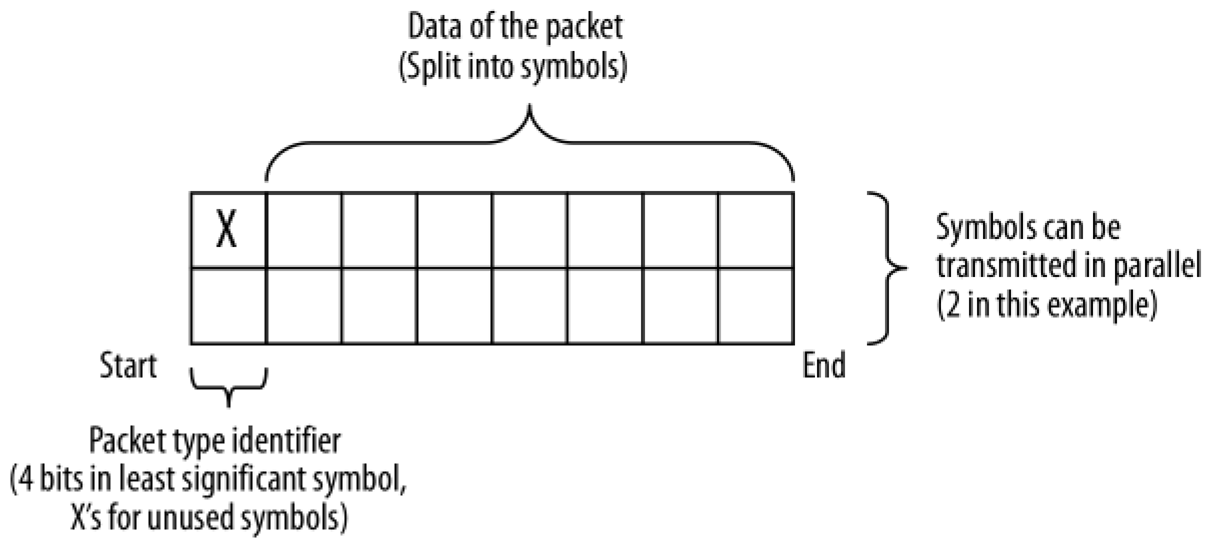
- ready - The sink makes use of this signal to show that it can receive new data. While the ready signal is low, the source stalls its operation and do not produce any data. When ready goes high, the source can resume operation, and produce data.
- valid - The source makes use of this signal to show that it has a valid output on the data bus, that can be read by the sink. The valid signal can only go high when the ready signal is already high. When both the signal are high, the transfer of data is done.
- Data - data to be transmited between the blocks.

Besides the mandatory signals, the protocol also define some optional signals, in the context of this project, it is important to mention the following:

- startofpacket/endofpacket - These two signals are important when packets are being transmitted between the blocks. Since there are both the start and end signal, each packet can have a different size without any problem.

9.2. Altera Video and Image Processing Suite

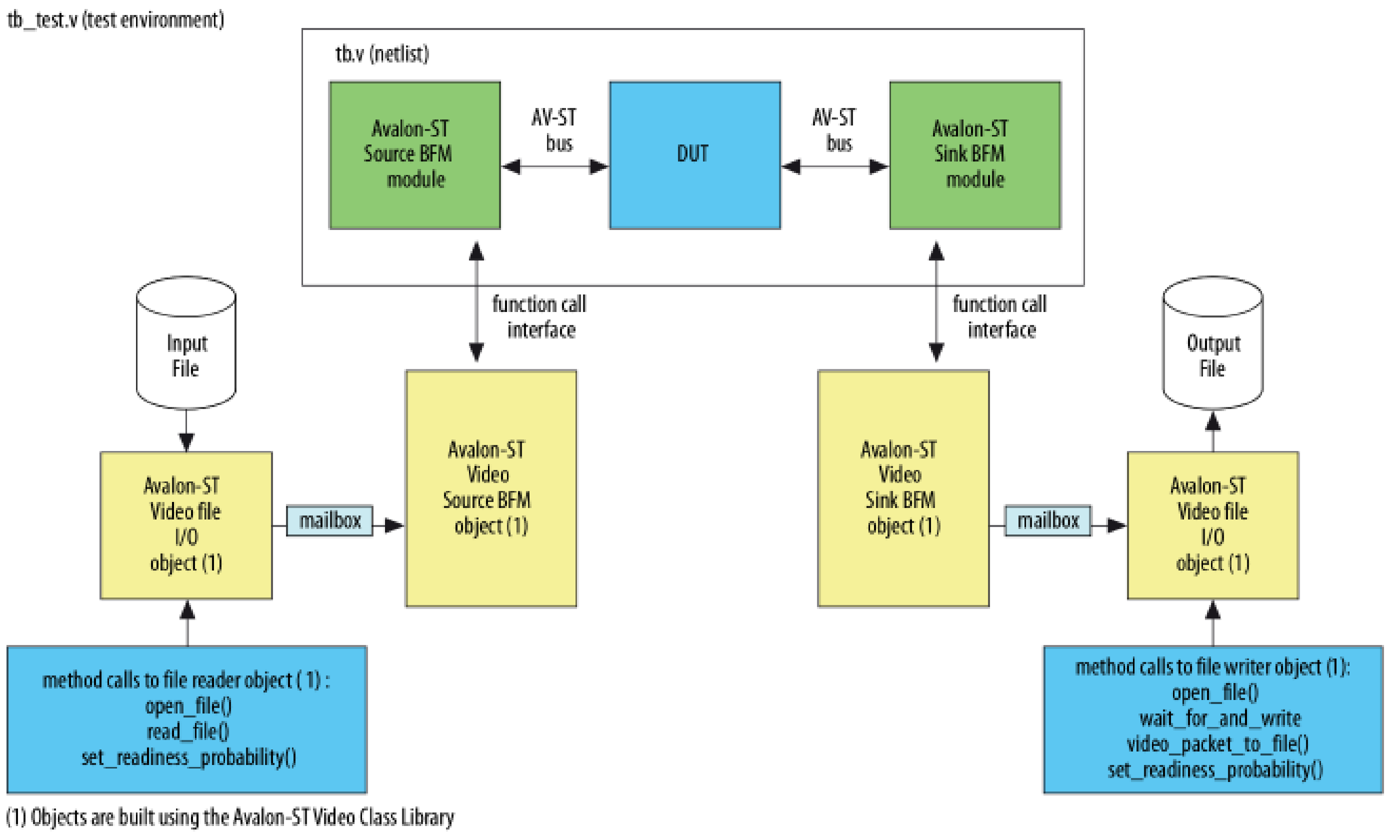
This suite is a set of IPs distributed freely by altera that focus on image and video processing. This suite makes use of the Avalon Streaming Protocol for communication between the blocks, and also defines some guideline on how the packages should be enconded.



The first symbol sent in a package should identify what is the package type:

- Video package - Contains video data, basicaly information about color for each pixel.
- Control package - Contains control information, such as end of frame, end of line, end of video, video resolution. This information can be used by the IPs for runtime configuration.
- User package - With undefined content.

Another useful thing provided by the VIP Suite, is the testbench.



tb_test.v (test environment)
tb.v (netlist)
Avalon-ST Source BFM module
AV-ST bus
DUT
AV-ST bus
Avalon-ST Sink BFM module
function call interface
function call interface
Input File
Output File
Avalon-ST Video file I/O object (1)
mailbox
Avalon-ST Video Source BFM object (1)
Avalon-ST Video Sink BFM object (1)
mailbox
Avalon-ST Video file I/O object (1)
method calls to file reader object (1) : open_file() read_file() set_readiness_probability()
method calls to file writer object (1): open_file() wait_for_and_write video_packet_to_file() set_readiness_probability()
(1) Objects are built using the Avalon-ST Video Class Library

Using this testbench it is possible to verify if an IP follows correctly the avalon streaming protocol and the VIP packet definitions. The testbench works by loading a video file from disk, inserting pixel by pixel in the DUT, and saving the output in another video file. The testbench provides lots of options to test the DUT against invalid data, bad timing and other problems that may appear.

9.3. IPs already developed

- RGB -> Luma (Modified from altera example)
- Threshold Filter
- Calculate the absolute difference between frames

9. Current Status

Original Video (10FPS)

After processing Video (10FPS-RED)

Background Estimation (30FPS)

After processing Video (30FPS-BLUE)

10. Resources

Source code on GitHub.

11. References

http://www.learnopencv.com/blob-detection-using-opencv-python-c/

http://brage.bibsys.no/xmlui/handle/11250/142506

http://www.codeproject.com/Articles/10248/Motion-Detection-Algorithms