EUROPEAN SPALLATION SOURCE

Document Type	Project Report
Document Number	ESS-0150750
Date	Aug 23, 2016
Revision	1 (1)
State	Preliminary
Confidentiality Level	Internal
Page	1 (24)

Target and Dump Proton Beam Imaging Systems CDR

Electronics implementation

	Name	Role/Title
Owner	David Michael Bang-Hauge	Electronics Engineer
Reviewer	Erik Adli	Oslo in-kind manager
Approver		

TABLE OF CONTENT **PAGE**

1 SCOPE

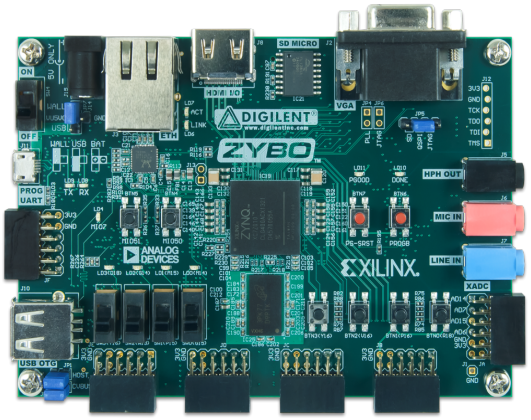
The purpose of this document is to describe the development and the implementation of the FPGA video processing algorithms. Chapter 3 presents the FPGA test platform, the target platform and the development tools used during the process. Chapter 4 describes the implementation, simulation and timing analysis of each processing algorithm in Vivado HLS. Chapter 5 presents an implementation running on the FPGA test platform.

2 INTRODUCTION

The motivation for developing and testing the video processing algorithms targeted for FPGA is to investigate whether the FPGA can be used as a machine protection signal or not. The frame rate of the final video system will run at 28 Hz. This means that the FPGA have to do all the processing needed in less than 35 ms. The most important tasks is to investigate and verify the functionality of the implemented algorithms by simulation, that timing is met for the target FPGA, what the frequency limit is and what is the resource usage. It is also important to do a functionality test by running the implemented algorithms on a FPGA. The Vivado High Level synthesis tool has been used to develop the implementation of video processing algorithms, simulate and verify the functionality of RTL generated, estimating the final timing, and resource utilization estimation. Vivado design suite is used for integration of the developed IP's and for configuration of the test platform FPGA. Five FPGA video processing algorithms have been developed at the time this document was written. A Median filter, Centroid position calculations, Percentage of beam outside defined footprint, Peak density of the beam and geometrical distortion correction.

3 FPGA TEST PLATFORM AND DEVELOPMENT TOOLS

3.1 Test Platform: Zybo Zynq-7000 ARM/FPGA SoC Trainer Board



Key Features

- Xilinx Zynq-7000 (XC7Z010-1CLG400C)
- 512 MB x32 DDR3 w/ 1050Mbps bandwidth
- 16-bits per pixel VGA output port
- High-bandwitdh peripheral controllers: 1G Ethernet, USB 2.0, SDIO

ZYNQ 7000 XC7Z010-1CLG400C

- 28,000 logic cells
- 240 KB Block RAM
- 650 MHz dual-core Cortex™-A9 processor

Figure 1 The Zybo board is a SoC development platform built around the Xilinx Zynq-7000 SoC (xc7z0101clg400c-1).

The Zybo Zynq developing board is used as a test platform for the video processing algorithm. It is built around a Xilinx Zynq-7000 SoC (xc7z010-1clg400c) and contains many features that make it a favorable choice for our application. Some of these features are: the VGA output port that was used for displaying a processed image, 512 MB DRAM makes it possible to store several test images. HLS timing and resource evaluations for the RTL synthesis and placement, was also done for the SoC device on the ZYBO board.

3.2 ESS Final Platform

The FPGA on the ESS final platform is a Kintex Ultrascale 40 (KU040) with yet unknown speedgrade. HLS timing and resource evaluations for the RTL synthesis and placement were done with speedgrade -1 and -3 (xcku040-sfva784-1-I, xcky040-sfva784-3-e). The speedgrade is a number indicating if a device is slower or faster, and have no consistent definition across the Xilinx FPGA families. For Kintex Ultrascale Ultrascale devices, -3 means faster than -1. Table 1 shows a few key resource comparisons of the Kintex Ultrascale 40 FPGA and the Zynq z-7010 SoC.

	Zynq Z-7010	Kintex Ultrascale 40
System Logic Cells (K)	28	530
Flip-Flops	35200	484800
LUTs	17600	242400
DSP Slices	80	1920
Block RAM (Mb)	2.1	21.1
I/O Pins	100	520
Processor Core	Dual core ARM CORTEX A9	None

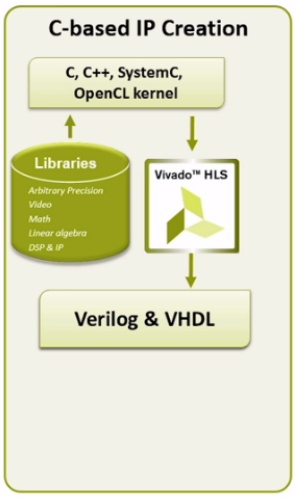
Table 1: Resource comparison of the Zynq Z-7010 and Kintex Ultrascale 40

3.3 Development tools

3.3.1 Vivado Design Suite: Vivado IDE

The Vivado Integrated Design Environment is a FPGA/SOC development tool from Xilinx. It allows you to do RTL design in VHDL, Verilog and SystemVerilog and integrate your own intellectual property (IP) together with a rich built-in IP library from Xilinx. Other key features are synthesizing designs, implementation for place and route, bitstream generation for configuring a target device, integrated logic analyzer, timing analysis, RTL diagram examination and much more. Vivado IDE is used for integrating the video processing algorithm in a larger functional test design for running on the ZYBO platform. This is explained further in chapter 5.

3.3.2 Vivado High Level Synthesis tool



The Vivado High Level Synthesis tool is an add-on in the Vivado Design Suite package. Its main feature is to transform a C-specification into a Register Transfer Level (RTL) implementation. C/C++ libraries provided like the HLS video library includes commonly used data structures, OpenCV interfaces, AXI4-Stream I/O, and video processing functions. Simulation of a design can be done pre- and post-synthesis. The pre-synthesis validation verifies the functionality of the C-program using a testbench also written in C. The post-synthesis verification, verifies that the RTL functionality is the same as the C-program functionality. The pre-synthesis validation and post-synthesis verification are referred to as C-simulation and C/RTL co-simulation respectively. The latency estimation in this document is generated from HLS synthesis. The latency estimation is presenting two values, latency and interval. The latency is the number of clock cycles it takes to produce the output. The interval is the number of clock cycles before new data can be presented. The final timing and resource estimation are generated using using the evaluation option during RTL export.

The video processing algorithms presented in this document have all been implemented using the Vivado High Level Synthesis tool.

4 HLS IMPLEMENTATION OF THE VIDEO PROCESSING ALGORITHMS

4.1 Centroid position

4.1.1 HLS implementation

The position of the centroid on the horizontal and vertical axis is given by:

$$C_x = \frac{\sum_x \sum_y xP(x,y)}{\sum_x \sum_y P(x,y)} \qquad C_y = \frac{\sum_x \sum_y yP(x,y)}{\sum_x \sum_y P(x,y)}$$

The RMS is given by:

$$RMS\_C_x = \sqrt{\frac{\sum_x \sum_y x^2 P(x,y)}{\sum_x \sum_y P(x,y)} - C_x^2} \qquad RMS\_C_y = \sqrt{\frac{\sum_x \sum_y y^2 P(x,y)}{\sum_x \sum_y P(x,y)} - C_y^2}$$

The implementation of these expressions in Vivado HLS is shown in the code snippet below

```
#pragma HLS dataflow

      LOOP0: for (int n = 1; n <= rows; n++){
                  for (int m = 1; m <= cols; m++)
                        {
                        #pragma HLS PIPELINE
                        video_in >> Apixel;
                        video_out << Apixel;
                        p_sum_x2 += m*m*Apixel.data;
                        p_sum_x1 += m*Apixel.data;

                        p_sum_y2 += n*n*Apixel.data;
                        p_sum_y1 += n*Apixel.data;

                        p_sum += Apixel.data;
                        }
                  }

      mp_sum_x1 = (float) (p_sum_x1)/(p_sum);
      mp_sum_y1 = (float) (p_sum_y1)/(p_sum);

      mp_sum_x2 = (float) (p_sum_x2)/(p_sum);
      mp_sum_y2 = (float) (p_sum_y2)/(p_sum);

      *p_sum_out = (float)p_sum;

      *Cog_X = mp_sum_x1;
      *Cog_Y = mp_sum_y1;

      *RMS_X = sqrtf((mp_sum_x2 - (mp_sum_x1)*(mp_sum_x1)));
      *RMS_Y = sqrtf((mp_sum_y2 - (mp_sum_y1)*(mp_sum_y1)));
```

4.1.2 HLS C/RTL co-simulation

A test image was running through a HLS C/RTL co-simulation of the algorithm. To have a set of comparable data, a centroid calculation was also done in Matlab. Figure 3 shows the test image and table 1 shows the results of the test image running through matlab and HLS C/RTL co-simulation.

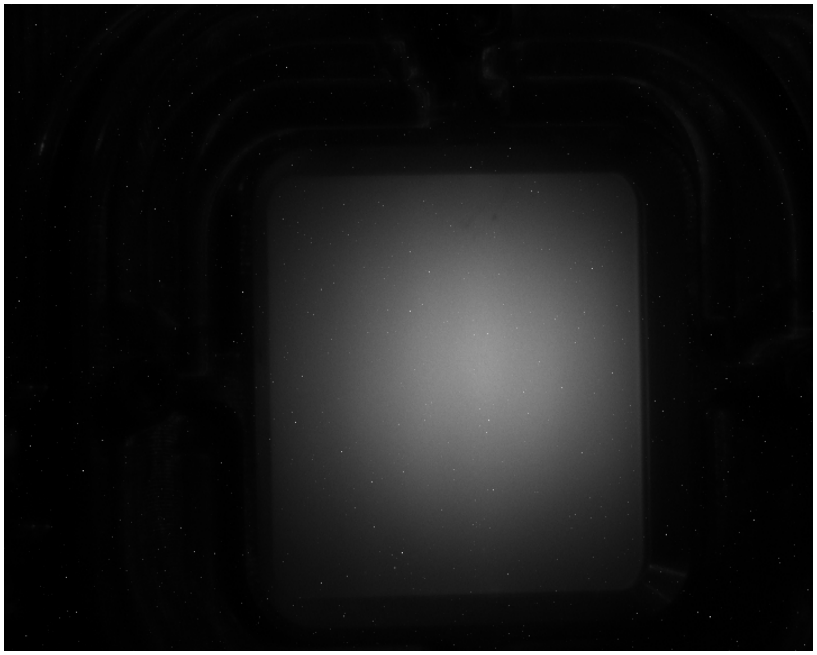


Figure 2: Test image for centroid calculation (720x576)

	MATLAB results	HLS C/RTL co-simulation results
Centroid X position	408.7179	408.717896
Centroid Y position	318.8381	318.838104
RMS X	106.1724	106.172401
RMS Y	106.0952	106.095215

Table 2

4.1.3 Timing and resource usage estimation

The results of the latency estimate for the synthesis and the final timing implementation of the centroid algorithm is shown in table 3.

FPGA/SoC FAMILY	ZYNQ-7000	ZYNQ-7000	Kintex Ultrascale	Kintex Ultrascale	Kintex Ultrascale
Device	xc7z010clg400-1	xc7z010clg400-1	xcku040-sfva784-1-i	xcku040-sfva784-1-i	xcku040-sfva784-3-e
CP required (ns)	10	5	10	5	3
Latency (Clock pulses)	417068	420562	417053	420532	421707
Latency (ms)	4,17	2,10	4,17	2,10	1,27
Interval (Clock pulses)	417026	420482	417026	420482	421634
Interval (ms)	4,17	2,10	4,17	2,10	1,26
CP achieved post-synthesis	8.098	6.128	6.235	3.764	2.167
CP achieved post-implemetation	8.719	5.188	7.819	4.357	2.584
RTL	vhdl	vhdl	vhdl	vhdl	vhdl
SLICE	1598	1786	-	-	-
LUT	4605	4776	4627	4637	4822
FF	4701	7729	3234	5183	7265
DSP	16	16	16	16	16
SRL	243	320	244	283	330
BRAM	0	0	0	0	0

Table 3: The latency estimate of the centroid algorithm synthesis

CP (Clock period) required is the target clock period in ns, set by the user pre-synthesis. CP achieved post-synthesis is the estimated minimum clock period the synthesized design will run at without having any timing violations. The CP achieved post-implementation is the estimated minimum clock period the implemented design will run at without having any timing violations. The estimated timing post-synthesis is done on a gate-level netlist based on a FPGA-family (Zynq 7000 or Kintex Ultrascale) basic primitive's library. The estimated timing post-implementation is done for the synthesised netlist placed and routed on a specific FPGA device (xc7z010clg400-1 or xcku040-sfva784-1-i). However the post-implementation timing estimation will not take into account that there can be other implementations in a design which can make routing and placement more complex, less efficient and with an increased route delay. This means that integrating the design in a larger design running on the maximum estimated frequency, the timing may fail.

The latency is the number of clock cycles it takes to produce the output. The interval is the number of clock cycles before new inputs can be applied. The table is also showing latency and interval in terms of milliseconds. In the case of the centroid position calculation running on the Zynq-z7010 with a target clock period of 10 ns. The results of the calculation is ready 417068 clock cycles (Latency*CP = 4.17 ms) after the first pixel of a frame enters the function. But a pixel from a new frame can be applied after 417026 clock cycles.

4.1.4 Discussion

As seen from table 2. the algorithm implemented in HLS succeeded in generating the same results as the matlab implementation. All solutions fulfils the requirement of completing the calculations within 35 ms but the post-implementation timing was not met for the xc7z010 running on 200 MHz. The minimum clock period achieved in the post-implementation was 5.188 ns. This means that the fastest achievable processing time on the ZYNQ-7z010 when having a resolution of 720x576 is 420562*5.188 ns = 2.18 ms but as mentioned the value may change when doing an implementation in a larger design.

4.2 Median Filter

4.2.1 HLS implementation

The median filter is implemented by taking the values of a 3x3 sliding window through a sorting network to find the median value to output. The input pixel to output pixel flow is shown in figure 4.

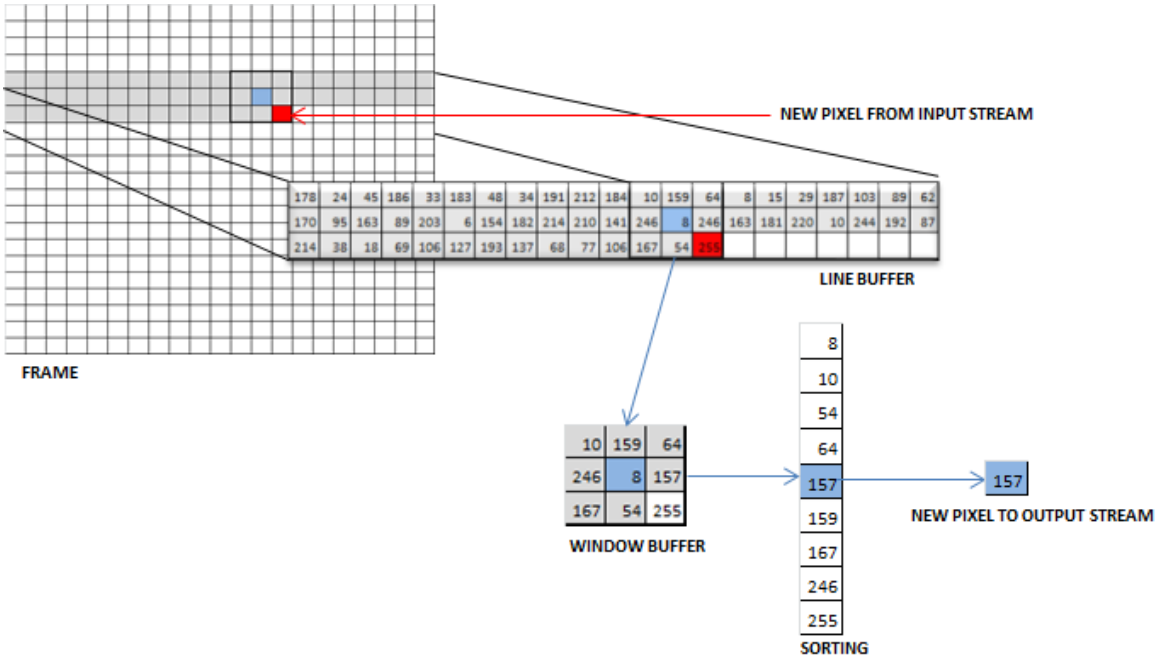


Figure 3: Median Filter implementation

4.2.2 HLS C/RTL co-simulation

Noise was added to the test image in figure 3 for the C/RTL co-simulation of the median filter. The result is shown in figure 5.

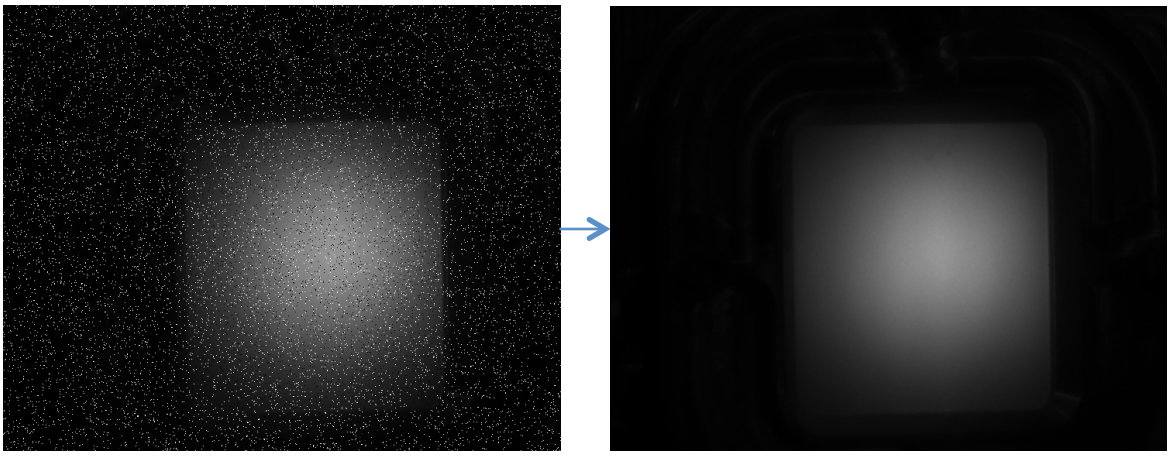


Figure 5: An image with popcorn noise streamed through the median filter in the C/RTL co-simulation.

4.2.3 Timing and resource usage estimation

The results of timing and resource usage estimation of the Median filter is shown in table 5

FPGA/SoC FAMILY	ZYNQ-7000	ZYNQ-7000	Kintex Ultrascale	Kintex Ultrascale	Kintex Ultrascale
Device	xc7z010clg400-1	xc7z010clg400-1	xcku040-sfva784-1-i	xcku040-sfva784-1-i	xcku040-sfva784-3-e
CP required (ns)	10	5	10	5	3
Latency (Clock pulses)	414732	414740	414731	414736	414739
Latency (ms)	4,15	2,07	4,15	2,07	1,24
Interval (Clock pulses)	414733	414741	414732	414737	414740
Interval (ms)	4,15	2,07	4,15	2,07	1,24
CP achieved post-synthesis	8.959	6.672	5.789	3.009	2.187
CP achieved post-implemetation	9.554	6.032	7.500	4.144	2.764
RTL	vhdl	vhdl	vhdl	vhdl	vhdl
SLICE	321	325	-	-	-
LUT	867	729	791	825	749
FF	797	1161	743	1000	1147
DSP	4	4	4	4	4
SRL	21	21	10	21	21
BRAM	2	2	2	2	2

Table 4: The latency estimate of the median filter synthesis

4.2.4 Discussion

The C/RTL co-simulation of the Median filter removed the pre-added popcorn noise as expected. This implementation of the median filter shifts the pixels one place down and one place to the right. The reason for this is that the output pixel corresponding to the median of all elements in the sliding window has the same frame coordinate as the last incoming pixel which is bottom-right. It should be presented as the pixel in the middle of the window as shown in figure 4. The Clock period achieved post-implementation shows that timing is not met for the ZYNQ running with a 200 MHz clock. As shown in the table all other solutions fulfils the target requirement and are also well within the required latency.

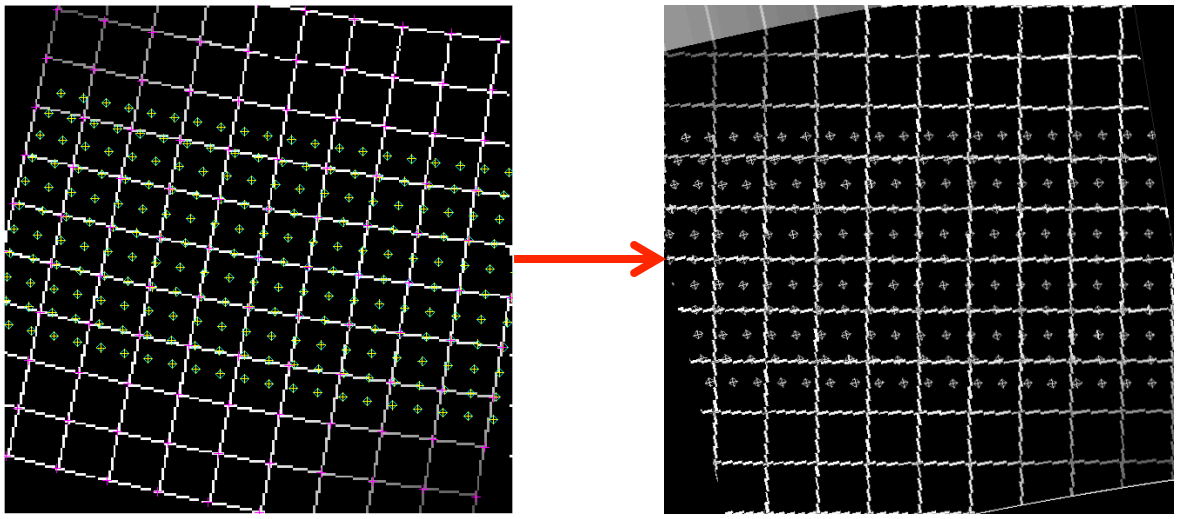
4.3 Geometrical Distortion Correction

4.3.1 HLS implementation

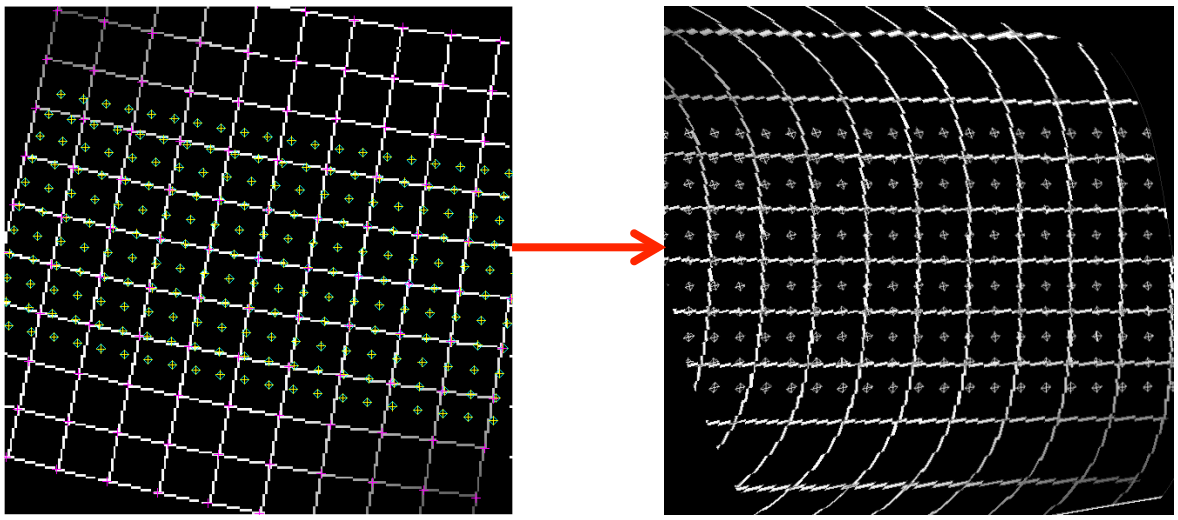
To implement the geometrical distortion correction, the HLS video library was used. The HLS function REMAP, remaps the source image to a destination image according to a given remapping. A line buffer buffers the required amount of horizontal lines needed for vertical displacement. A map for both vertical and horizontal frame coordinates, maps the output pixel to the corresponding input pixel.

4.3.2 HLS C/RTL co-simulation

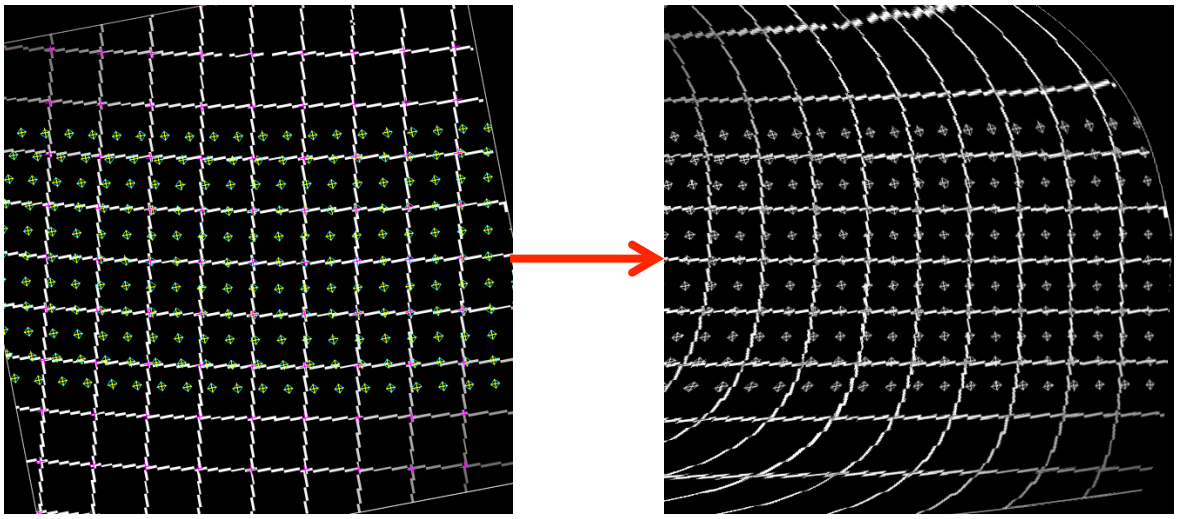
The following figures are the results of three C/RTL co-simulations. The resolution of the original image is 585x585 pixels. The line buffer consists of 128 lines.



Original image | Rotation



Rotation and distortion correction



Pre rotated Image | Distortion correction

4.3.3 Timing and resource usage estimation

To achieve comparable results the resolution was set to 720x576. The line buffer was set to 64 lines.

FPGA/SoC FAMILY	ZYNQ-7000	ZYNQ-7000	Kintex Ultrascale	Kintex Ultrascale	Kintex Ultrascale
Device	xc7z010clg400-1	xc7z010clg400-1	xcku040-sfva784-1-i	xcku040-sfva784-1-i	xcku040-sfva784-3-e
CP required (ns)	10	5	10	5	3
Latency (Clock pulses)	441477	444515	441477	441477	442692
Latency (ms)	4,41	2,22	4,41	2,21	1,33
Interval (Clock pulses)	441475	444515	441475	441475	442691
Interval (ms)	4,41	2,22	4,41	2,21	1,33
CP achieved post-synthesis	5.995	5.040	5.298	3.676	2.420
CP achieved post-implemetation	7.995	5.035	6.691	4.187	2.780
RTL	vhdl	vhdl	vhdl	vhdl	vhdl
SLICE	414	428	-	-	-
LUT	1021	1064	1009	997	1109
FF	1215	1334	1216	1264	1276
DSP	2	2	2	2	2
SRL	60	63	64	64	74
BRAM	37	37	29	37	37

Table 5

4.3.4 Discussion

In the two first simulations the image had a large vertical displacement due to rotation. To remap this correctly without losing any pixels the remap function must contain enough line buffers to cover the vertical displacement. This will consume much of the resources available in terms of BRAM. Using a resolution of 720x576 and a line buffer with 128 lines the IP will use 65 BRAM blocks of 18k. This is 54% of the available BRAM on the Zynq-xc7z010. One possibility to avoid this is to pre-rotate the camera. In the final timing implementation and the resource usage implementation the line buffer was set to 64 lines instead. The BRAM usage then drops to 37 blocks. Again the timing is not met for the Zynq running with 200 MHz. All other solutions fulfil the timing requirement.

4.4 Peak Density of the Beam

4.4.1 HLS implementation

The peak density implementation was done in a similar way was as the median filter. The mean value of the pixels in a sliding window buffer is compared with the last maximum mean value. If the mean value is larger than the last maximum mean value the new maximum mean value is stored together with the centre position of the window.

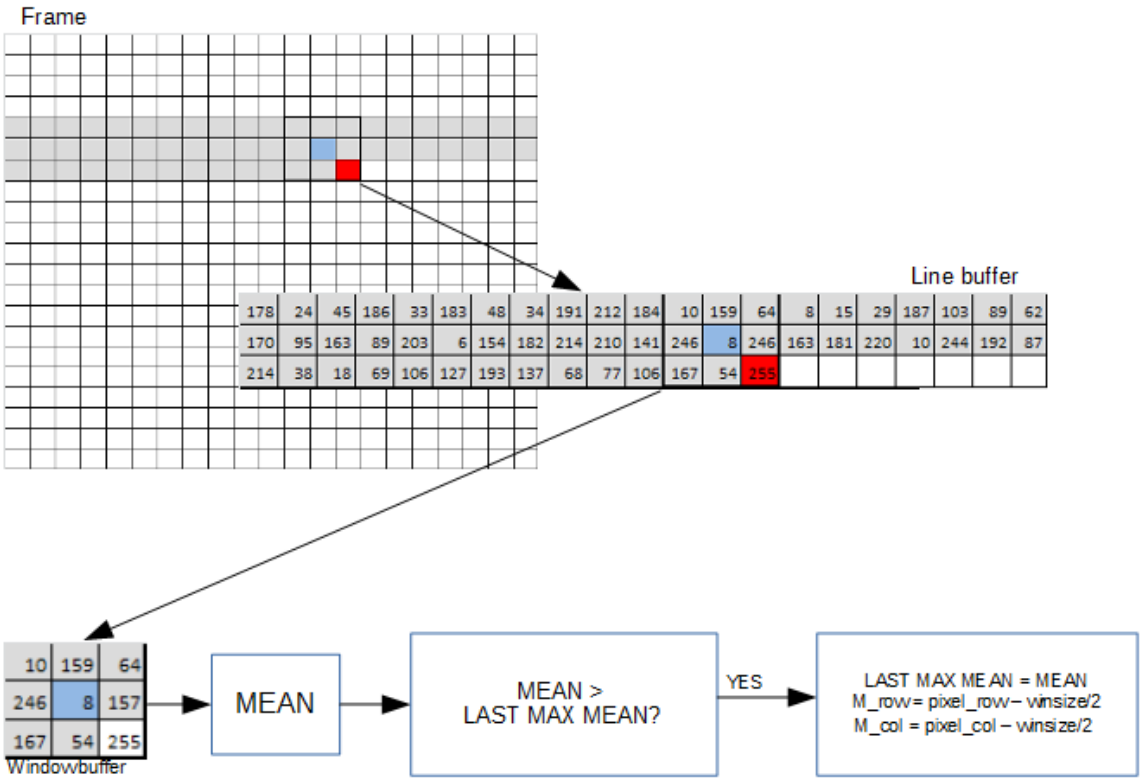


Figure 4

4.4.2 HLS C/RTL co-simulation

The image shown in figure 3 used for the centroid calculation was also used as a test image for the simulation of the peak density algorithm. Table 5 show a comparison of the simulation done in HLS and the results from a matlab implementation. The window for both HLS and matlab implementation was set to 10x10 pixels. The row and column indexes in the matlab results are one more than for the HLS results. This is because array indices starts from 1 in matlab and from 0 in the C programming language.

	MATLAB results	**HLS C/RTL co-simulation results**
Peak density	149.3900	149.389999
Row	321	320

Column	423	422

Table 6

4.4.3 Timing and resource usage estimation

FPGA/SoC FAMILY	ZYNQ-7000	ZYNQ-7000	Kintex Ultrascale	Kintex Ultrascale	Kintex Ultrascale
DEVICE	xc7z010clg400-1	xc7z010clg400-1	xcku040-sfva784-1-i	xcku040-sfva784-1-i	xcku040-sfva784-3-e
CP required (ns)	10	5	10	5	3
Latency (Clock pulses)	414752	1658940	414738	414752	829490
Latency (ms)	4,15	8,29	4,15	2,07	2,49
Interval (Clock pulses)	414752	1658940	414738	414752	829490
Interval (ms)	4,15	8,29	4,15	2,07	2,49
CP achieved post-synthesis	7.012	4.517	6.718	3.873	2.629
CP achieved post-implemetation	8.547	4.762	8.547	4.423	2.898
RTL	vhdl	vhdl	vhdl	vhdl	vhdl
SLICE	859	1053	-	-	-
LUT	2153	2432	2119	2229	2434
FF	2610	4369	1859	2604	3259
DSP	4	4	4	4	4
SRL	120	96	89	95	100
BRAM	9	9	9	9	9

Table 7

4.4.4 Discussion

As seen from table 9 the RTL gives the same results as a matlab implementation which means that the RTL works as intended. All solutions managed to fulfil the required clock period but for the 200 MHz implementation for the Zynq-7z010 the latency is almost 4 times the latency as for the 100 MHz implementation. The target initiation interval for the mean calculation of the window is set to 1 clock cycle. Which means the synthesis tool will try to add the elements in the window in parallel. The achieved clock cycles are 4 cycles hence the total latency will be 4 times longer. All solutions fulfil the 35 ms requirement.

4.5 Percentage of beam outside defined footprints

4.5.1 HLS implementation

This implementation sums up the pixel values for the total, outside inside and inside a rectangle. The rectangle is defined with parameters top, bottom, left and right borders. The timing and resource usage estimation is based on outputting three summations.

4.5.2 HLS C/RTL co-simulation

In the HLS C/RTL co-simulation the image shown I figure 3 was used. The border parameters were set to: top border = 200, bottom border = 400, left border = 300 and right border = 500.

	MATLAB result	HLS C/RTL co-simulation results
Total pixel sum	9508224	9508224
Sum outside rectangle	5230963	5230963
Sum inside rectangle	4277261	4277261

Table 8

4.5.3 Timing and resource usage estimation

FPGA/SoC FAMILY	ZYNQ-7000	ZYNQ-7000	Kintex Ultrascale	Kintex Ultrascale	Kintex Ultrascale
Device	xc7z010clg400-1	xc7z010clg400-1	xcku040-sfva784-1-i	xcku040-sfva784-1-i	xcku040-sfva784-3-e
CP required (ns)	10	5	10	5	3
Latency (Clock pulses)	417032	417035	417029	417032	417035
Latency (ms)	4,17	2,09	4,17	2,09	1,25
Interval (Clock pulses)	417026	417026	417026	417026	417026
Interval (ms)	4,17	2,09	4,17	2,09	1,25
CP achieved post-synthesis	5.683	5.683	3.384	2.705	1.997
CP achieved post-implemetation	7.571	5.490	5.834	3.449	2.446
RTL	vhdl	vhdl	vhdl	vhdl	vhdl
SLICE	448	493	-	-	-
LUT	1300	1339	1550	1302	1419
FF	1389	1724	1106	1389	1712
DSP	0	0	0	0	0
SRL	66	72	64	66	72
BRAM	0	0	0	0	0

Table 9

4.5.4 Discussion

It is not decided yet whether the output should be three summations or if it is going to be the actual percentage of beam outside the rectangle. To output the percentage an additional division will be included in the implementation. This may affect the total latency. The timing is not met for Zynq-z7010 running on 200MHz. The timing is met for all other solutions and fulfils the latency requirement.

4.6 Total estimated resource usage

Table 10 and 11 show the estimated total key resource usage of the five video processing implementations.

	xcku040-sfva784-1-i	xcku040-sfva784-1-i	xcku040-sfva784-3-i	Available
CP required	10	5	3	
LUT	10096	9990	10533	242400
FF	8158	11440	14659	484800
DSP	26	26	26	1920
BRAM	40	48	48	1200 18Kb + 600 36Kb

Table 10: Total estimated resource usage for the Kintex Ultrascale 40

	xc7z010clg400-1	xc7z010clg400-1	Available
CP required	10	5	
LUT	9946	10340	17600
FF	10712	16317	35200
DSP	26	26	80
BRAM	48	48	120

Table 11: Total estimated resource usage for the Zynq-7z010

4.7 Latency resolution dependency

The implementations were synthesised for several resolutions to find the relationship between resolution and latency. The resolutions were:

- 720x480
- 720x576
- 1280x720
- 1920x1080

The tables below shows the trendlines for the relationship between latency and resolution

	xc7z010clg400-1	xc7z010clg400-1	xcku040-sfva784-1-i	xcku040-sfva784-1-i	xcku040-sfva784-3-i
CP required	10.000	5.000	10.000	5.000	3.000
Latency	1.0013*Res + 1671.4	1.0034*Res + 3890.8	1.0013*Res + 1656.4	1.0004*Res + 3431.7	1,0017*Res + 4123,5
Interval	1.0013*Res + 1629.4	10034*Res + 3810,8	1,0013*Res + 1629.4	1.0004*Res + 3381.7	1,0017*Res + 4050,5

Table 12: Centroid position calculation latancy resolution dependency

	xc7z010clg400-1	xc7z010clg400-1	xcku040-sfva784-1-i	xcku040-sfva784-1-i	xcku040-sfva784-3-i
CP required	10.000	5.000	10.000	5.000	3.000
Latency	1.0535*Res + 22955	1.0532*Res + 25873	1.0535*Res + 22955	1.0535*Res + 22955	1.0529*Res + 24159
Interval	1.0535*Res + 22953	1.0532*Res + 25873	1.0535*Res + 22953	1.0535*Res + 22953	1.0529*Res + 24158

Table 13: Geometrical distortion correction latency resolution dependency

	xc7z010clg400-1	xc7z010clg400-1	xcku040-sfva784-1-i	xcku040-sfva784-1-i	xcku040-sfva784-3-i
CP required	10.000	5.000	10.000	5.000	3.000
Latency	Res + 32	4*Res + 60	Res + 18	Res + 32	2*Res + 50
Interval	Res + 32	4*Res + 60	Res + 18	Res + 32	2*Res + 50

Table 14: Peak density of the beam latency resolution dependency

	xc7z010clg400-1	xc7z010clg400-1	xcku040-sfva784-1-i	xcku040-sfva784-1-i	xcku040-sfva784-3-i
CP required	10.000	5.000	10.000	5.000	3.000
Latency	Res + 11.507	Res + 23.451	Res + 8.0422	Res + 12.056	Res + 14.563
Interval	Res + 12.507	Res + 24.451	Res + 9.0422	Res + 13.056	Res + 15.563

Table 15: Median filter latency resolution dependency

	xc7z010clg400-1	xc7z010clg400-1	xcku040-sfva784-1-i	xcku040-sfva784-1-i	xcku040-sfva784-3-i
CP required	10.000	5.000	10.000	5.000	3.000
Latency	1.0013*Res + 1635.4	1.0018*Res + 1753.2	1.0013*Res + 1632.4	1.0013*Res + 1635.4	1.0013*Res + 1638.4
Interval	1.0013*Res + 1629.4	1.0018*Res + 1744.2	1.0013*Res + 1629.4	1.0013*Res + 1629.4	1.0013*Res + 1629.4

Table 16: Percentage of beam outside defined footprints latency resolution dependency

4.8 Discussion

Five implementations of different video processing algorithms have been investigated in this document, and the functionality of all them seems to work as intended. Table 10 shows that the total resource usages for all solutions running with the Kintex Ultrascale 40 are well within the available resources. It is uncertain what additional video processing algorithms would consume but since the algorithms implemented until now use up to 50% of the resources available on ZYNQ-z7010 as seen from table 11, processing algorithms such as the Richardson-Lucy deconvolution would certainly blow the ZYNQ-z7010 resource budget. From the timing and resource usage estimation it is also obvious that the Zynq-z7010 is not capable of running any of the algorithms at 200 MHz. The requirement to process the data faster than 28 Hz is fulfilled for all other solutions.

5 TEST PLATFORM IMPLEMENTATION

5.1 FPGA run without geometrical correction

The implementation was done using the same test image used for all C/RTL co-simulations except the geometrical correction algortihm. Figure 5 shows the implementation in Vivado using the Vivado HLS generated IP's for *Centroid position*, *Median Filter*, *Percentage of beam outside defined footprint* and the *Peak density of the beam*.

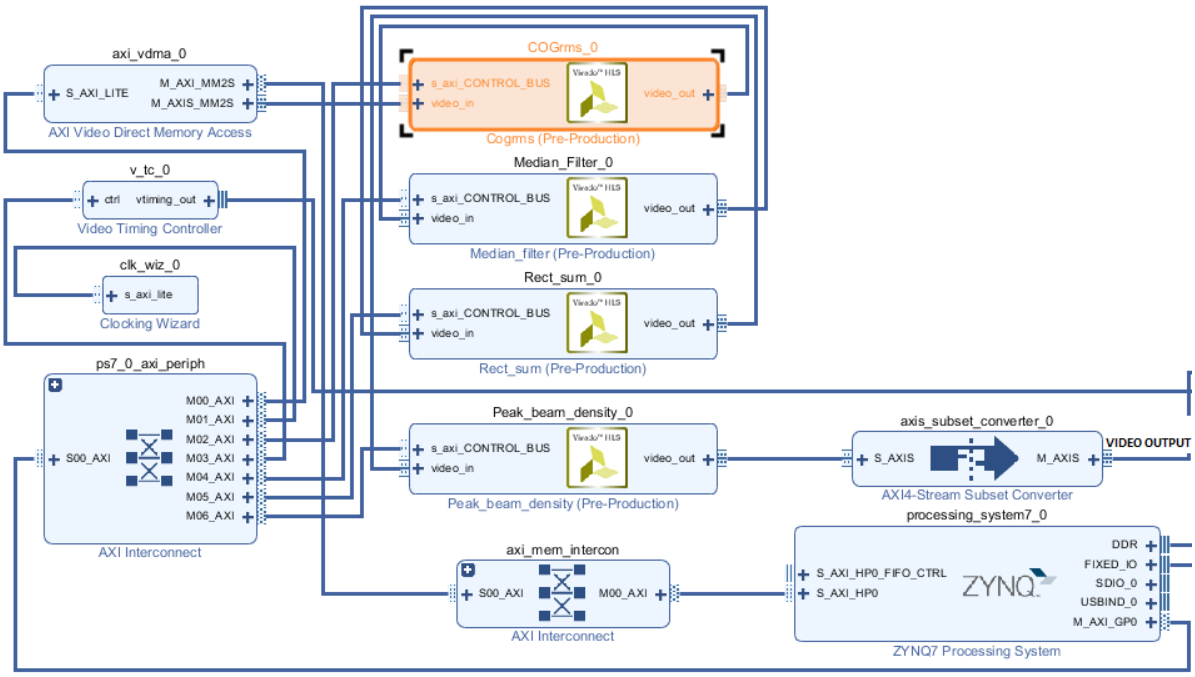


Figure 5

The Zynq7 processing system containing the ARM cortex A9 was exploited for AXI bus communication with the IP's and for receiving and sending serial data from/to a computer. The test image is stored onto the Zybo DRAM using the serial line of the Zybo to receive the image data sent from Matlab. The results of the configuration shown in figure 5 on the Zynq and running software on the ARM processor that integrates the HLS IP and reads data from the AXI bus is shown in figure 6 and 7. During the first run the median filter is turned off. On the second run a noise version of the image was loaded onto the DRAM and the median filter was turned on. This is shown in figure 8 and 9.

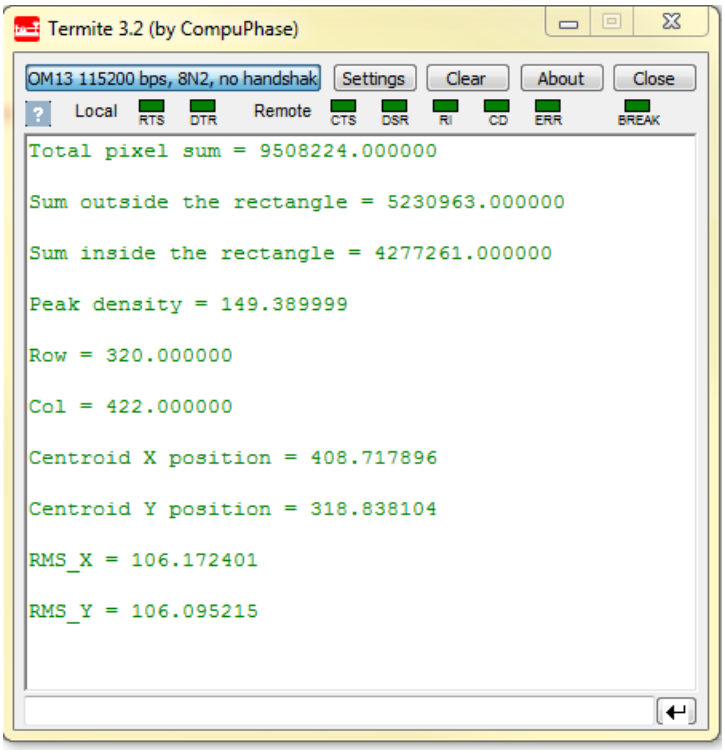


Figure 6: Results in the PC terminal

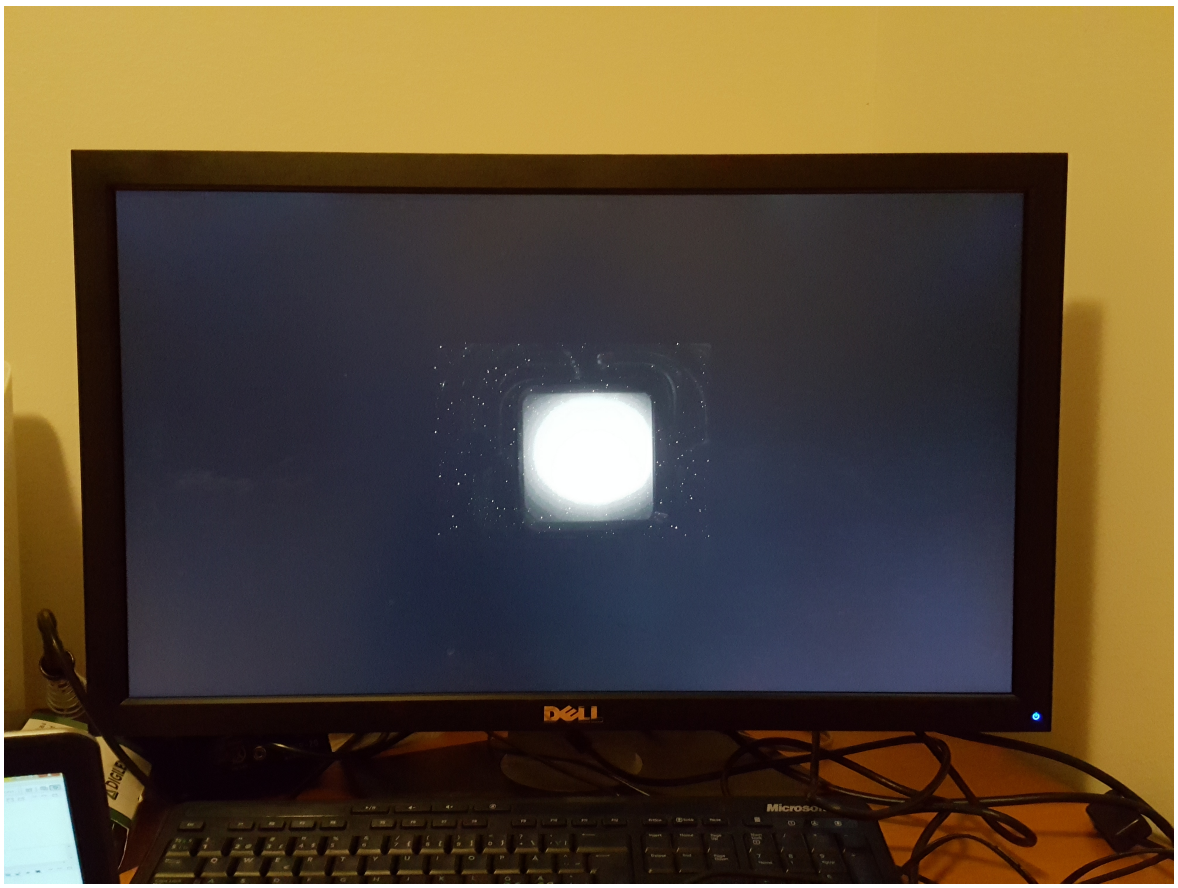


Figure 7: The image shown on a screen connected to the VGA port of the ZYBO

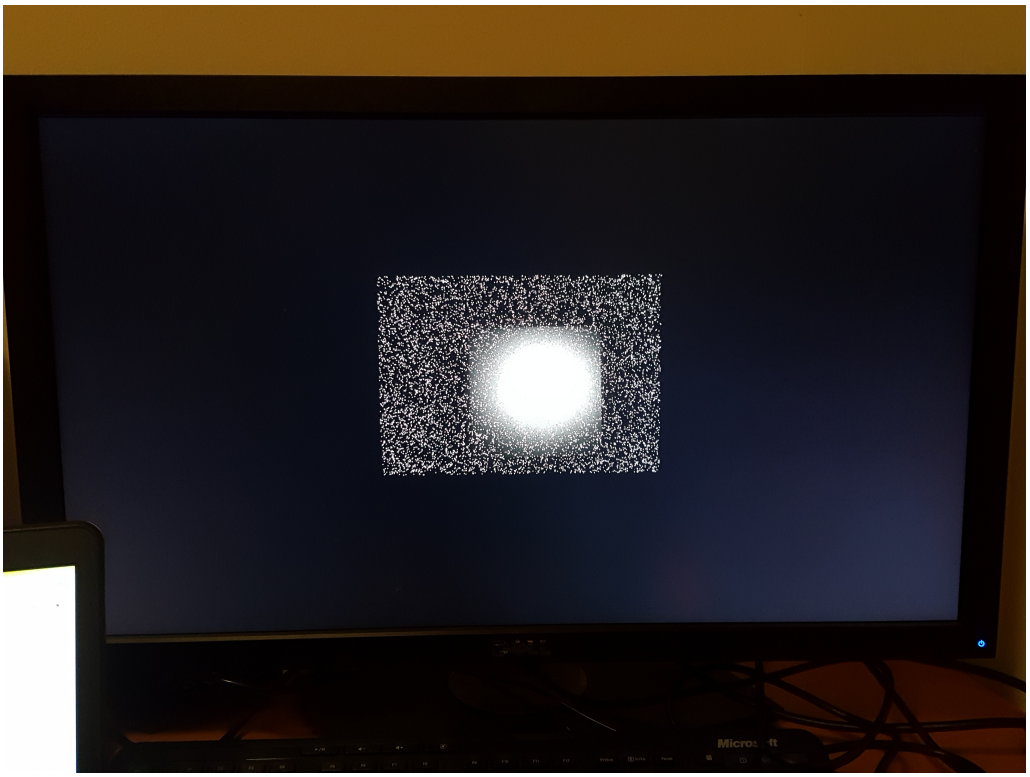


Figure 8: Noisy image when median filter is turned off

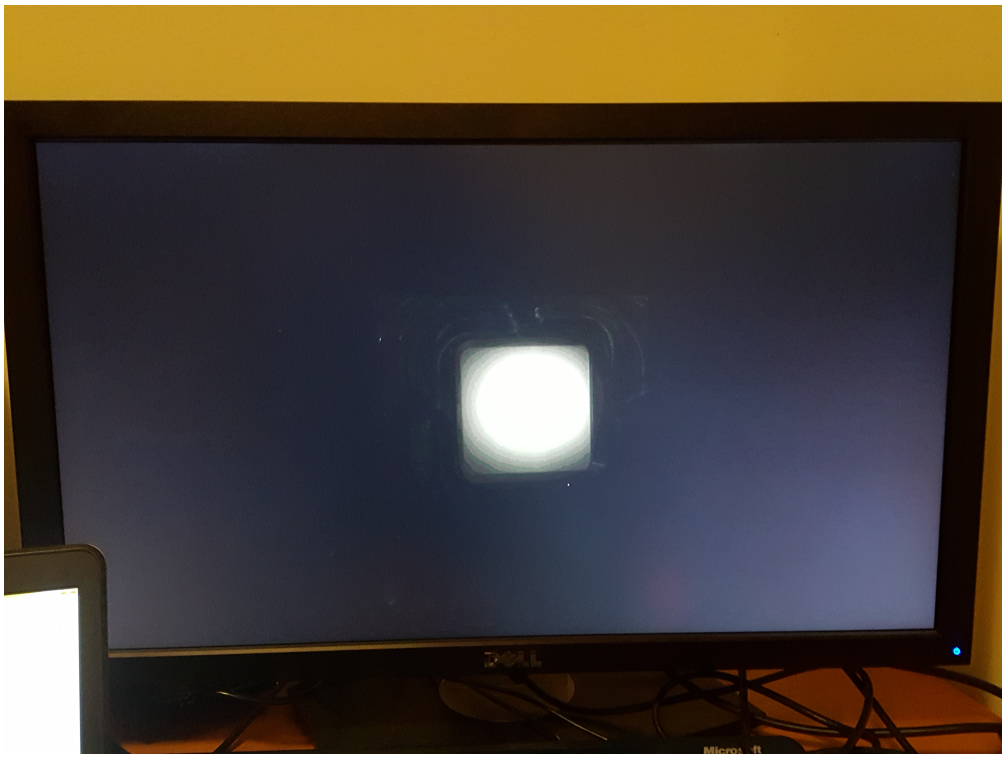


Figure 9: Noisy image when median filter is turned on

The results of the first run is shown in table

	FPGA results	HLS C/RTL co-simulation results
Centroid X position	408.7179	408.717896
Centroid Y position	318.838104	318.838104
RMS X	106.172401	106.172401
RMS Y	106.095215	106.095215
Total pixel sum	9508224	9508224
Sum outside rectangle	5230963	5230963
Sum inside rectangle	4277261	4277261
Peak Density	149.389999	149.389999
Row	320	320
Column	422	422

Table 17: The results achieved during the FPGA run compared with the results achieved during the HLS simulation

5.2 FPGA run with geometrical distortion correction

The geometrical distortion correction IP was integrated into the Vivado design suite using two VDMAs. One streaming the image data and one streaming the image mapping. The setup is shown in the figure below.

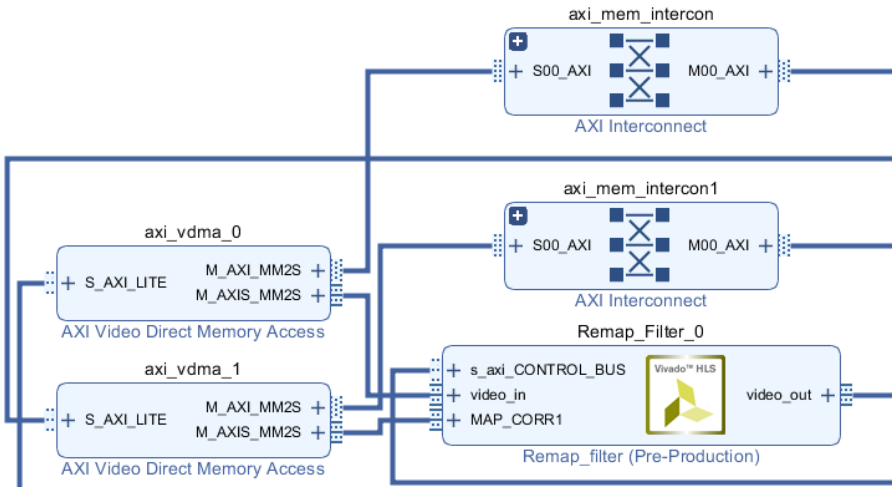


Figure 10: Geometrical distortion correction Vivado setup

The image loaded into the FPGA was a pre-rotated image similar to the image used for the C/RTL co-simulation done for the geometrical distortion correction. The results of the run is shown in the two figures below:

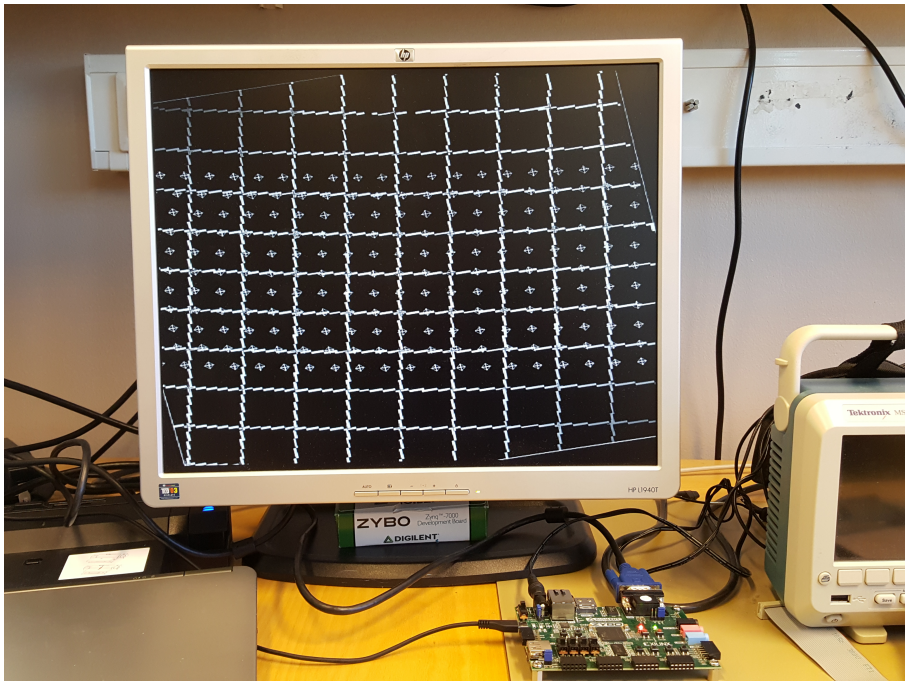


Figure 11: Distorted grid image running with a 1:1 mapping

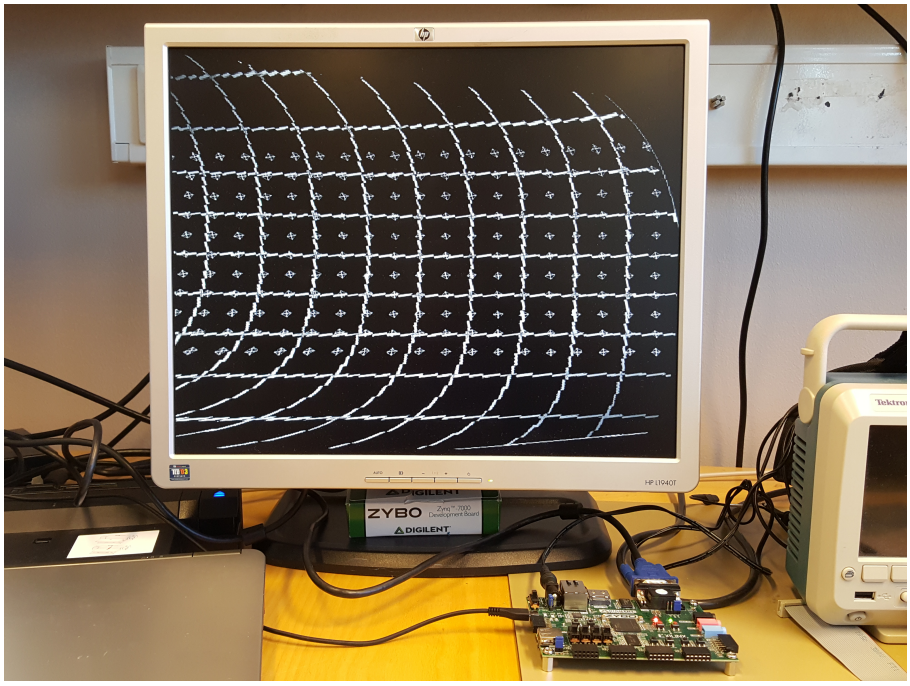


Figure 12: Distorted grid image running with a distortion correction mapping

5.3 Discussion

The results from the FPGA readout gave the exact same results as the C/RTL co-simulation. This verifies that the functionality works as intended. The geometrical

distortion correction did run as intended. The distortion correction mapping for the pre-rotated image is not perfectly correcting the image but as the final result is the same as the C/RTL co-simulation it means that this originates from an incorrect pre-created mapping. The FPGA implementation work as intended.

6 CONCLUSION

Five video processing algorithms have been developed. Verification on the functionality has been done by C/RTL co-simulation in Vivado HLS and on a ZYNQ-z7010 FPGA. The results from the C/RTL co-simulation showed that the functionality worked as intended. The HLS latency estimations for the solutions where timing was met, all individual implemented algorithms were well within 35 ms but as more complex algorithms will be implemented the latency margin have to be as large as possible. The implementations until now have been done with single precision numbers where needed and there have been no attempt to optimise the code by reduce the number of bits the values are represented with. The use of fixed point instead of floating numbers can reduce latency and resource usage and must be looked into in the future.

7 GLOSSARY

See also: https://confluence.esss.lu.se/display/BIG/Abbreviations

8 DOCUMENT REVISION HISTORY

Revision	Reason for and description of change	Author	Date
1	CDR	David Michael Bang-Hauge	2017-10-05