



UNIVERSITY OF KAISERSLAUTERN

Department of Electrical Engineering and Information Technology

Microelectronic Systems Design Research Group

THESIS

Real-Time Image Processing on FPGA

Implementation of Miscellaneous Edge Detection Algorithms

Presented: May 10, 2010

Author: Matthias Jung

Research Group Chief: Prof. Dr.-Ing. N. Wehn

Tutor: Dipl.-Ing. M. Alles

Statement

I declare that this thesis was written by myself and that I only used my specified aids.

Kaiserslautern, May 10, 2010

Matthias Jung

Contents

1	Introduction	1
2	Transmission Systems	2
2.1	Signal and System Theory	2
2.2	Dirac Delta	2
2.3	Impulse Response	3
2.4	Convolution	3
2.5	Discrete Convolution	4
2.6	Frequency Domain	5
2.7	Fourier Optics	5
3	Optical Filters	6
3.1	Neighbourhood Operators	6
3.2	Linear Filters	7
3.2.1	Moving Average - Low-Pass Filters	7
3.2.2	Edge Detection - High Pass Filters	9
3.2.3	Difference Approach	9
3.2.4	Robert Cross Operator	11
3.2.5	Sobel Operator	12
3.2.6	Laplace Operator	13
3.3	Non-Linear Filters	14
3.3.1	Rank Value Filter	14
4	Real-Time Systems	15
4.1	Scheduling	15
5	FPGAs	16
6	Hardware	18
6.1	Camera	18
6.2	FPGA Board - Euterpe	19
7	Implementation	20
7.1	Cam Capture	20
7.2	Filter	21
7.2.1	Dirac	22
7.2.2	Roberts Filter	22
7.2.3	Sobel	23
7.2.4	Laplace	23
7.3	Scheduler	23
7.4	Framebuffer	24
7.5	SRAM Controller	26
7.5.1	Proof: <i>Is the SRAM access scheduling feasible?</i>	26
7.6	Synthesis	27
8	Conclusion	29
9	Appendix	30
	Bibliography	30

List of Figures	31
Source Code	32
9.1 cam_fb.vhd	32
9.2 cam_capture.vhd	36
9.3 filter.vhd	37
9.4 scheduler.vhd	42
9.5 sram_controller.vhd	45
9.6 framebuffer.vhd	47

1 Introduction

Today's digital technology such as multimedia, communication, computers, robots and medical technology assists us in our daily life. For instance industrial robots on an assembly line are able to recognize simple objects like screws and sort these automatically according to their size or detect and dump faulty work pieces. Computers are able to digitalize a text on a paper with OCR (Optical Character Recognition). Another example is a magnetic resonance scanner which is used to find and qualify tumors in a human body. All these applications use digital image processing algorithms.

It is hard to implement these algorithms on a general purpose computer with respect to real-time aspects because a high amount of concurrency is needed to compute the result of a large filter in only one clock cycle. There are special hardware components such as DSPs (Digital Signal Processors) or FPGAs (Field Programmable Gate Arrays) which can directly compute typical operations e.g. convolution. A FPGA is often a very good choice instead using a DSP. For a FIR filter implementation with eight coefficients a DSP needs eight clock cycles. A FPGA only needs one clock cycle for all eight operations but it takes more area for the additional hardware.

Figure 1.1 shows a simplified hierarchy of digital image processing. The immense problem is the high amount of data. An 8-bit grey scale image with the resolution of 1280×1024 pixel needs 1.25MByte memory. With 80 pictures per second the transfer rate is 100MByte/s. It is necessary to reduce the amount of data considering only contours and edges of the sighted objects to remove information which is not needed by the succeeding components e.g. object recognition. With an edge detection algorithm the picture can be reduced to an 1-bit image with the size of 16KByte.

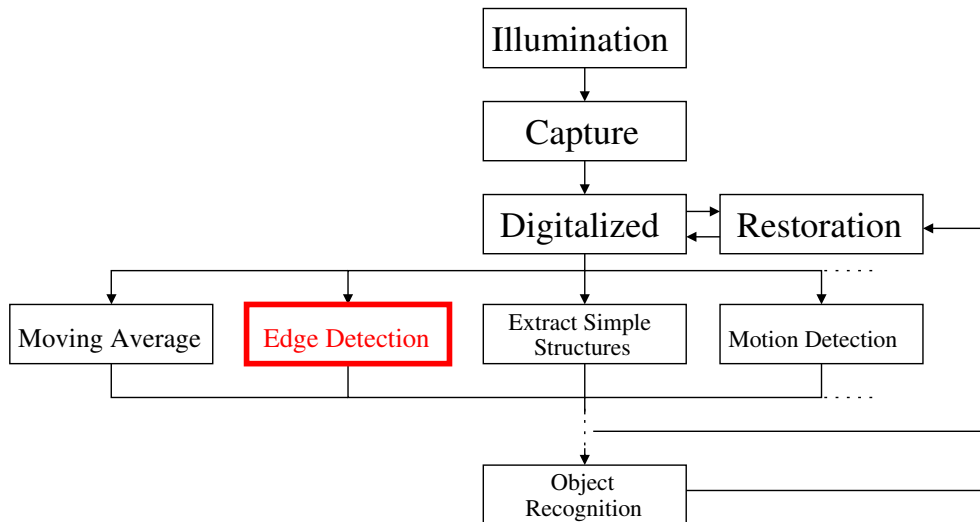


Figure 1.1: Simplified hierarchy of digital image processing

In this thesis various edge detection algorithms are implemented with a FPGA in real-time. Chapter 2 gives an insight into transmission systems and system theory. In chapter 3 different optical filters are described (e.g. low-pass and high-pass). Chapter 4 explains the basic of real-time systems. A short introduction of FPGAs is given in chapter 5 where a FPGA is compared with a DSP. Chapter 6 and chapter 7 show the hardware and the implementation of this thesis. And in chapter 8 I will conclude and discuss my results.

2 Transmission Systems

2.1 Signal and System Theory

The system theory describes the interrelationship between an input signal $s_1(x)$ and an output signal $s_2(x)$ in a random transmission system. The input signal is called *cause* and the output signal is called *effect*.

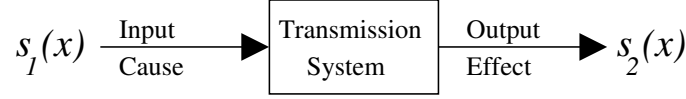


Figure 2.1: Transmission system model

An operation G (e.g. derivative) is applied to the signal $s_1(x)$ in the transmission system:

$$s_1(x) \rightarrow s_2(x) = G\{s_1(x)\} \quad (2.1)$$

Transmission systems can be divided into different classes:

- **Time-invariant:**

A system $s_1(x) \rightarrow s_2(x)$ is time-invariant when $s_1(x + x_o) \rightarrow s_2(x + x_o)$ also holds.

- **Linearity:**

A system is linear when $G\{u(x) + a \cdot v(x)\} = G\{u(x)\} + a \cdot G\{v(x)\}$ holds (Superposition and Proportionality).

- **Causality:**

A system is causal when the effect always occurs after the cause. An acausal system is a prophet.

Transmission systems that are linear and time-invariant are called *LTI* systems and play an important role in practice. In figure 2.2 we can observe the superposition and proportionality principle of linearity. The signal $s_{2b}(x)$ consists of particular effects of signal $s_{2a}(x)$.

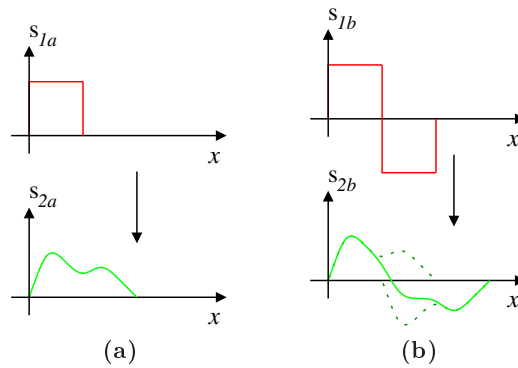


Figure 2.2: Example for a LTI system

2.2 Dirac Delta

The dirac delta is a mathematical construct which represents a function $\delta(x)$ that has the value zero everywhere except at $x = 0$ where its value is infinitely large.

$$\delta(x) = \begin{cases} +\infty, & x = 0 \\ 0, & x \neq 0 \end{cases} \quad (2.2)$$

The total integral of $\delta(x)$ is defined as follows:

$$\int_{-\infty}^{\infty} \delta(x) dx = 1 \quad (2.3)$$

Another property of dirac delta is:

$$\delta(x) = \delta(-x) \quad (2.4)$$

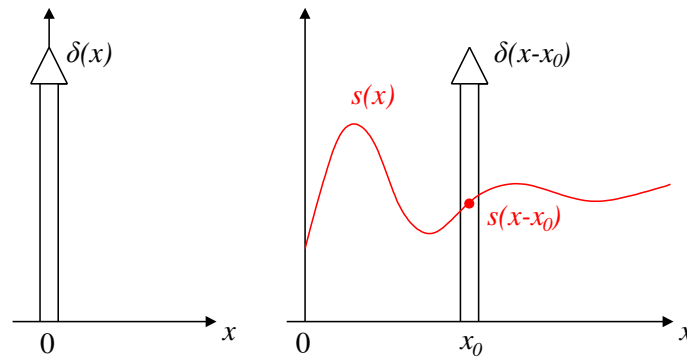


Figure 2.3: Dirac delta and its sifting property

The dirac delta has a sifting property as shown in equation 2.5. The function $s(x)$ is multiplied with zero except the point where the dirac impulse is. Therefore it is possible to replace $s(x)$ with $s(x_0)$ and move it in front of the integral. The area of the dirac delta is always equal to one, independent of x_0 . So the dirac delta sifts the value of $s(x)$ at the point of x_0 .

$$\int_{-\infty}^{\infty} s(x) \cdot \delta(x - x_0) dx = s(x_0) \underbrace{\int_{-\infty}^{\infty} \delta(x - x_0) dx}_{=1} = s(x_0) \quad (2.5)$$

2.3 Impulse Response

The response of a LTI system which was excited with a dirac delta $\delta(x)$ is called *impulse response* $h(x)$. Every LTI transmission system is fully characterized by its impulse response $h(x)$. With the knowledge of the impulse response $h(x)$ it is possible to calculate every response on any excitation. The operator for this calculation is called *convolution*. Systems with a finite impulse response are called *FIR filter*.

2.4 Convolution

Operations which transform an input signal by means of a mathematical mapping into an output signal are called *filter*. For any input sequence $s_1(x)$, the output sequence $s_2(x)$ can be calculated by the convolution of $s_1(x)$ and $h(x)$. Convolution is one of the

most significant operations in signal processing. For continuous signals the convolution is defined as:

$$s_2(x) = s_1(x) * h(x) = \int_{-\infty}^{\infty} h(\xi) \cdot s_1(x - \xi) d\xi \quad (2.6)$$

In signal processing the convolution mask $h(x)$ is usually zero except for a small region at the origin. The convolution leads to a new signal $s_2(x)$ that represents a weighted averaging of $s_1(x)$ in a small environment of x . For instance the new signal can be more smooth as the input signal $s_1(x)$.



Figure 2.4: LTI system with its impulse response $h(x)$

The method to estimate the impulse response $h(x)$ is to use the dirac delta $\delta(x)$ as the input of the system. Here is the proof:

$$\begin{aligned}
 s_1(x) &= \delta(x) \\
 s_2(x) &= \delta(x) * h(x) \\
 s_2(x) &= \int_{-\infty}^{\infty} h(\xi) \cdot \delta(x - \xi) d\xi \\
 s_2(x) &= h(x) \cdot \int_{-\infty}^{\infty} \delta(x - \xi) d\xi \\
 s_2(x) &= h(x) \quad q.e.d
 \end{aligned} \quad (2.7)$$

2.5 Discrete Convolution

For discrete applications as digital signal processing the discrete convolution is defined as:

$$s[n] = \sum_{\nu=0}^{N-1} h[\nu] \cdot g[n - \nu] \quad (2.8)$$

For two-dimensional discrete applications as digital image processing the discrete convolution is defined as:

$$s[m, n] = \sum_{\mu=0}^{M-1} \sum_{\nu=0}^{N-1} h[\mu, \nu] \cdot s_1[m - \mu, n - \nu] \quad (2.9)$$

Example: 1-D Discrete Convolution

Given is the signal $s_1[x] = \{1, 2, 1\}$ and the filter function $h[x] = \{-1, 0, 1\}$:

$s_1[x] :$	1	2	1				$h[x] :$
	-1	-2	-1				$\cdot (-1)$
		0	0	0			$\cdot 0$
			1	2	1		$\cdot 1$
$s_2[x] :$	-1	-2	0	2	1		

2.6 Frequency Domain

With the Fourier transform it is possible to estimate the frequency parts from a random signal as shown in equation 2.13. Considering a LTI system in the frequency domain is more obvious because the convolution corresponds to a multiplication (cp. equation 2.11).

$$s(t) \circ \bullet S(f) = \int_{-\infty}^{\infty} s(t) \cdot e^{-j2\pi ft} dt \quad (2.10)$$

$$s_2(t) = s_1(t) * h(t) \circ \bullet S_2(f) = S_1(f) \cdot H(f) \quad (2.11)$$

The transformation of the impulse response $h(x)$ is called *transfer function* $H(f)$. For discrete application the Fourier transform is defined as:

$$s[n] \circ \bullet S[\nu] = \frac{1}{N} \sum_{\nu=0}^{N-1} s[n] \cdot e^{-\frac{j2\pi n\nu}{N}} \quad (2.12)$$

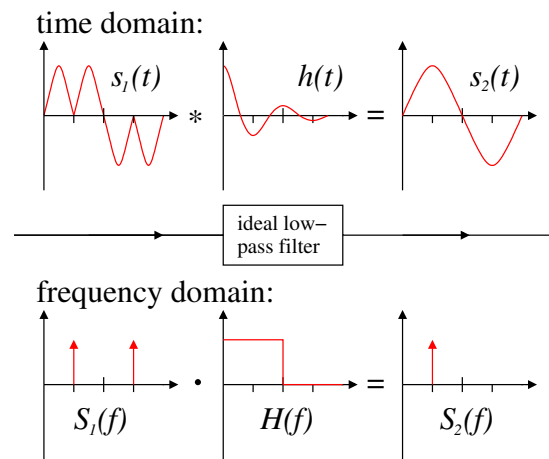


Figure 2.5: Frequency and time domain of an ideal low-pass filter

2.7 Fourier Optics

For two-dimensional applications the discrete Fourier transformation is defined as:

$$s[n, m] \circ \bullet S[\nu, \mu] = \frac{1}{NM} \sum_{\nu=0}^{N-1} \sum_{\mu=0}^{M-1} s[n, m] \cdot e^{-\frac{j2\pi n\nu}{N}} \cdot e^{-\frac{j2\pi m\mu}{M}} \quad (2.13)$$

Figure 2.6 shows an example for the transformation of an image in Fourier domain and applying a directional filter. This directional filter lets only vertical structures pass.

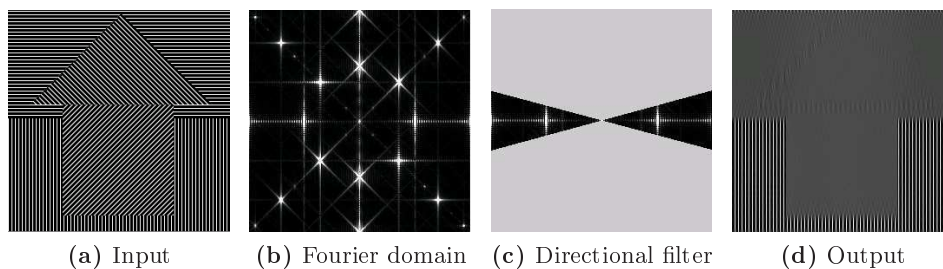


Figure 2.6: Example for Fourier filtering

3 Optical Filters

3.1 Neighbourhood Operators

To extract attributes of an image analyzing each pixel individually is not enough. It is important to consider adjacent pixel as well. Neighbourhood operators combine adjacent pixels in a small area and compute a new image. After applying a neighbourhood operator, information is lost and you cannot conclude the original image anymore. For this reason neighbourhood operators are called filters.

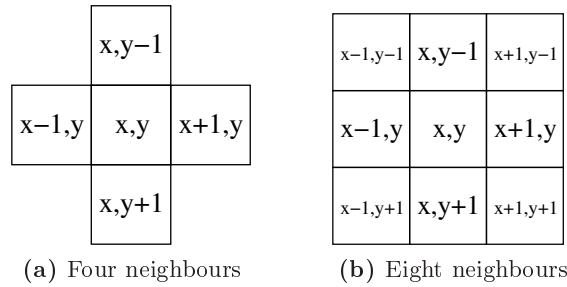


Figure 3.1: Neighbourhood operators

These operators are able to do the following:

- Detection of local structures as edges or lines
- Reconstruction, correction and restoration
- Averaging and diffusing
- Morphology

The neighbourhood operator moves pixel-by-pixel and line-by-line over the input image I and computes the output image O by convolution or sorting as shown in figure 3.2. When the neighbourhood operator is overhanging the border of the image, some pixel values are missed. There are three methods to compensate the missing pixels:

- Use the pixels from opposite border (cyclic)
- Set the value to zero
- Extrapolate an value by considering the border pixels

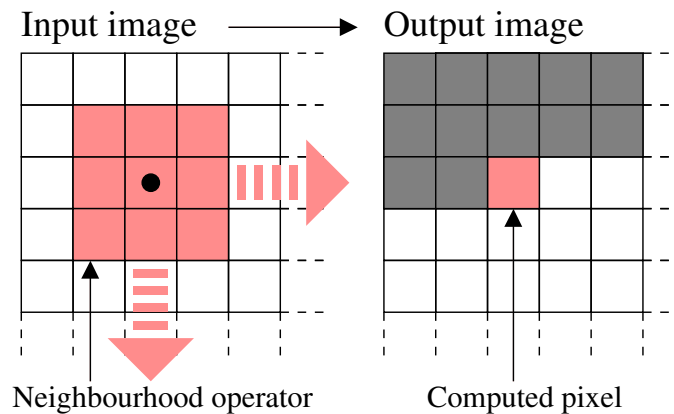


Figure 3.2: Applying a neighbourhood operator to an image. (grey = already computed)

3.2 Linear Filters

3.2.1 Moving Average - Low-Pass Filters

Moving average filters also known as blur filters compute an average of pixels in a small area. Structures with a high frequency as edges are blurred so the sharpness of the whole image is reduced. All averaging filters are low-pass filters. A common application for these filter type is suppression of Gaussian noise. The assumption for a blur filter is that the sum of all coefficients of the operator must be equal to one:

$$\sum_m \sum_n h[m, n] = 1 \quad (3.1)$$

If there is a single corrupted pixel received, a blur filter will include and enhance this error in adjacent pixels. There are also non-linear approaches known as rank filters. For instance the median filter which suppresses single corrupted pixels. (See chapter 3.3 Non-Linear Filters)

3.2.1.1 Box Filter

The box filter is the simplest low-pass filter. Box filters add up all pixels in the neighbourhood and divide the result by the count of pixels. For example a 3×3 convolution operator which can be separated into two 3×1 operators:

$$R_{3 \times 3} = \frac{1}{9} \cdot \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} = \frac{1}{3} \begin{bmatrix} 1 & 1 & 1 \end{bmatrix} * \frac{1}{3} \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \quad (3.2)$$

The transfer function of a box filter is a *sinc*(x, y) function. The attenuation does not increase monotonically with higher frequency but oscillates. That shows that the filter passes some high frequencies as well. The filter is easy to implement but in certain cases not very effective. Increasing the mask size of the operator will lead to a decreasing of the cutoff frequency.

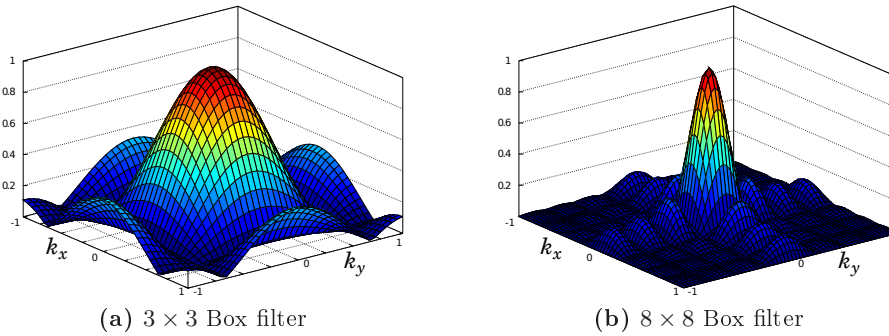


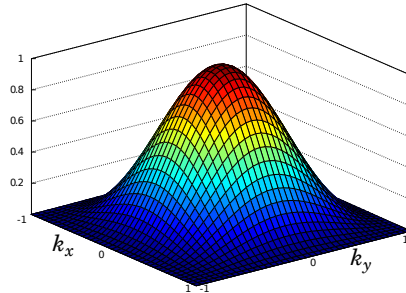
Figure 3.3: Transfer functions of box filters

3.2.1.2 Binominal Filters

A small box filter mask will lead to a bad transfer function and a good transfer function will lead to a big filter mask. For an ideal low-pass the filter mask is even infinitely large. Filter design is a complex optimization problem so compromises must be found. A more efficient method than box filters are binominal filters which are based on the values of the discrete binomial distribution. The values of this distribution can be calculated with the triangle of Pascal as shown in Figure 3.4a.

n	f(n)	b(n)
1	1	1
2	1/2	1 1
3	1/4	1 2 1
4	1/8	1 3 3 1
5	1/16	1 4 6 4 1
...	...	...

(a) Pascal's Triangle



(b) Transfer function of binomial filter

Figure 3.4: Binominal Filter results of Pascal's Triangle

For example a 3×3 operator which can also be separated into two 3×1 operators:

$$B_{3 \times 3} = \frac{1}{16} \cdot \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} = \frac{1}{4} \begin{bmatrix} 1 & 2 & 1 \end{bmatrix} * \frac{1}{4} \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} \quad (3.3)$$

The transfer function of a binominal filter attenuates and does not oscillate so this approach will produce better results than a simple box filter. In general the binominal filter can be estimated as follows:

$$B_{n \times n} = f^2(n) \cdot [b(n) * b(n)] \quad (3.4)$$



(a) Grey scale image with noise



(b) Box filter



(c) Binominal filter



(d) Median filter

Figure 3.5: Averaging with low-pass filters, and the non-linear median filter

3.2.2 Edge Detection - High Pass Filters

Edge detection filters extract structures with high frequency namely edges. An edge occurs when the intensity values of adjacent pixels differ strongly. All edge detection approaches are high-pass filters. Also in this case there are several possibilities to design such a filter. The task of edge detection requires neighbourhood operators that are sensitive to changes and suppress areas of constant grey values. The assumption is that the sum of all coefficients of the operator must be equal to zero:

$$\sum_m \sum_n h[m, n] = 0 \quad (3.5)$$

The basic idea is to observe the first or second derivative of the input image. A minimum or maximum in the first derivative or x-interception in the second derivative corresponds to an edge.

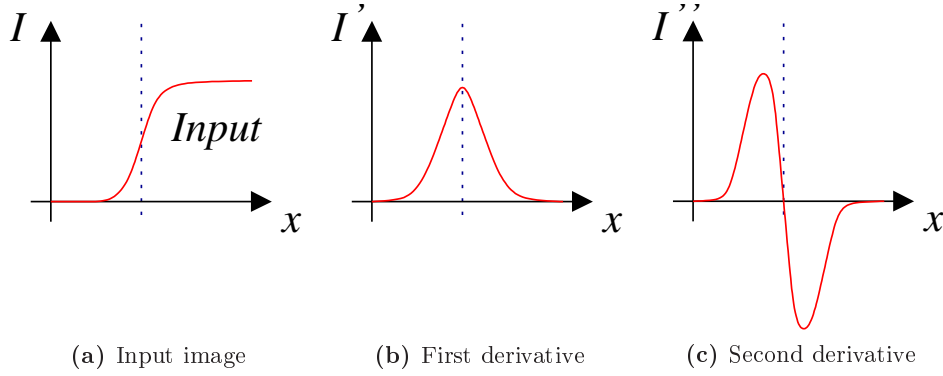


Figure 3.6: 1d-image function and its derivatives

In two-dimensional space an edge is a maxima of the magnitude of the gradient vector. Therefore first-order derivatives in all directions must be computed.

$$D_{xy} = \begin{bmatrix} D_x \\ D_y \end{bmatrix} \quad (3.6)$$

$$|D_{xy}| = \sqrt{D_x \cdot D_x + D_y \cdot D_y} \approx |D_x| + |D_y| \quad (3.7)$$

The direction of the edge can also be computed with:

$$\Theta = \arctan\left(\frac{D_y}{D_x}\right) \quad (3.8)$$

We work with discrete systems, so we have discrete differences which can only approximate a continuous differentiation. The errors associated with these approximations require careful consideration. A serious error is the anisotropic edge detection i.e. edges are not detected equally well in all directions.

3.2.3 Difference Approach

The simplest approach to detect edges is to subtract the current pixel from its predecessor. It is similar to a convolution with a $\delta(x)$ dirac doublet.

$$\dot{f}(x) = f(x) * \dot{\delta}(x) \quad (3.9)$$

As already explained we have to compute these subtractions in all directions. The output is the sum of the absolute values of the convolutions. The convolution operators are:

$$D_x = \frac{1}{2} \begin{bmatrix} 1 & -1 \end{bmatrix} \quad D_y = \frac{1}{2} \begin{bmatrix} 1 \\ -1 \end{bmatrix} \quad (3.10)$$

$$O = |I * D_x| + |I * D_y| \quad (3.11)$$

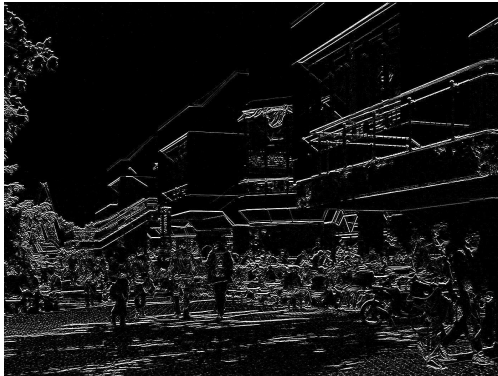
This filter will not detect diagonal edges and is not very noise resistant as the transfer function in Figure 3.10c shows. We have to find better approaches.



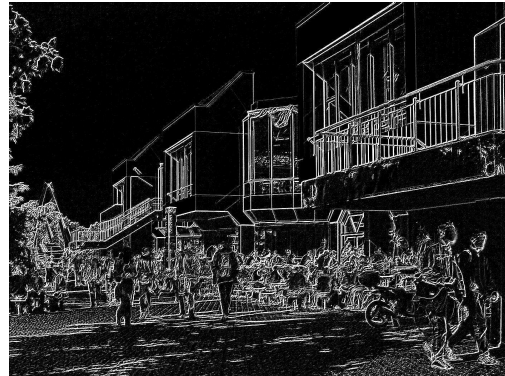
(a) Grey scale image



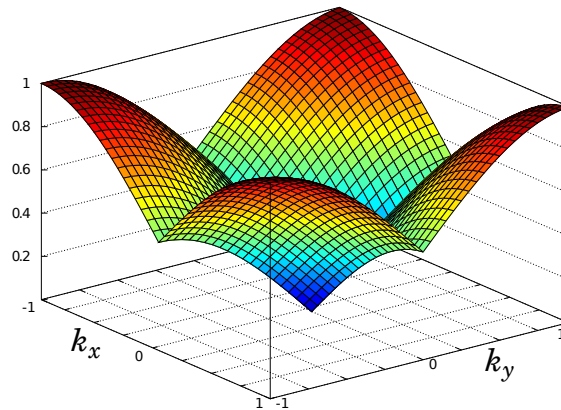
(b) Convolution with x-operator



(c) Convolution with y-operator



(d) Sum of absolute values



(e) Transfer function of the filter

Figure 3.7: Simple difference approach

3.2.4 Robert Cross Operator

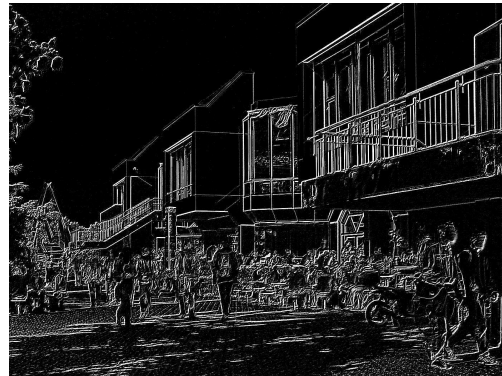
This cross operator was introduced by L. G. Roberts in 1965 [4]. The operator computes the differences between diagonally adjacent pixels. The filter delivers better results as our first approach but it is not very noise resistant either.

$$D_1 = \frac{1}{2} \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \quad D_2 = \frac{1}{2} \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} \quad (3.12)$$

$$O = |I * D_1| + |I * D_2| \quad (3.13)$$



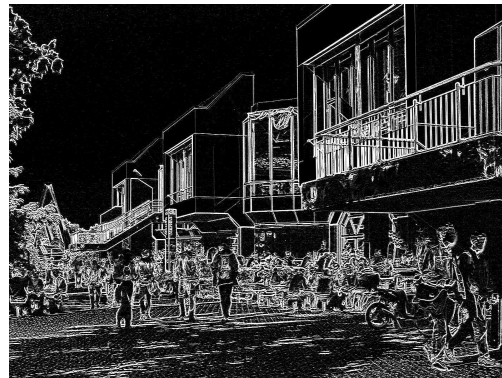
(a) Grey scale image



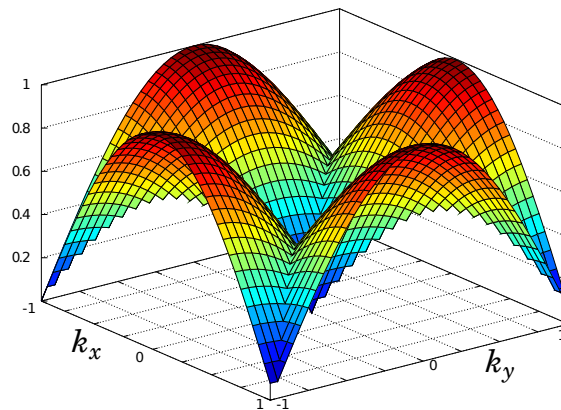
(b) Convolution with x-operator



(c) Convolution with y-operator



(d) Sum of absolute values



(e) Transfer function of the filter

Figure 3.8: Roberts edge detector

3.2.5 Sobel Operator

The Sobel Operator is a very good approach for edge detection because it is a noise resistant operator. The operator was presented by I. Sobel in 1968. The transfer functions show a band-pass characteristic which suppress high frequency noise. The edges are detected in all directions.

$$D_x = \frac{1}{8} \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} \quad D_y = \frac{1}{8} \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \quad (3.14)$$

$$O = |I * D_x| + |I * D_y| \quad (3.15)$$



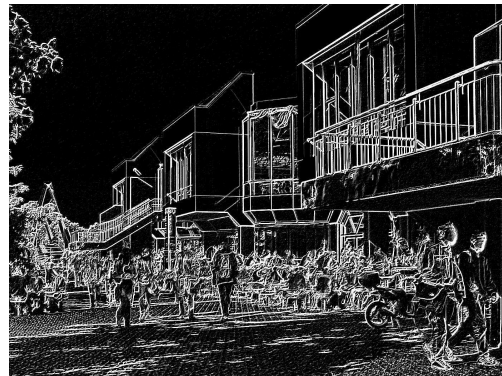
(a) Grey scale image



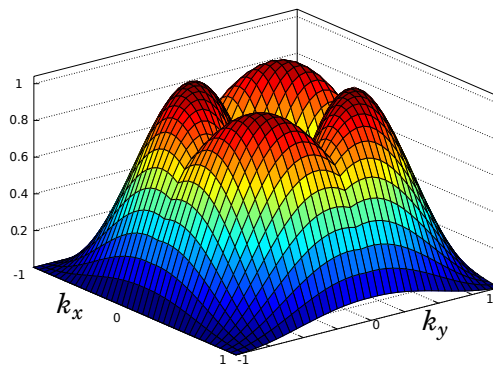
(b) Convolution with x-operator



(c) Convolution with y-operator



(d) Sum of absolute values



(e) Transfer function of the filter

Figure 3.9: Sobel edge detector

3.2.6 Laplace Operator

In contrast to the already discussed edge detectors the Laplace operator uses zero crossings in the second derivative. But it is obvious that not every zero crossing is an edge, it could also be noise. An edge occurs only when there are peaks before and after a zero crossing that are significantly higher than the noise level. The Laplace operator is very noise sensitive.

We can conclude the Laplace operator from applying the first derivative two times:

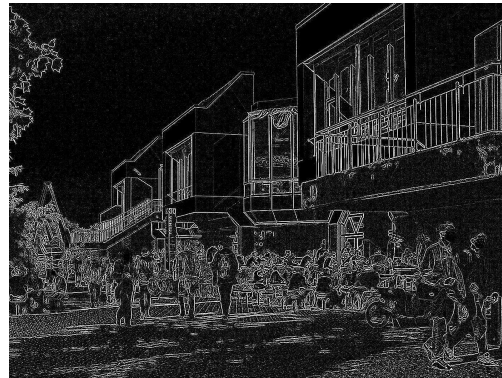
$$\frac{1}{2} \cdot [1 \ -1] * \frac{1}{2} \cdot [1 \ -1] = \frac{1}{4} \cdot [1 \ -2 \ 1] \quad (3.16)$$

$$D = \frac{1}{8} \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix} = \frac{1}{4} [1 \ -2 \ 1] * \frac{1}{4} \begin{bmatrix} 1 \\ -2 \\ 1 \end{bmatrix} \quad (3.17)$$

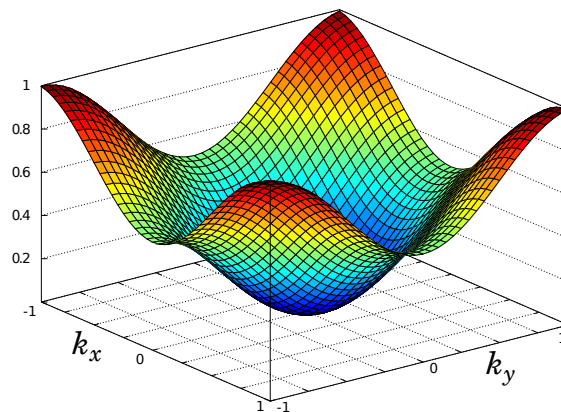
$$O = |I * D| \quad (3.18)$$



(a) Grey scale image



(b) Convolution with Laplace operator



(c) Transfer function of the filter

Figure 3.10: Laplace edge detector

3.3 Non-Linear Filters

With convolution we have learned how to weight and add pixels in a neighbourhood. There are also other operators which compare and select adjacent pixels.

3.3.1 Rank Value Filter

A rank value filter sorts the values in the neighbourhood operator in ascending order. If the values are sorted, one value of this list is used for the output image. When the value in the middle of the list is used, the filter is called *median filter* and if the value at the end or at the beginning is used, the filter is called *maximum filter* or *minimum filter*.

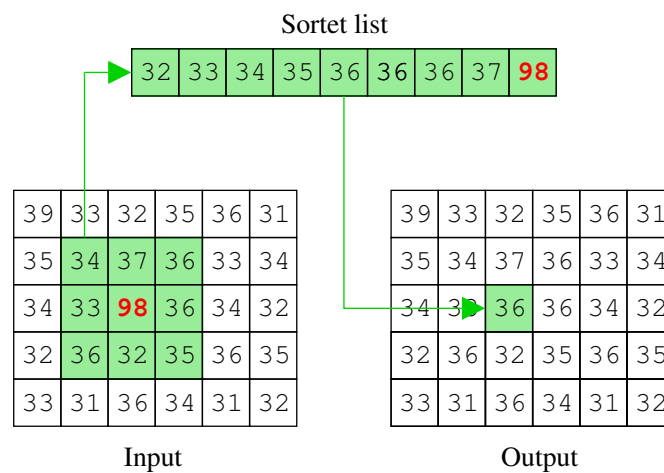


Figure 3.11: Median filter, compare *Digital Image Processing* [1]

The median filter is a good approach to remove noise which appears due to transmission errors. A moving average filter would enhance this error in the adjacent pixels. But the median approach could reintroduce other errors.

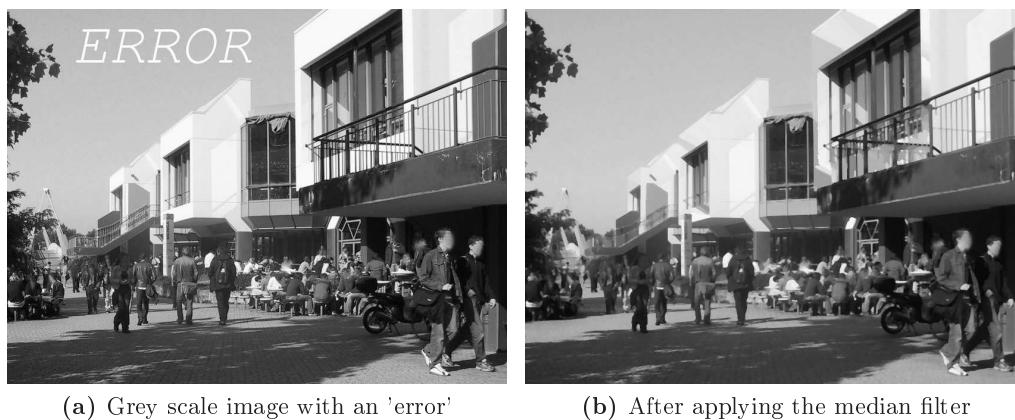
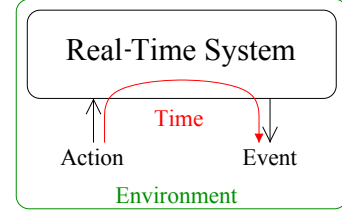


Figure 3.12: Median filter

4 Real-Time Systems

Real-time computing plays an important role for controlling systems. Examples of applications that require real-time computing are nuclear power plants, automotive electronics, airplanes and so forth. A real-time system is a system that reacts to events in its environment by performing predefined actions within specified time intervals. The correctness of results depends on the value and its time of delivery. Deliver the correct value too late (or too early) is wrong and can lead to tremendous consequences (e.g. someone could die). For *hard* real-time applications it is very important that the deadlines are met. There are also *soft* real-time applications where the miss of a single deadline does not jeopardize the behaviour of the system (e.g. frame dropping in video applications). In standard personal computer systems the computer controls the speed of the user who has to wait for the results of computation, but in the real-time world the environment, which can not be halted, controls the speed of the system. For this thesis I will explain only the basic items of real-time and scheduling.



4.1 Scheduling

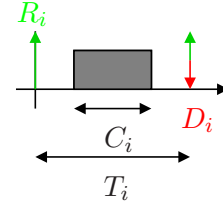
To explain the basic ideas of scheduling we have to define a task i :

T_i : Period of task i .

R_i : Release time, relative from start of period time at which task i may start to execute.

D_i : Deadline, relative from start of period time at which task i must be completed (often the deadline is equal to the period).

C_i : Worst case execution time (the longest time a program could execute).



The Utilisation of a single task i is defined as:

$$U_i = \frac{C_i}{T_i} \quad (4.1)$$

The Utilisation U of the whole system within all tasks is defined as:

$$U = \frac{C_0}{T_0} + \frac{C_1}{T_1} + \dots + \frac{C_{n-1}}{T_{n-1}} = \sum_{i=0}^{n-1} \frac{C_i}{T_i} \quad (4.2)$$

A task set is *schedulable* when all tasks meet their deadlines during the lifetime of the task set / system. A necessary condition for schedulability is that $U \leq 1$. If this condition fails we know that the task set cannot be schedulable, but if it holds it does not mean automatically that it will be schedulable.

There are further conditions for different scheduling procedures. If we want sureness we have to do an exact test. All scheduling tests are based on the *critical instance* approach. The *critical instance* is one point in time where all tasks become ready to execute simultaneously. This is the worst case that can happen to the system and so it is a good starting point for an exact test. The point in time when the scheduling begins to repeat is called *hyperperiod* which is also important to know. This point can be calculated through the least common multiple of the periods T_i . Now we can simulate the scheduling from the critical instance for one hyperperiod. If there are no conflicts in this simulation the system is schedulable.

5 FPGAs

A FPGA (field-programmable gate array) is an integrated circuit which is designed to be configured by the hardware designer after the manufacturing (programmable logic). The FPGA configuration is generally specified using a hardware description language like VHDL or Verilog. Compared to application-specific integrated circuits (ASICs), they are slower and require more power but for low output figures they are much cheaper. It is possible to reconfigure the FPGA or update the functionality after shipping (banana hardware - the hardware matures at the customer).

The advantages of FPGAs are:

- Very fast for computing intensive application due to parallelized computation
- Very flexible due to reprogrammable configurable logic blocks (CLB)
- Many arbitrary definable input and output pins (> 1000)
- Less redesigns of circuit boards
- Useful for rapid prototyping
- Integration of IP-blocks possible (e.g. CPU)

A FPGA is often a very good choice instead using a DSP (digital signal processor). For a FIR filter implementation with eight coefficients a DSP needs eight clock cycles. A FPGA only needs one clock cycle for all eight operations.

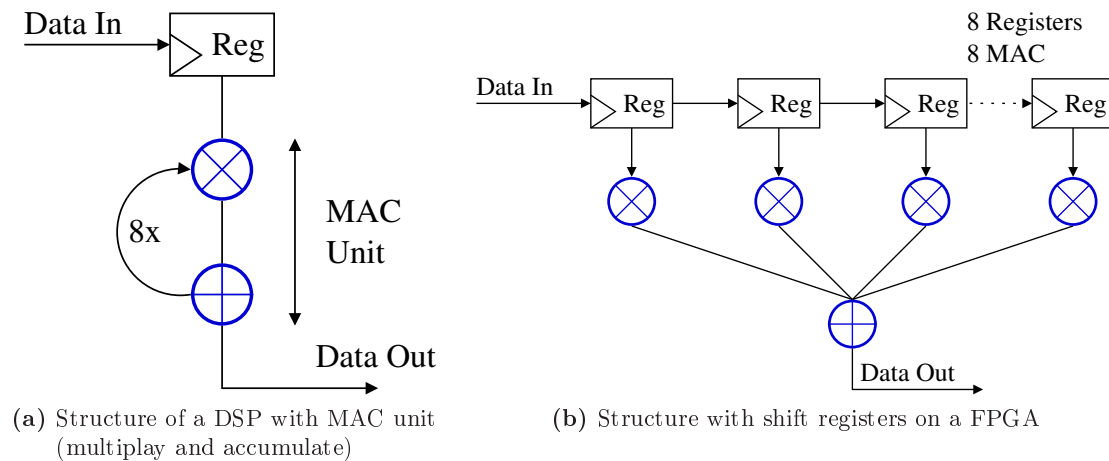


Figure 5.1: FIR filter with eight coefficients

A FPGA consists of a matrix of configurable logic blocks (CLBs) connected via programmable interconnects as shown in Figure 5.2. The CLB is the basic logic unit in a FPGA. The exact numbers and features vary from device to device, but every CLB consists of a configurable switch matrix and comprises e.g. four *slices*. In general a slice contains look-up tables, multiplexers, registers and combinatorial logic.

The designer describes the logical behaviour of the desired system in a hardware description language like VHDL. The *synthesis tool* optimizes the hardware description with different algorithms and heuristics with respect to various constraints (e.g. timing constraints). After the synthesis the required components are known and the *place and route tool* can map the components on the FPGA.

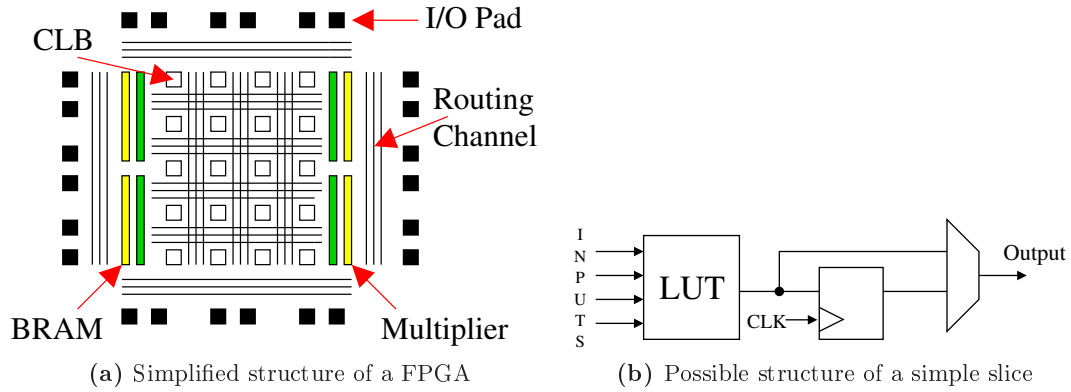


Figure 5.2: Structural principle of a FPGA

In this thesis a Xilinx FPGA is used. These FPGAs use look-up tables (LUTs) to implement the logic equations and a SRAM to save its configuration. A look-up table with k inputs can implement 2^{2^k} different functions. Some other manufacturers use multiplexers instead of look-up tables. When the FPGA is powered off, its configuration is lost because SRAM is a volatile memory. After powering on the FPGA it needs to be reprogrammed which can be done by an external flash memory.

6 Hardware

The task of this thesis was to implement the previously described roberts, sobel and laplace edge detection filters on a FPGA in real-time with a camera. The real-time constraint is that the input image must be processed and displayed before the next input image arrives from the camera. The underlying hardware consists of a video camera and the FPGA board *Euterpe* which was developed by my tutor Matthias Alles [6]. The output happens on a VGA monitor where all filter results can be observed concurrently.

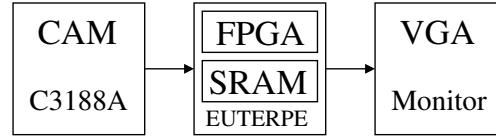


Figure 6.1: Abstract Hardware Components

6.1 Camera

The Camera which is used is the Omnivision OV7620 CMOS VGA Camera [8]; it has a resolution of 640×480 pixels and delivers the data in the Y'UV format with 25 FPS.



The Y'UV model defines a colour space in terms of one luminance (Y') and two chrominance (UV) components. For this project only the luminance values are considered. The Camera uses the *Zoomed Video Port* (ZV Port) as output interface. The ZV Port is a single-source uni-directional video bus which is normally used between a PC Card socket and a VGA controller. The ZV Port complies with CCIR601 standard timing to allow decoders to deliver real-time digital video straight into the VGA frame buffer from a PC Card. The picture is delivered in even and odd lines which

can be evaluated by the signal FODD. The signal VSYNC signals the beginning of a new picture. The signal HREF signalizes the submission of a line. If the HREF signal is set to one, a new Y-value arrives every pixel clock PCLK ($74ns$). The timing diagrams are shown in Figure 6.3. For this application a picture with the size of 320×240 is required. So the input image must be resized by using only every second pixel of the even image.

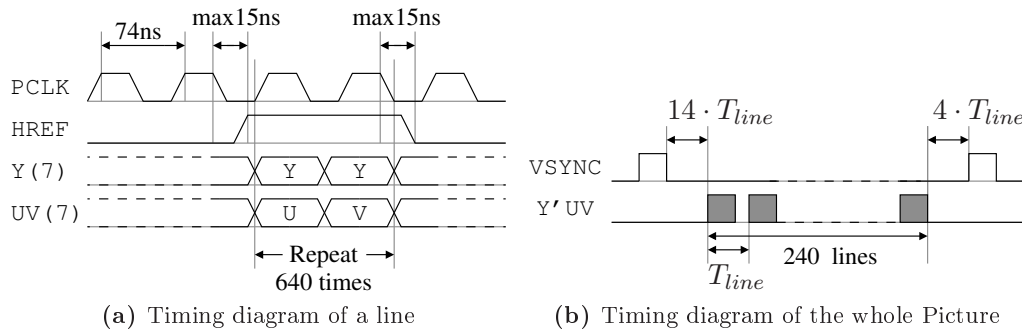


Figure 6.3: Camera timing diagrams, redraw from the datasheet [8]

6.2 FPGA Board - Euterpe

The Euterpe FPGA Board was developed by Matthias Alles. The main part of the Board is a Xilinx Spartan 3 XC3S400 FPGA with 3584 slices (with 4-input LUTs), 16 multipliers and 16 block rams. The Board has the following specifications [6]:



Figure 6.4: The board *Euterpe*

- Xilinx Spartan 3, XC3S400 in a PQ208 package
- 4 Mbit configuration Flash for the FPGA
- 18Mbit pipelined, synchronous, ZBT SRAM, organized as 512k*36 using a Cypress chip (CY7C1370CV25). The SRAM theoretically operates with up to 167MHz (leading to a maximum bandwidth of 667MB/s)
- 4Mbit serial Flash
- 2 VGA-ports, one with a 15 bit (3*5 bit for RGB) and one with a 9-bit(3*3 bit) resolution. Both are realised with R-2R-resistor-ladders
- 1 RS232-interface, realised with a MAX3232. This chip operates with up to 250 kbit/s
- 2 PS/2-ports for mouse and keyboard
- 2 stereo AD-converters and 1 stereo DA-converter
- 2 expansion connectors, one is meant for the digital video camera while the other one has a better pin layout (both connectors use the same FPGA-I/Os)
- 2 buttons and 1 free programmable LED

For this thesis the video camera expansion connector is used. The SRAM serves as video memory for the *framebuffer* entity and the output is done with the 15 bit VGA output port.

7 Implementation

The Task of this thesis was to display different edge detection outputs on a monitor. As shown in figure 7.1 the roberts, sobel and laplace output can be observed in parallel. The resolution of each filter output is 320×240 so the resolution of the whole output image is 640×480 . A greater resolution is not possible due to the limited size of the SRAM. The camera input image of the camera (640×480) must be resized to 320×240 by using only every second pixel of the even image.

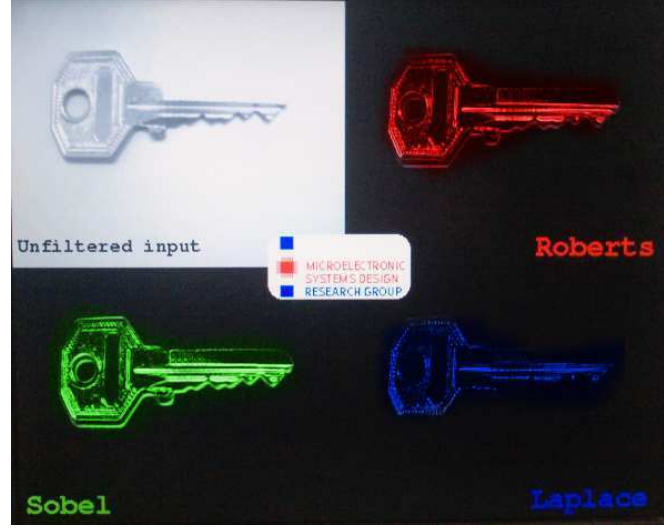


Figure 7.1: Picture of the running system

Figure 7.2 shows the structure of the system. Every box represents a VHDL entity. The green numbers are the period times of data arrival. In the following I will explain each component of the system in detail.

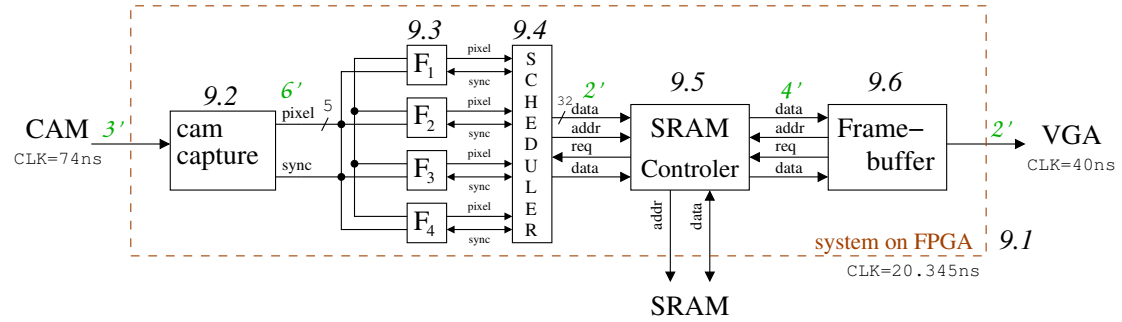


Figure 7.2: Top-level of the system (the chapter numbers refer to the according source code)

7.1 Cam Capture

Every $74ns$ the camera sends a pixel to the cam capture entity. The FPGA clock has a value of $20.345ns$ so the pixels arrive asynchronously. It can be estimated roughly that every third clock a new pixel arrives at the entity:

$$\left\lfloor \frac{74ns}{20.345ns} \right\rfloor = \lfloor 3.63733 \rfloor = 3 \quad (7.1)$$

The *cam capture* entity regards only even pictures and every second pixel so a new pixel arrives every sixth clock at the filter components F . The arrival of a new pixel is signaled with the *sync* signal.

7.2 Filter

An one-dimensional FIR filter can be realised with shift registers for the time delay, multipliers for the coefficients (which represent the convolution operator) and adders.

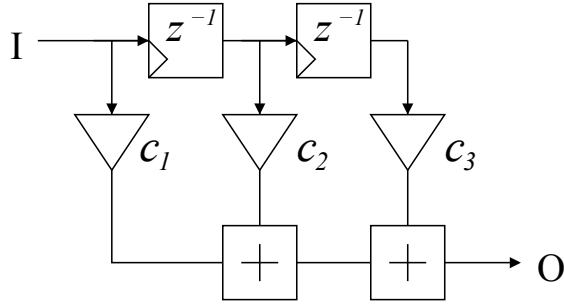


Figure 7.3: Structure of a simple FIR filter

To implement two-dimensional convolution filters in software is very simple. Only two nested loops are needed. In a streamed hardware application other approaches are necessary. A structure for two-dimensional approaches can be derived easily from the simple FIR filter above (Figure 7.3).

The structure of each filter is similar, only the operator is different. The filter entity is constructed for 3×3 neighbourhood operators and consists of four block RAMs and four shift registers as shown in figure 7.4. To compute an output value the convolution operator needs three lines buffered. There is always one block RAM for storing new pixels and the output is generated by means of the three other block RAMs in parallel.

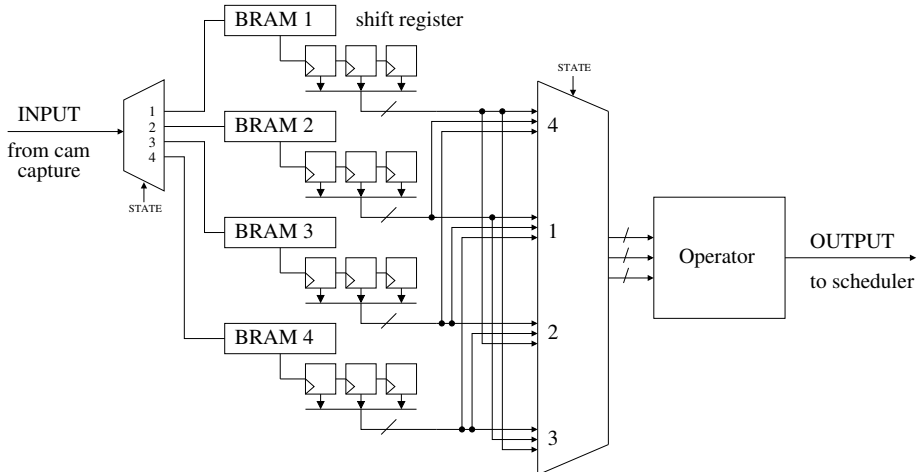


Figure 7.4: Structure of the filter entity

Every sixth clock a pixel arrives at the input and is written into a particular block RAM. The other block RAMs deliver the saved values to the shift registers which form the neighbourhood for the operator unit. The operator unit can calculate a result each clock cycle so a line with calculated pixels can be bursted out to the scheduler. The pixels where the neighbourhood operator overhangs are extrapolated as explained in chapter 3.1.

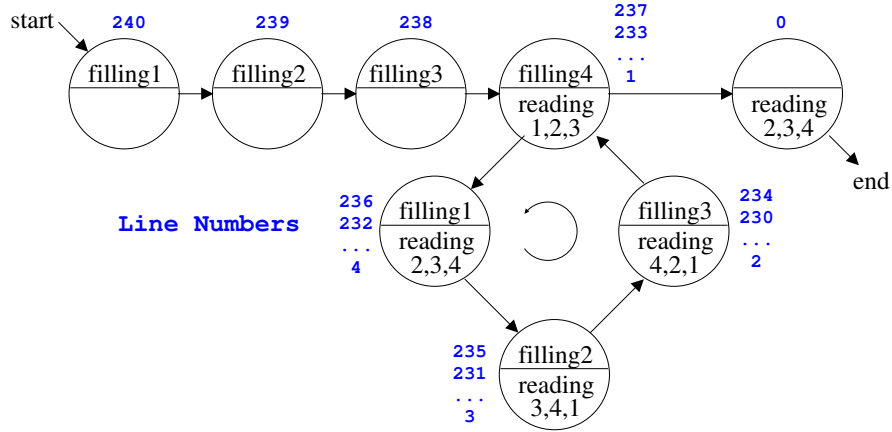


Figure 7.5: Automaton which describes the reading and writing of different block RAMs

Figure 7.5 describes the order of events in the filter entity. At the beginning three lines are buffered and then the calculation of output begins during the next block RAM is filled with data.

The filling process is slower (every 6th clock) than the reading process (every clock). So the *scheduler* can burst out the data of 4 filters to the *SRAM controller* sequential and the *cam capture* entity can fill the block RAM with a new line at the same time.

7.2.1 Dirac

The Dirac filter relays the pixels from the input to the *scheduler* over two block RAMs without any calculation. This leads to a grey value image on the VGA monitor.

7.2.2 Roberts Filter

From the neighbourhood operators of the Roberts filter (Equation 3.12), a characteristic function can be derived. The inputs of the operator are the signals: $a_1, a_2, a_3, b_1, b_2, b_3, c_1, c_2, c_3$.

$$O = |a_1 - b_2| + |a_2 - b_3| \quad (7.2)$$

This function leads to the following Hardware implementation:

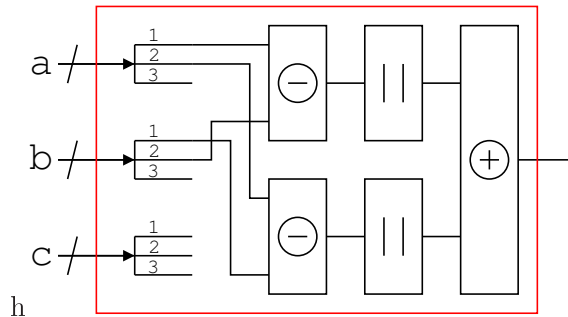


Figure 7.6: Hardware for the Roberts filter

7.2.3 Sobel

The characteristic function of the Sobel approach can be derived from Equation 3.14:

$$O = |a_1 + 2 \cdot b_1 - (1_3 + 2 \cdot b_3 + c_1)| + |a_1 + 2 \cdot a_2 + a_3 - (c_1 + 2 \cdot c_2 + c_3)| \quad (7.3)$$

The multiplication with two is realised with a binary shift so a multiplier is not necessary.

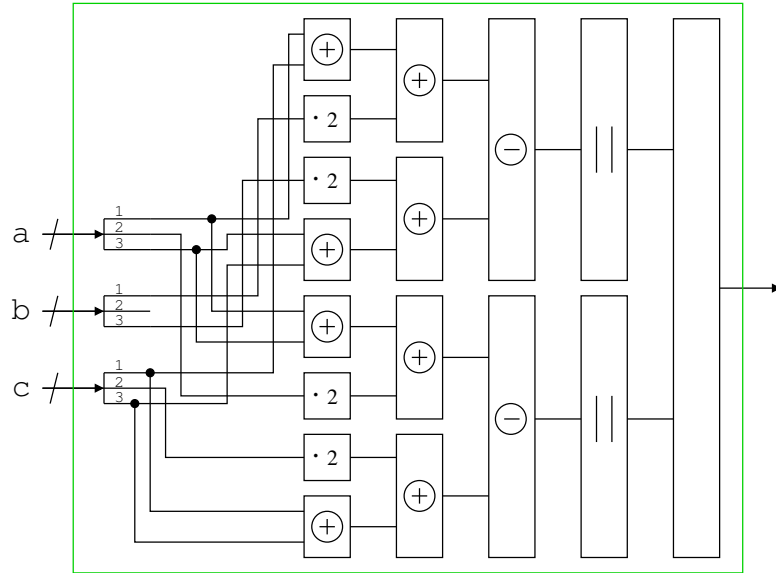


Figure 7.7: Hardware for the Sobel filter

7.2.4 Laplace

The characteristic function of the Laplace filter can be derived from Equation 3.17:

$$O = |a_2 + b_1 + b_3 + c_2 - 4 \cdot b_2| \quad (7.4)$$

The multiplication with four is realised with two left shifts.

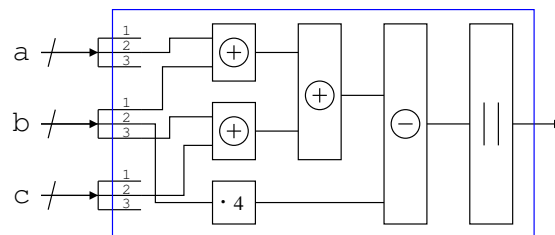


Figure 7.8: Hardware for the Laplace filter

7.3 Scheduler

The scheduler processes each filter one after another and requests the result with the *req* signal. If the filter is ready (has three lines in buffer) and can begin with computing, it signals that with the *valid* signal. The computed line is bursted out and the *scheduler* requests the *SRAM controller* for memory access.

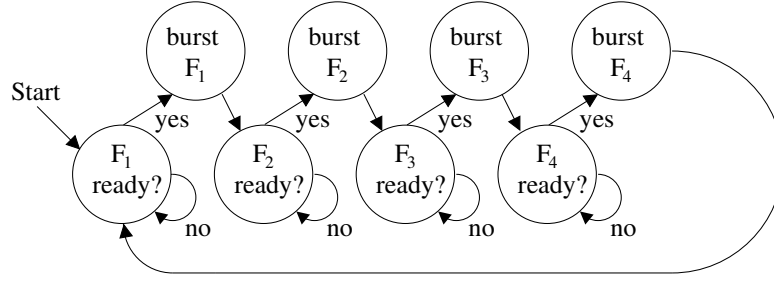


Figure 7.9: Automaton of the Scheduler

Every second clock the scheduler writes two pixel values to the *SRAM Controller* because the SRAM word consists of two pixels:

RRRRR & GGGGG & 0 & BBBBB & RRRRR & GGGGG & 0 & BBBBB

If the *SRAM* is already in use by the *framebuffer*, the scheduler reattempts a clock cycle later. In chapter 7.5 the reattempts are explained in detail.

7.4 Framebuffer

The *framebuffer* entity was provided by my tutor Matthias Alles. I enhanced it with a additional caption layer. The entity displays the content of the SRAM to the VGA port. The *framebuffer* uses the industrial standard VGA timing which was initially developed for cathode ray tubes. This standard and the connectors are so ubiquitous that even liquid crystal displays provide a legacy VGA control interface.

In a tube monitor an electron beam is shot on a screen that is coated with a material that emits light when the beam hits it. The intensity of the beam and the position where it hits the screen can be controlled through deflection coils. An image consists of many scan lines which specify the intensity. The beam follows each scan line from left to right and then moves rapidly to the starting point of the next scan line below. While moving to the next scan line, the beam is turned off (horizontal blanking). After the last scan line the beam returns to the topleft corner of the screen (vertical blanking). There are three analogue signals for the scanlines (red, green, blue) that provide the intensity of the beam. There are also two other digital signals HSYNC and VSYNC that signal the end of a scan line and the end of an image. [11]

Screen refresh rate	60Hz
Vertical refresh	31.46875kHz
Pixel freq.	25.175MHz

(a) General timing

Scanline part	Pixels	Time [μs]	Part	Lines	Time [ms]
Visible area	640	25.422045680	Visible area	480	15.253227408
Front porch	16	0.6355511420	Front porch	10	0.3177755710
Sync pulse	96	3.8133068520	Sync pulse	2	0.0635551142
Back porch	48	1.9066534260	Back porch	33	1.0486593843
Whole line	800	31.777557100	Whole frame	525	16.683217477

(b) Horizontal timing

(c) Vertical Timing

Figure 7.10: VGA timing values for a resolution of 640×480 pixels [10]

The required pixel frequency for the VGA output is 25MHz and the system frequency is 50MHz . So the *framebuffer* halves the system clock to get the pixel frequency. Every second clock cycle a pixel is displayed on the VGA output. But the *framebuffer* can read two pixel values from the *SRAM Controller* at once so every fourth clock cycle data is requested from the *SRAM*.

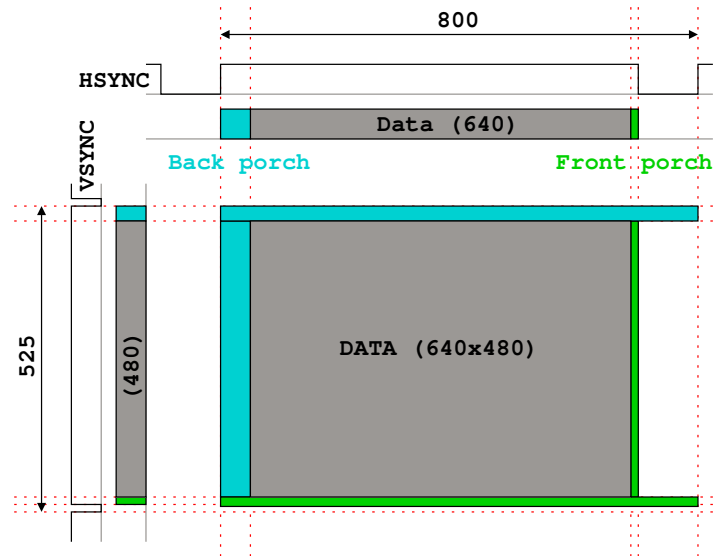


Figure 7.11: VGA timing

Figure 7.10 and 7.11 show the industrial standard timing for an 640×480 image. In this thesis the timing is roughly fulfilled but the most VGA monitors can handle small discrepancies.

The *framebuffer* also provides an additional layer for captions (compare figure 7.1). The captions for the individual filters and the logo of the chair of microelectronics are stored here as bit arrays. If a pixel is output, the caption layer is checked at the pixel position. If there is an entry in the bit array the pixel is transposed.

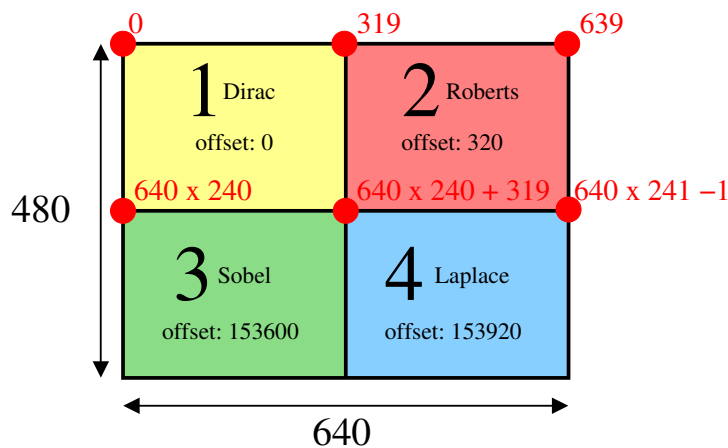


Figure 7.12: Structure of the output picture with memory addresses

Figure 7.12 shows the arrangement of the different filter outputs in the output image. Every filter entity has an offset that determines the position of the filter.

7.5 SRAM Controller

The SRAM is the bottleneck of the system because it is only a single port RAM. The access to the SRAM must be controlled anyway. The *SRAM controller* entity provides a simple interface for the control. It has ports for the *scheduler* and for the *framebuffer* but the port for the *framebuffer* has a higher priority. That means that the access for the *framebuffer* is always granted and the *scheduler* has to retry its access a clock cycle later if it is necessary.

This bottleneck refers to the real-time constraint that the input image must be processed and displayed before the next input image arrives from the camera. Before I wrote the source code I proved that it is feasible to have an error-free SRAM access.

7.5.1 Proof: Is the SRAM access scheduling feasible?

The proof is very easy by means of real-time tools. For the real-time model we assume two tasks. The reading of the SRAM from the *framebuffer* entity Task₀ and the writing from the *scheduler* Task₁. The *framebuffer* has a higher priority as the *scheduler* so $P_0 = 1$ and $P_1 = 0$. The worst case execution time for each task is one clock cycle $C_1 = C_2 = 1'$. The period of Task₁ is $T_1 = 4'$ (Chapter 7.4) and the period of Task₀ is $T_0 = 2'$ (Chapter 7.3).

		Task ₀ <i>framebuffer</i>	Task ₁ <i>scheduler</i>
Priority	P	1 (high)	0 (low)
WCET	C	$1'$	$1'$
Period	T	$4'$	$2'$

Figure 7.14: Overview of scheduling parameters

The first step is to prove that the necessary condition, that the utilisation of the system is not higher than 100%, holds. If the necessary condition does not hold, the system is definitely not schedulable.

$$U = \sum_{i=0}^n \frac{C_i}{T_i} \leq 1 \quad \rightarrow \quad U = \frac{1}{4} + \frac{1}{2} = 0.75 < 1 \quad \checkmark \quad (7.5)$$

Now we know that the system might be schedulable. The next step is to do the exact test for one hyperperiod. The hyperperiod can be estimated as the least common multiple of T_0 and T_1 which is $H = 4$. Now we can observe the Scheduling at the critical instance for one hyperperiod:

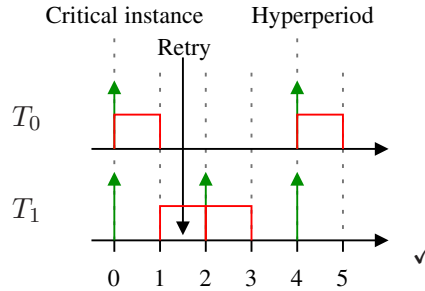


Figure 7.15: Exact scheduling test

→ There are no conflicts in the test so the task set is schedulable.

7.6 Synthesis

The synthesis has produced the expected structure of the system (compare Figure 7.2):

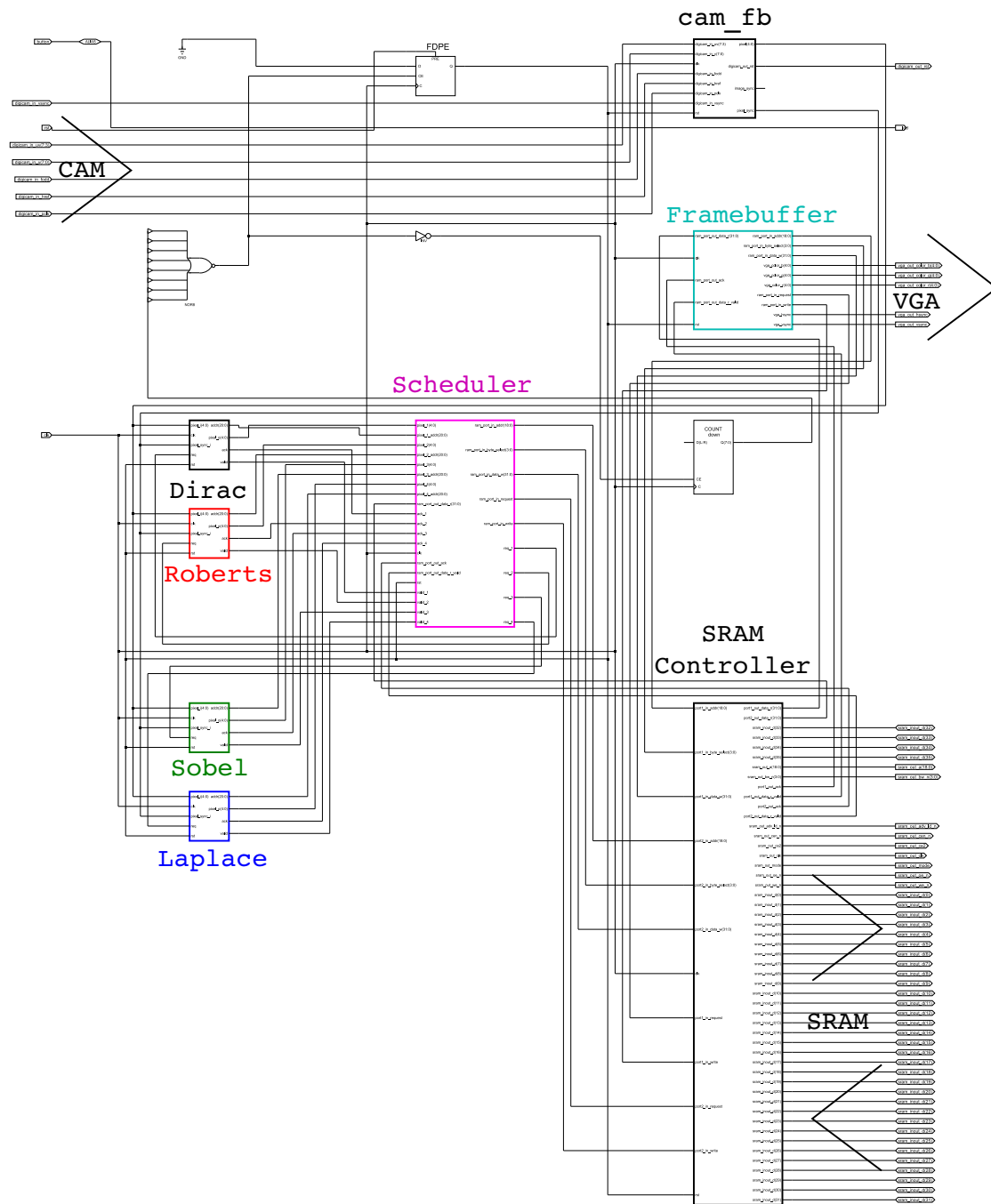


Figure 7.16: Structure after Synthesis

Adders/Subtractors:	51			
Counters:	18			
Comparators:	36			
Number of Slices:	1529	out of	3584	42%
Number of Slice Flip Flops:	900	out of	7168	12%
Number of 4 input LUTs:	2779	out of	7168	38%
Number used as logic:	2748			
Number used as Shift registers:	31			
Number of IOs:	105			
Number of bonded IOBs:	96	out of	141	68%
Number of BRAMs:	14	out of	16	87%
Number of GCLKs:	1	out of	8	12%

Figure 7.17: Device utilization summary of the synthesis report

The synthesis report shows us that the system uses less than the half of available slices but almost all BRAMs.

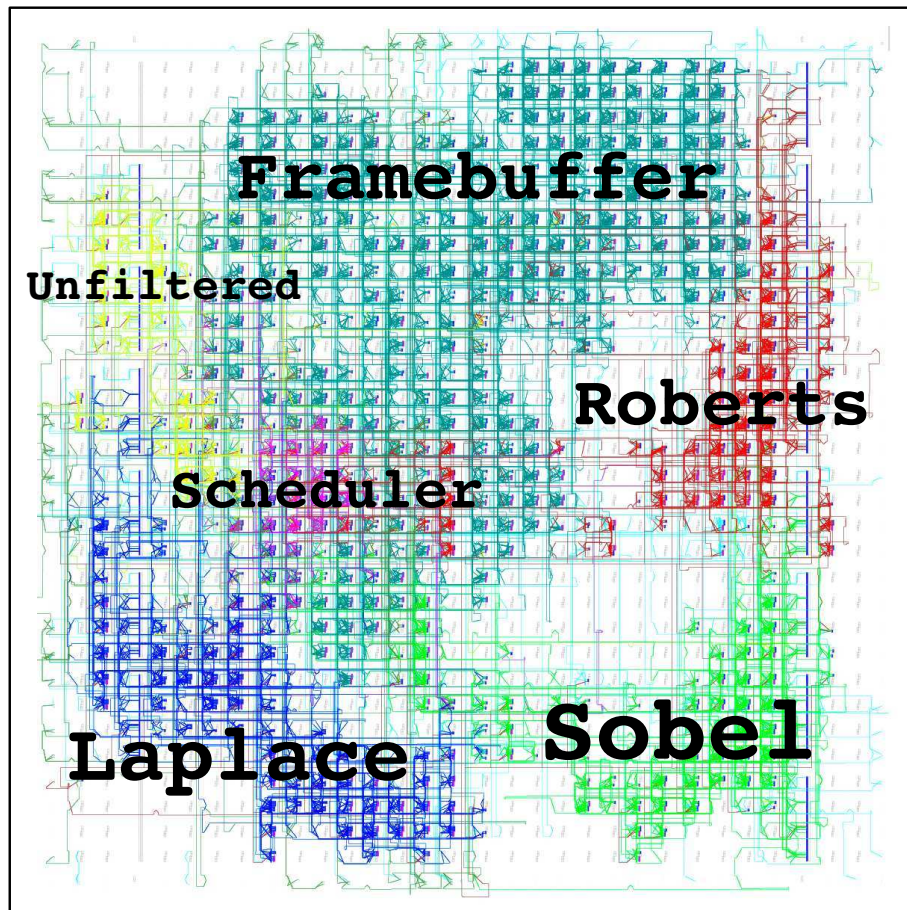


Figure 7.18: Routing on FPGA

The picture above shows us the routing on the FPGA. All filter components are located around the *scheduler* and as supposed near to the BRAMs. The *framebuffer* takes much space because the captions for the display output are saved in this entity as bit arrays which are distributed to LUTs. The *roberts* and *dirac* filters are as expected smaller than *roberts* and *laplace* due to their reduced operator size.

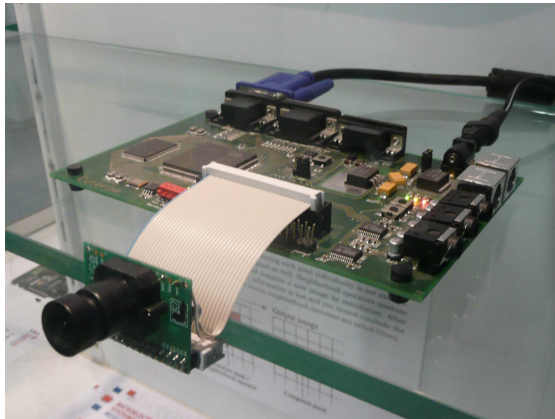
8 Conclusion

In this thesis miscellaneous edge detectors were implemented on an embedded system with a FPGA. Edge detection is an important step in the digital image processing hierarchy because the data rate is drastically reduced. Following units like object recognition obtains only relevant data.

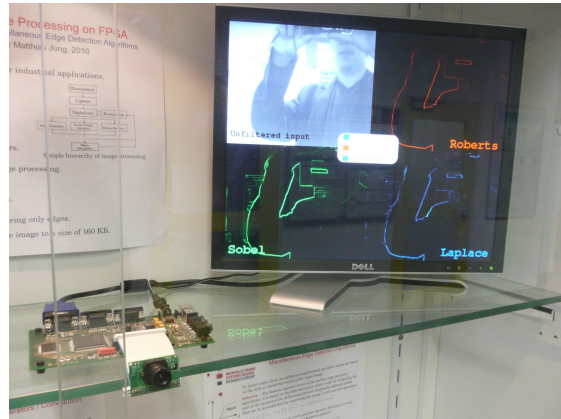
The Roberts, Sobel and Laplace two-dimensional filters were implemented by deriving a filter structure from a one-dimensional FIR filter. The operators can be easily modified so the implementation of other filters (e.g. low-pass) is possible. Also the implementation of a non-linear filter like the *median filter* is possible with a pipelined comparator network. It was necessary to buffer lines from the camera to apply the neighbourhood operators. The bottleneck of the system is the SRAM access, but in this thesis it was proved that there will be no conflicts in the scheduling.

All filters have their advantages and disadvantages. The Roberts and Laplace filter are easy to implement but they are not resistant to noise. The Sobel filter suppresses noise and extracts the edges well. This is a good compromise which leads to useful results. The Sobel filter is often used in industrial applications.

For the future it would be useful to implement morphology and object recognition algorithms on FPGA.



(a) The board Euterpe with camera



(b) The whole system

Figure 8.1: The hardware is exhibited in building 11

9 Appendix

References

- [1] Bernd Jaehne, "Digitale Bildverarbeitung", Springer Verlag, 2005
- [2] Werner Rupprecht, "Signale und Uebertragungssysteme", Springer Verlag , 1993
- [3] Giorgio C. Buttazzo, "Hard Real-Time Computing Systems", Springer, 2005
- [4] Lauwrence Gilbert Roberts, "Machine Perception of Three.Dimensional Solids", MIT, 1963
- [5] Peter J. Ashenden, "The designer's guide to VHDL", Morgan Kaufmann, 2008
- [6] Matthias Alles, "Euterpe, FPGA board",
URL: <http://www.student.uni-kl.de/~alles/fpga/about.htm> (Fetched 22.03.2010)
- [7] Ecole Polytechnique Federale de Lausanne, "Fourier Filtering Applet", URL:
<http://bigwww.epfl.ch/demo/fourierfilter/> (Fetched 22.03.2010)
- [8] OV7620 SINGLE-CHIP CMOS VGA COLOR DIGITAL CAMERA DATASHEET,
"Omnivision"
- [9] Image of Camera, URL: <http://www.asahi-net.or.jp/~qx5k-iskw/robot/image.html>
(Fetched 30.03.2010)
- [10] VGA Industry standard timing, URL: <http://tinyvga.com/vga-timing/640x480@60Hz> (Fetched 06.04.2010)
- [11] Kolja Sulimma, Digital Technology Lab University of Kaiserslautern, Assignment 8:
"Controlling a CRT Display"