FYS-3921

Master's Thesis In Electrical Engineering

Time and Temperature Control of a Camera

Anders Kristiansen

July 12, 2010

Faculty of Science and Technology

Department of Physics and Technology

University of Tromso

Abstract

In scientific experiments there are an increasing usage of cameras, along with other instruments, to observe parameters of physical processes. In this thesis, we will discuss how a microcontroller (ATMEL ATmega 324 PA) can be used to control the operation temperature in a camera.

It will be discussed why this control of temperature can be useful for the quality of the pictures. Since this thesis is a practical type, we will develop an active cooling system that handles the temperature control. The thesis will include descriptions of these hardwares, which has been developed to achieve this cooling.

When we use cameras in scientific experiments we rely on strict and accurate time control of exposure time in the camera. We will make a trigger system to check the stability and precision of three different trigger sources.

Preface

This Master's thesis is written as a part of my education in electrical engineering at The University of Tromsø (UiT). The thesis is a part of the course FYS-3921, and is weighted with 30 points.

First I want to thank my supervisor Professor Torsten Aslaksen who has been at service along the way. I also want to thank divisional engineer Karl Magnus Fossan who have been helping me with equipment that was needed along the way. He took time out of his day to explain why and how the electronic circuits can catch noise from the surroundings. He also explained the engineer way to solve these noise problems. I also want to thank Allied vision tech. support that helped me understand more about the Guppy F044 NIR, especially when the data sheet didn't give answers.

Finally, I want to thank Astrid and Åge Kristiansen that have been motivating me when I was writing. I also want to thank Heidi Dreyer Hansen for helping me with my English grammar, Linda Dreyer Hansen and Arnulf Hansen for a lot of good food and Tinka for nice walks during the process. To all others who in one way or another have contributed to this Master's thesis, thank you.

Tromsø July 10. 2010
Anders Kristiansen

Contents

Chapter 1

Introduction

Aerial photography is a technology, that is mainly used in cartography to construct topographic maps. It is useful for site planning, archaeology, movie production, environmental studies, surveillance, commercial advertising, conveyancing and artistic projects. [7] We want to check if it is possible to make an automated system that can start taking pictures in presence of an external trigger.

1.1 Problem

Cameras are used more rapidly in sientific experiments, where they are mostly used to observe parameters in physical processes. If we want an accurate comparison with observations done by other cameras and/or other types of instruments, a strict control of exposure time on the camera is required. Therefore we will study the precision and stability of three different trigger sources, one software trigger and two external triggers.

When a camera takes a picture, the quality of that picture is very dependent on the temperature of the sensor chip in the camera. If we want good quality on the data from the sensor chip, it is prefered that the sensor chip has a temperature as close as possible to the lower limit of the operative temperature to the camera. This can be solved with an active cooling device. In this thesis we will measure the temperature in the camera, and cool it if necessary. We will also discuss the improvement of this active cooling device.

1.2 Goal

We need to develop hardware that will be used to measure the temperature inside the camera. The precision of this hardware needs to be at least 0.1 degrees Celsius. We use the cooling system to control the operational temperature inside the camera and hold it stabil around 0-1 °C.

We have to develop software that can control the camera, the microcontroller and show the quality of the pictures as we vary the temperature in the camera. We will check which other sources can influence the quality of the pictures.

We will use one software trigger source, a computer, two different external trigger sources, a microcontroller and a frequency generator. This is to check the precision and stability when we use an external trigger compared to a software trigger.

1.3 Structure

The structure of the thesis is as follows:

- Chapter 2 will discuss the system design of both the cooling system and the trigger system.
- Chapter 3 will discuss the control unit and its internal ADC (Analog to Digital Converter).
- Chapter 4 will discuss the camera and different noise sources that could occure in it.
- Chapter 5 will discuss the design of the thermometer and do the necessary calculations to get the best possible resolution.
- Chapter 6 will discuss the design of the cooling system and the necessary calibrations.
- Chapter 7 will discuss the quality of the pictures.
- Chapter 8 will discuss the possibilities we have when we want to use an external trigger on the camera.
- Chapter 9 will discuss the precision and stability of three different trigger sources.
- Chapter 10 will discuss the results we got in this thesis.
- Chapter 11 will draw a conclusion from this thesis.

Chapter 2

System Design

The automated system that will be made in this master thesis consists of two smaller systems, one cooling system and one trigger system. The automated system will take pictures at a predefined camera temperature.

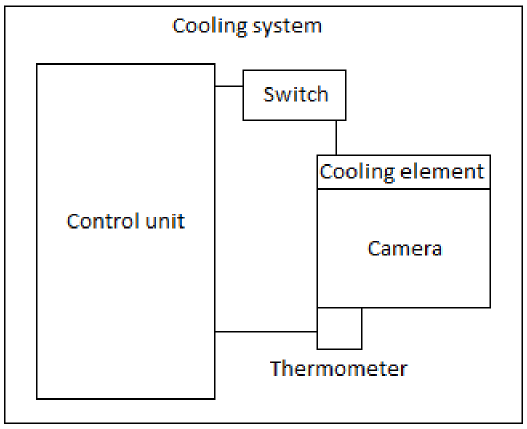


Figure 2.1: This is a sketch of the cooling system. The control unit gets an input from the thermometer and will compute an output from this. The output signal will turn the switch on or off. The cooling element is controlled by the switch and will cool the temperature inside the camera.

In figure 2.1 we see the layout of the cooling system. In the figure we see that the thermometer is placed on the camera to measure the cameras temperature. The thermometer provides the control unit with a voltage value. The control unit can convert this analog voltage into a digital number. After the control unit has converted the analog voltage, it will know if the camera is too hot and needs cooling. The control unit will turn on the switch when the camera is too hot and the cooling element will recieve power to start the cooling process.

The trigger system is supposed to use either a software trigger source or an external trigger source. We will use this trigger system to check the precision and stability of three trigger sources. In figure 2.2 we see the layout of the trigger system.

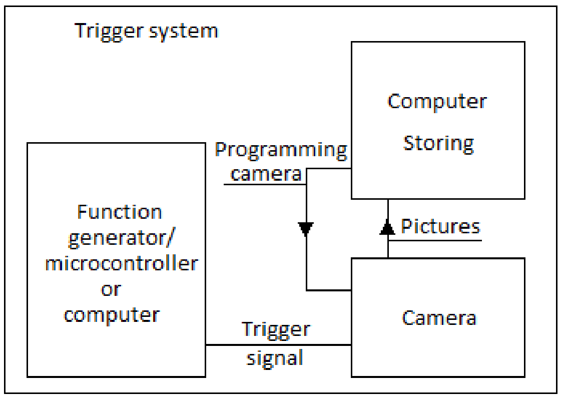


Figure 2.2: This is a sketch of the trigger system. The camera is programmed by a computer. The camera then recieve a trigger signal from either a function generator, microcontroller or a computer. After the picture is taken it is sent to the computer from the camera.

We've got a computer to program the camera. When the camera is programmed it can understand what trigger source it is going to recieve a signal from. It will also know the exposure time, gain and how many pictures it is going to take. The computer is receiving and storing the pictures after they have been taken. One at a time, the three trigger sources will provide the camera with a trigger signal.

Chapter 3

Control Unit

The microcontroller used in this thesis is an ATMEL ATmega 324PA. The microcontroller from ATMEL has a built-in Analog-to-Digital Converter (ADC). In the figure 3.2 on the following page we see the block diagram of this Analog-to-Digital Converter. [4]

We can see from the chart that the ADCSRA, ADCSRB and ADMUX are the registers that manage the components in the ADC. These registers can be used to control the ADC. For instance, they can control when to start a conversion or if the ADC should be enabled. These registers helps the analog voltage from the thermometer to be converted into a digital value through this ADC.

We are now going to study these registers, in figure 3.1 we see how the representation of the bits is in each register.

Bit	7	6	5	4	3	2	1	0

Figure 3.1: Description of the Bit-representation of the ADMUX, ADCSRA and ADCSRB registers.

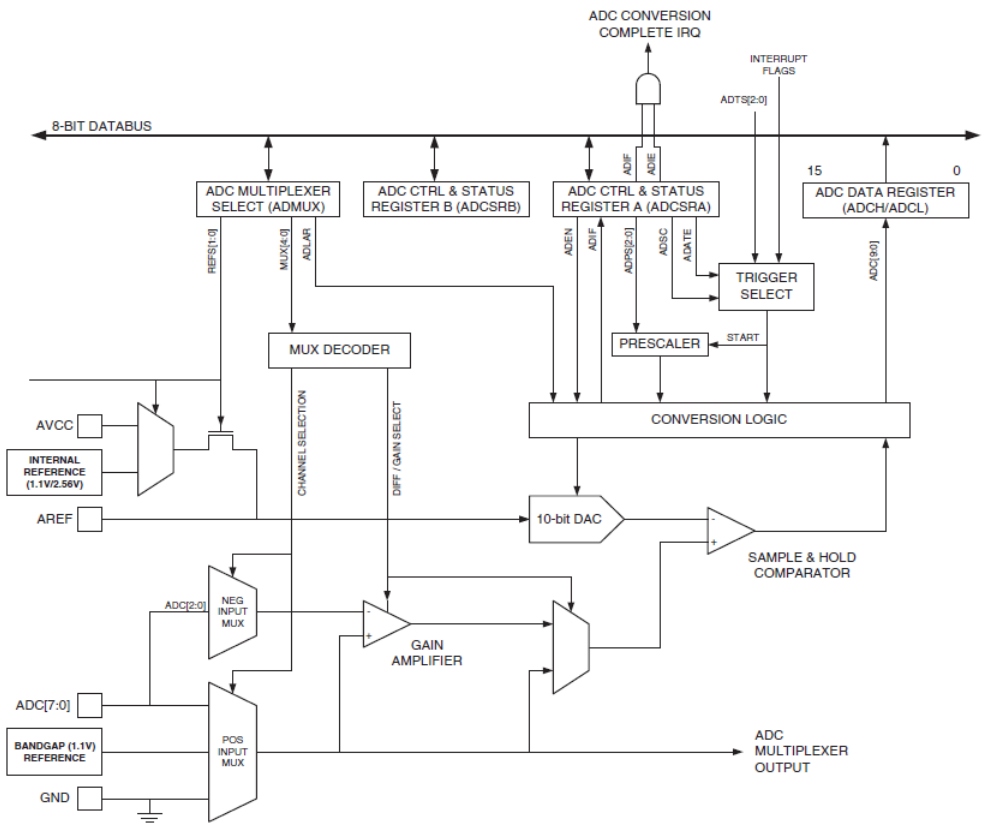


Figure 3.2: Analog-to-digital Converter Block Schematic. This is a block diagram of the ADC that is inside the microcontroller

3.1 ADCSRA - ADC Control and Status Register A.

Bit 0:2 are the ADC Rescale Select Bits. These bits determine the division factor between the crystal frequency on the card and the input clock on the ADC. The input clock on the ADC will determine how accurate the measurements will be. [4]

$$\frac{\text{Crystal frequency}}{\text{Division factor}} = \text{ADC input clock} \tag{3.1}$$

Bit 3 represents the ADC Interrupt Enable. When this bit is written to one you have activated the ADC's interrupt mode. [4]

Bit 4 refers to ADC interrupt flag. This bit is set when a conversion completes and the data registers are updated. This interrupt is executed if the ADC interrupt is enabled. [4]

Bit 5 is the ADC Auto Trigger Enable bit. This bit decides whether the ADC Auto Trigger is active or not. The trigger source is selected by setting the ADC Trigger Select bits in ADCSRB (discussed in section 3.2). [4]

Bit 6 tells the ADC to start a conversion, this is for the case when you use single conversion mode. In single conversion mode, you need to write this bit to one in your program to start each conversion. In free running mode this will be automated, but it will take more time to convert one input because the microcontroller need to set all the values itself. When we use free running mode this bit will exchange between one and zero, since it tells us if the ADC is converting or not. [4]

Bit 7 decides if the ADC is on or off. When this bit is written to one it enables us to use the ADC, and if this bit is zero the ADC will be switched off. [4]

3.2 ADCSRB - ADC Control and Status Register B.

Bit 0:2 refers to ADC auto trigger source. This are the bits that selects which trigger source we want to use after setting the ADC auto trigger enable bit in ADCSRA. In table 3.1 on the following page we can see the different sources we can choose from. [4]

Bit 3:7 are reserved bits, which we can't control. These bits are for future use in the ATmega 324PA. These bits must be written to zero when ADCSRB is written, this is to ensure compability with future devices. [4]

3.3 ADMUX - ADC Multiplexer Selection Register.

Bit 0:4 provide the multiplexers with control signals, so that the right combination of analog inputs will be linked to the ADC. The cooling system gets

ADTS 2	ADTS 1	ADTS 0	Trigger Source
0	0	0	Free Running mode
0	0	1	Analog Comparator
0	1	0	External Interrupt Request 0
0	1	1	Timer/Counter0 Compare Match
1	0	0	Timer/Counter0 Overflow
1	0	1	Timer/Counter1 Compare Match B
1	1	0	Timer/Counter1 Overflow
1	1	1	Timer/Counter1 Capture Event

Table 3.1: ADC Auto Trigger Source Selections.

a single-ended (how this is implemented in hardware is explained in chapter 6) input on the first pin on port A, ADC0, on the microcontroller. [4]

Bit 5 represents the ADC Left Adjust Result (ADLAR) bit. It contains the control signal for how the converted result from the ADC is presented in the ADC Data Register. If the ADLAR bit is 1, the representation of the converted result is left adjusted; otherwise it would be right-aligned. If we use a 8-bit ADC we can represent the result left adjusted, but if we want to use a 10-bit ADC we need to have the result right adjusted. The reason for this is the way the microcontroller read its calculated value from the ADC Data Register. [4]

Bit 6:7 - REFS 0:1, represents the reference voltage that is used in the converter. These bits are called Reference Selection Bits, and since there are two of these bits we will be able to have four choices of this reference voltage. [4] In table 3.2 it is shown what choices we got:

REFS 1	REFS 0	Voltage Reference Selection
0	0	AREF, Internal Vref is turned off
0	1	AVCC with external capacitor at AREF pin
1	0	Internal 1.1V Ref. with external capacitor at AREF pin
1	1	Internal 2.56V Ref. with external capacitor at AREF pin

Table 3.2: REFS-table to the Analog-to-digital Converter.

3.4 Implementation of the Registers

Now we use these registers to program the microcontroller. In this thesis the registers are set to enable the ADC, and the converted value is set in free running mode, meaning that the ADC converts the incoming value at all time. When the first conversion is finished, the second is started. The ADC is programmed to start the converted value at a rising edge on the crystal.

3.4.1 ADCSRA

Bit 0:2 are the ADC Rescale Select Bits. We use a maximal division factor, so we get maximized the accuracy on the measurements. The maximal division factor is 128 and is retrieved by writing these bits to 111. We now use equation 3.1 on page 6 to retrieve the input clock frequency for the ADC. We see that the frequency will be 15,6 kHz (156250 Hz).

Bit 3 represents the ADC Interrupt Enable. We don't need this interrupt in our case, this bit is zero.

Bit 4 refers to ADC interrupt flag. We do not use the interrupt mode, therefore it's no use for this flag and this bit is set to zero.

Bit 5 is the ADC Auto Trigger Enable bit. We have an ADC which is auto triggered, this bit will therefore be one and the selected source will be discussed later.

Bit 6 tells the ADC to start a conversion. We use free running mode and as we have discussed this mode will alternate the bit between one and zero. This bit is written to zero in the beginning; it has no effect since the bit-value will be one as it gets a conversion started.

Bit 7 decides if the ADC is on or off. We want to enable the ADC, so we write this bit to one.

The ADCSRA's bit pattern is now 1010 0111 (bin) or 0xA7 (hex).

3.4.2 ADCSRB

Bit 0:2 refers to ADC auto trigger source. We will use the free running mode, 000. It will take more time than the single conversion mode, but there is no time limit in this measure feature. This free running mode uses 25 clock cycles per conversion, while the single conversion mode uses about 13 clock cycles per conversion.

The ADCSRB's bit pattern is now ANDed with 1111 1000 (bin) or 0xF8 (hex), this is done so we can set the last three bits without changing the first five. The first five are the reserved bits. The final bit pattern in ADCSRB is now xxxx x000 (bin), the x's marks the reserved bits.

3.4.3 ADMUX

Bit 0:4 provide the multiplexers with control signals, so that the right combination of analog inputs will be linked to the ADC. The system gets a single-ended input on the first pin on port A, ADC0, the control signal will therefore be 00000.

Bit 5 is the ADLAR bit. This code operates with a 10-bit ADC and we need our result to be right-adjusted, so this bit will be 0.

Bit 6:7 - REFS 0:1. We have chosen the REFS bits to be 00, which gives us reference voltage from the AREF pin. The voltage on this pin is 2,5V, we are now going to see why. When we choose the referance voltage, we need to know how the microcontroller works. The settings in the microcontroller will change if the input voltage value exceeds 2,5V on ADC0, since the ADC0 will go from low to high. We also want to use a 10-bit resolution ADC, in order to achieve max resolution from the ADC. Remember our goal is to achieve a resolution, 0,1 degrees for each digital value or better. With these two requirements we see that the reference voltage needs to be 2,5V.

The bit pattern in the ADMUX is now 0000 0000 (bin) or 0x00 (hex). The microcontroller is programmed to have a 10-bit resolution, and therefore, there will be 1024 levels.

$$\begin{aligned}\text{Voltage per level} &= \frac{\text{Reference voltage}}{\text{Number of levels}}\\ &= 0,0024V/\text{level}\end{aligned}$$

The voltage sent into the converter will get a value between 0 and 1024, and this value will be written in the ADC Data Register, see figure 3.2 on page 6. To obtain this value we use the read_adc feature (see Appendix C).

3.5 ADC Accuracy

The ADC accuracy is important since we know that there will occure errors inside the microcontroller while it calculates the output. Therefore we check what errors could occure in a microcontroller. In addition to the following error sources we got Integral Non-Linearity and Differential Non-linearity.

3.5.1 Offset Error

An offset error is shown in figure 3.3, the actual curve will have a deviation from the ideal curve. The offset will be the same at all points on the curve, so the actual curve will have the same slope as the ideal curve. [4]

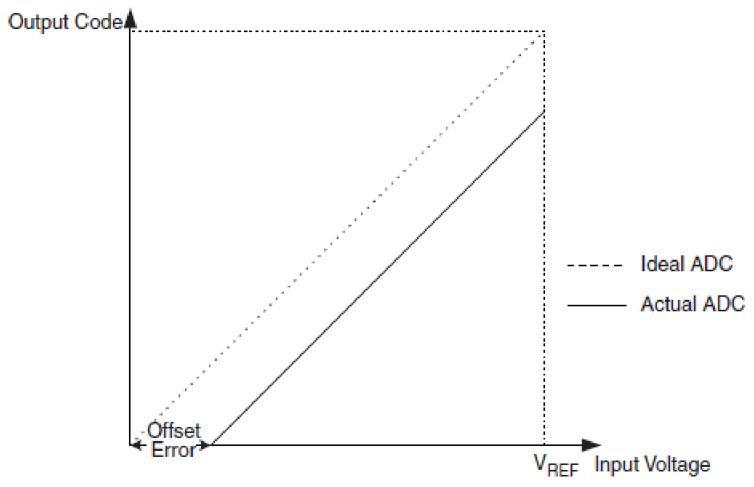


Figure 3.3: This is a representation of the offset error we might have in a microcontroller.

3.5.2 Gain Error

A gain error is shown in figure 3.4 on the next page. We can see that the only difference between the ideal and the actual curve is the slope. This means that the microcontroller have been calculating the output with a slightly different gain than it was supposed to. To find this error we need to check the deviation between the two curves of the last transition. [4]

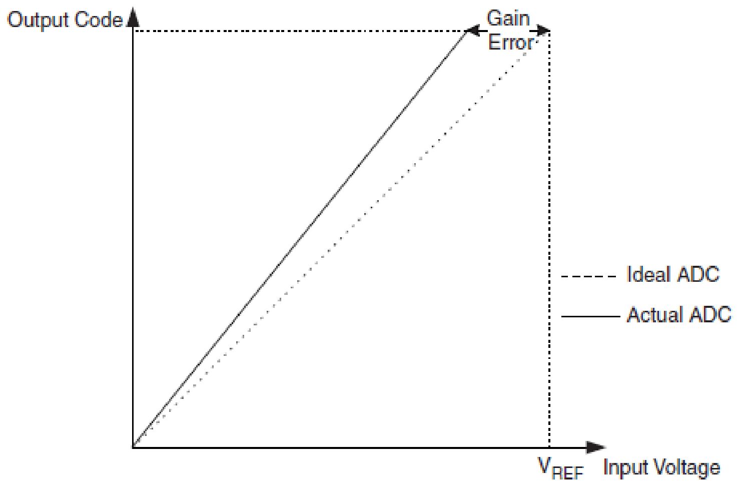


Figure 3.4: This is a representation of the gain error we might have in a microcontroller.

3.5.3 Quantization Error

Our microcontroller got a finite number of values it could adress to the input voltage, due to this quantization, an input voltage which is 1 LSB wide will be adressed the same value (LSB in this content stands for least significant bit). Maximum $\pm 0.5LSB$ from the true value. [4]

3.6 Timing inside the ADC

When we use the microcontroller as an external trigger source it is important that it is precise. In chapter 9 we will see how we use this microcontroller to make a square pulse at a frequency of 10 Hz. In chapter 8 we will see how this square pulse can be used as an external trigger.

From figure 3.5 on the facing page we can see how the timing works inside the microcontroller. We see from the figure that the end off each operation happens after the clock has performed a cycle.

This may be enough to make the microcontroller a bad trigger source. Since the instruction fetch and instruction execute is as long as the clock cycle it should'nt have any affect on the frequenzy of the square pulse.

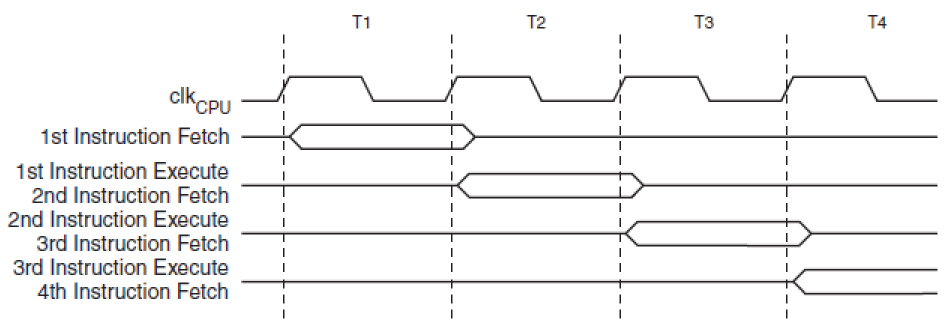


Figure 3.5: This shows us the internal timing in the microcontroller when operations are processing. We see that both instruction fetch and instruction execute is ended after the clock cycle is finished.

Chapter 4

Camera

The AVT Guppy F044 NIR is the camera that will be used. The camera have a 6-pin FireWire connection and an 8 pin HIROSE cable connection. The AVT Guppy F044 NIR is also equipped with a Sony CCD ICX429 device. It can have a shutter speed from 62μs and up to 67 108 864μs. The camera is highly sensitive to Infrared light. The camera support Trigger_Mode_0, Trigger_Mode_15 and trigger delay which will be discussed in chapter 7. The camera can also amplify the pictures by programming the camera to use gain. The camera can amplify pictures with 24dB (decibel) in 680 gain steps. [13] This means:

$$\begin{aligned}\text{Increment length} &= \frac{\text{Decibel}}{\text{Gain Steps}} \\ &= 0.035\frac{dB}{step}\end{aligned}$$

4.1 The CCD Detector

The camera that is used in this master thesis consists of a Sony CCD ICX429 device. This is a charge coupled device (CCD), which was invented in the 1970's. The CCD is used in professional astronomical imaging. It has replaced film and photographic plates because the CCD's ability to collect light is more efficient. Since the resulting data is digital, it allows images to easily be processed in a computer. [6]

The CCD is usually referred to as an electronic photon detector. If you have a CCD camera it consists of a two-dimensional array of photon detectors that lies in a layer of silicon, this silicon works as a semi-conducting material. In order to collect images this array of photon detectors is placed at the focal plane of a telescope. [6]

Each one of these detectors is referred to as a pixel. These pixels are capable of collecting photons and storing the produced number of electrons individually. Since each individual pixel has detected varying intensities of light, it's possible to produce a digital image by reading it into a computer. [6]

4.2 Primary Sources of Noise

4.2.1 Readout Noise

Readout noise is a type of additive electronic noise; it is added to the final picture as it is being read out of the device. In order to understand the readout noise, we can study a picture that has been illuminated with exactly 100 photons in every pixel. [10]

The final image will not have exactly 100 registrated electrons in each pixel because of the random fluctuations within the readout amplifier. The uniform illumination of each and every pixel isn't very practical. If we instead take a picture where we are supposed to get zero in each pixel, it will be easier to check how big these fluctuations will be. [10]

To capture such a photo we put the lens cap on the camera and use an exposure time equal to zero. A photo like this will contribute to construct a bias frame. A bias frame is generated by averaging nine or more images. We have to do this averaging to get the right image of the readout noise, since it will be different for each time there is captured a photo. The readout noise can be removed by subtracting this bias frame from the source image. [6] [10]

4.2.2 Thermal and Dark Noise

Thermal noise is the most basic type of noise, caused by thermal vibrations of bound charges. Consider a resistor at a temperature of T degrees Kelvin(K), as depicted in figure 4.1 on the next page. [8]

The electrons in this resistor are in random motion, with a kinetic energy that is proportional to the temperature, T. These random motions produce small, random voltage fluctuations at the resistor terminals, as illustrated in the figure. This voltage has a zero average value, but a nonzero rms value. [8]

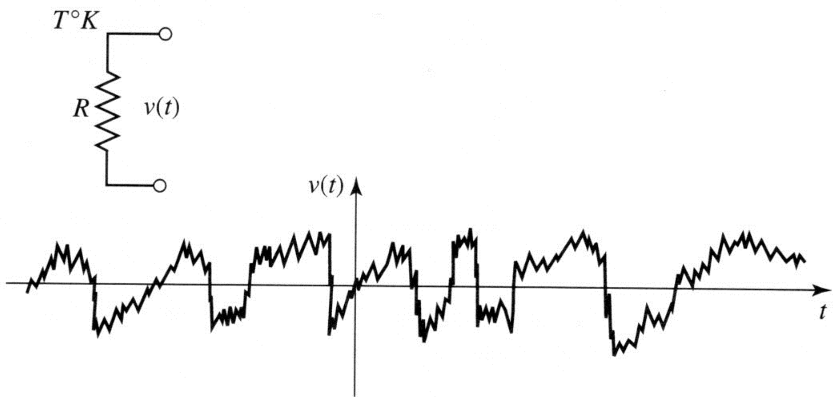


Figure 4.1: A random voltage generated by a noisy resistor.

This thermal noise is another source of noise in a CCD camera. In a CCD this thermal noise is represented in a different way. In this case there are two types of electrons, those we want and those we don't want. [6]

The type of electrons we want are the ones that are generated by the photons which has been hitting the pixels. The other type of electrons are generated by heat that is produced in the system. When these electrons hit the pixels, there is no way the system can distinguish these unwanted electrons from the wanted photons. [6]

The electrons that have been excited even when light does not hit the detector are referred to as dark current. The electrons from each pixel will be counted and more dark current will contribute to increase the mean value of a picture (more of this in chapter 7). As we understand this dark noise is a consequence of the thermal noise. If we use a long enough exposure time, the detector could be filled with electrons generated by the thermal noise. [6]

This noise can be eliminated by cooling the CCD detector. But even when the CCD is cooled, there will still be thermal noise. This noise can be removed in the same way as for the readout noise. [6]

If we make a dark frame, we can subtract this dark frame from the source image and compute the image as correct as possible. A way we can create this dark frame is to putt the lens cap on the camera and set the exposure time and other settings, as gain, to be the same as the settings in the source image. We take nine or more pictures and averaging these to make a frame which will represent the dark noise in a source image. [6]

4.2.3 Shot Noise

In electronic circuits there could occur noise of the form "shot noise". This noise occurs because of the electric currents random nature. We know that the electric current consists of individual charged electrons, when this current is moving it won't move in a continual motion. There will be a difference in distance between each individual electron. [2]

This means that the time between each electron arrives at a cross section of a wire, is completely independent from when other electrons arrives. This will lead to a slight variation in the flow. When we talk about DC (Direct Current) we don't get any information about this variation, we use the mean value when referring to this current. [2]

In a CCD camera we will also have shot noise as a source of noise. When you take a picture, there is no way to get the exact same number of photons recorded in each pixel with a constant exposure time. There will be a slight variation in intensity. This noise is removed at the same time as the dark current is removed. [2]

4.2.4 Other Sources of Image Degradation

The quality of the pictures can be disturbed in other ways than the ones already mentioned. For instance hot spots. Hot spots are pixels that have higher dark current than average. The reason for this may be that the silicon used in the CCD-chip took some damage under the manufacturing process. [6]

Sometimes the electrons get caught in traps when the readout is under process. When the vertical transfer of charge is blocked during the readout process, dark columns and long vertical dark streaks will occur on the pictures. [6] Bright vertical streaks could also appear on the pictures; these are also caused by traps. But in this case, the captured electrons have been leaked out into the closest pixels during the readout process. These defects will be revealed in a dark frame. [6]

Chapter 5

Thermometer

We are going to make a thermometer with a NTC-thermistor (Negative Temperature Coefficient-Thermistor). Since this thermometer is to be used as an indicator for if we have to cool the camera or not, we need quite good resolution.

We have decided that a resolution near 0.1 degrees Celsius for each digital value will be enough. In figure 5.1 on the following page, we can see a Wheatstone bridge that will be used to measure the temperature. This bridge will be modified as we design and calculate the values for R_1 to R_4.

The output voltage from a resistive deflection bridge as the Wheatstone bridge [3] is defined by formula:

$$V_{out} = V_s(\frac{R_1}{R_1+R_4} - \frac{R_2}{R_2+R_3}) \tag{5.1}$$

The Wheatstone bridge that we use includes just one variable resistor; a 10kΩ NTC-thermistor. We can exchange one of the resistors, see figure 5.2 on page 21. The NTC will vary as the temperature increase or decrease. This can be used to get different output voltage for various values of the temperature. Since we have exchanged R_1 to R_T we rewrite the equation 5.1.

$$V_{out} = V_s(\frac{R_T}{R_T+R_4} - \frac{R_2}{R_2+R_3})$$

If we divide by R_T in the first fraction and divide by R_2 in the second fraction, we get:

$$V_{out} = V_s(\frac{1}{1+\frac{R_4}{R_T}} - \frac{1}{1+\frac{R_3}{R_2}})$$

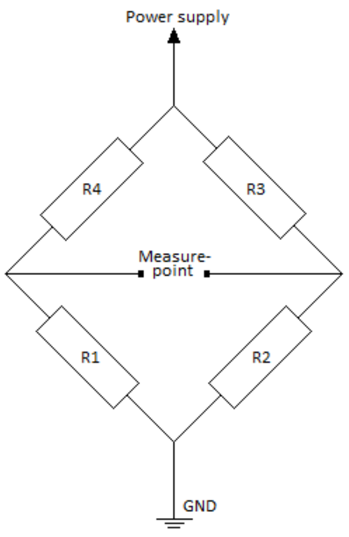


Figure 5.1: This is a sketch of the Wheatstone bridge befor we start the designing prosess.

We use R_2, R_3 and R_4 as fixed resistors, and need to address these resistors to some values. The supply voltage will be decided later as we design the bridge.

From R_T we can find the minimum and maximum output voltage.[3]

$$\begin{aligned} V_{min} &= V_s(\frac{1}{1+\frac{R_4}{R_{T_{min}}}} - \frac{1}{1+\frac{R_3}{R_2}}) \\ V_{max} &= V_s(\frac{1}{1+\frac{R_4}{R_{T_{max}}}} - \frac{1}{1+\frac{R_3}{R_2}}) \end{aligned}$$

From the thermistor's datasheet we know how the resistant in the NTC varies with temperature, see figure 5.3 on page 22. [11] We know that the temperature will lie in the [0 30] degrees Celsius range. We can see the resistors value for these temperatures; it will be about 32000Ω at 0 degrees Celsius and around 8000Ω at 30 degrees Celsius.

To design the bridge we require a balanced bridge, which means that $V_{min} = 0$.[3] To fulfill this we can see that:

$$\frac{R_4}{R_{T_{min}}} = \frac{R_3}{R_2}$$

This means that $R_3 = R_4$ and R_2 got to have the same value as $R_{T_{min}}$, $R_{T_{min}}$ is about 8000Ω. We use a potentiometer to match this resistant. To find R_3

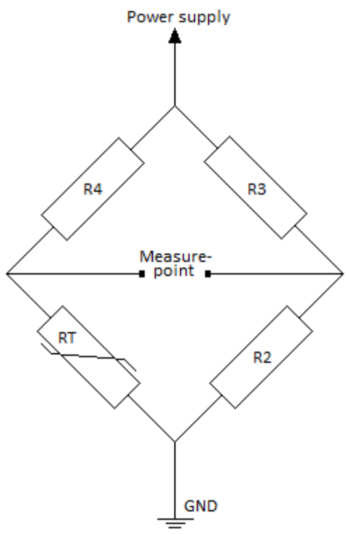


Figure 5.2: The Wheatstone bridge with a NTC-termistor.

and R_4 we need to study some more.

We want to get as much difference between V_{min} and V_{max} as possible, since we want a good resolution. To achieve this we check where V_{max} - V_{min} got the largest value as we vary R_3 and R_4.

$$\begin{aligned} V_{max} - V_{min} &= V_s(\frac{1}{1+\frac{R_4}{R_{T_{max}}}} - \frac{1}{1+\frac{R_3}{R_2}}) - V_s(\frac{1}{1+\frac{R_4}{R_{T_{min}}}} - \frac{1}{1+\frac{R_3}{R_2}}) \\ y = \frac{V_{diff}}{V_s} &= \frac{1}{1+\frac{R_4}{R_{T_{max}}}} - \frac{1}{1+\frac{R_4}{R_{T_{min}}}} \end{aligned} \tag{5.2}$$

From figure 5.4 on page 23 we can see how this voltage difference vary with different resistors at R_3 and R_4. The voltage difference is viewed in percent of the supply voltage. We see that we have maximum around 16000Ω. This will give maximum difference in the output voltage for a temperature range of 30 degrees Celsius.

We check the result by deriving the expression 5.2 to retrieve this maximal difference between V_{min} and V_{max}, as we vary R_4. To find the maximum we

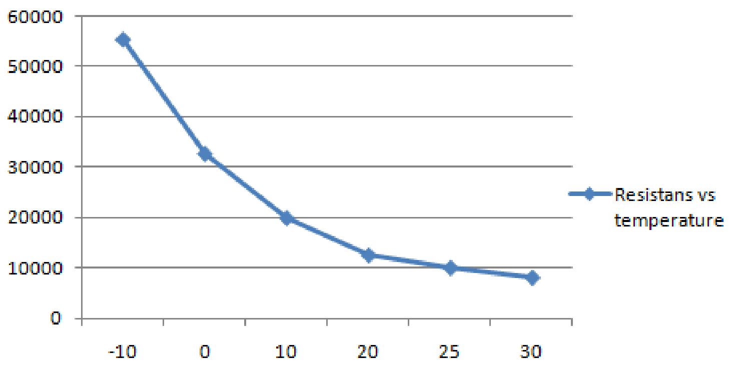


Figure 5.3: Resistance in the NTC versus temperature in the NTC. Resistance [kΩ] at the y-axis and temperature [°C] at the x-axis

need to check where the derived expression is equal to zero.

$$\begin{aligned} y &= \frac{1}{1+\frac{R_4}{R_{T_{max}}}} - \frac{1}{1+\frac{R_4}{R_{T_{min}}}} \qquad (5.3) \\ \frac{\delta y}{\delta R_4} = 0 &= -\frac{1}{R_{T_{max}}(1+\frac{R_4}{R_{T_{max}}})^2} + \frac{1}{R_{T_{min}}(1+\frac{R_4}{R_{T_{min}}})^2} \\ R_4^2[\frac{1}{R_{T_{min}}} - \frac{1}{R_{T_{max}}}] - (R_{T_{max}} - R_{T_{min}}) &= 0 \end{aligned}$$

Solve this second order equation and we retrieve R_4:

$$R_4 = 16000$$

We have found $R_3 = R_4 = 16000\Omega$. We don't have this resistant, so we used the closest one we got, approximate $17,2k\Omega$. We have now designed the bridge to have maximum voltage difference at the output measure points.

To find the supply voltage we need to know how much the microcontroller can take. The microcontroller can take an input difference of 2,5V on the ADC input. The reason for this is as we explained in chapter 3, that when the input voltage on the ADC exceeds 2,5V the pin value goes from low to high and destroy the settings in the microcontroller.

We have learned to stay away from boundaries like these when designing circuits. Therefore we say that the maximum voltage the microcontroller

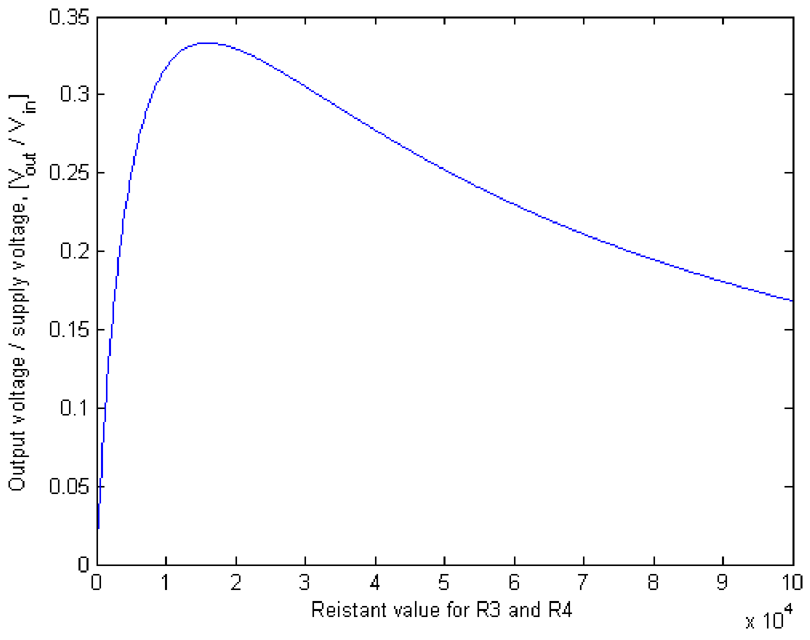


Figure 5.4: Here we have the voltage difference in relation to the supply voltage as we vary R_3 and R_4.

can handle at the ADC input is $V_{diff} = 2,4V$. We can now find what supply voltage we need.

$$\begin{aligned} \frac{V_{diff}}{V_s} &= \text{Voltage difference in percent of supply voltage.} \\ V_s &= 7,2V \end{aligned}$$

In the calculation we use 0,333, this is the voltage difference in percent of the supply voltage at $R_4 = 17,2k\Omega$, see figure 5.5 on the following page.
We see that the supply voltage over the bridge needs to be 7,2V to get the maximal voltage difference to be about 2,4V at the differential output from the bridge. Since we use a NTC(Negative Temperature Coefficient) thermistor, this voltage difference is achieved when the temperature is 0 degrees Celsius.

Now we have taken advantage of all the 1024 digital values, so we should be able to calculate the resolution.

$$\begin{aligned} \text{Resolution} &= \frac{\text{Degrees Celsius}}{\text{Number of levels}} \qquad (5.4) \\ &= 0,029[\frac{°C}{\text{level}}] \end{aligned}$$

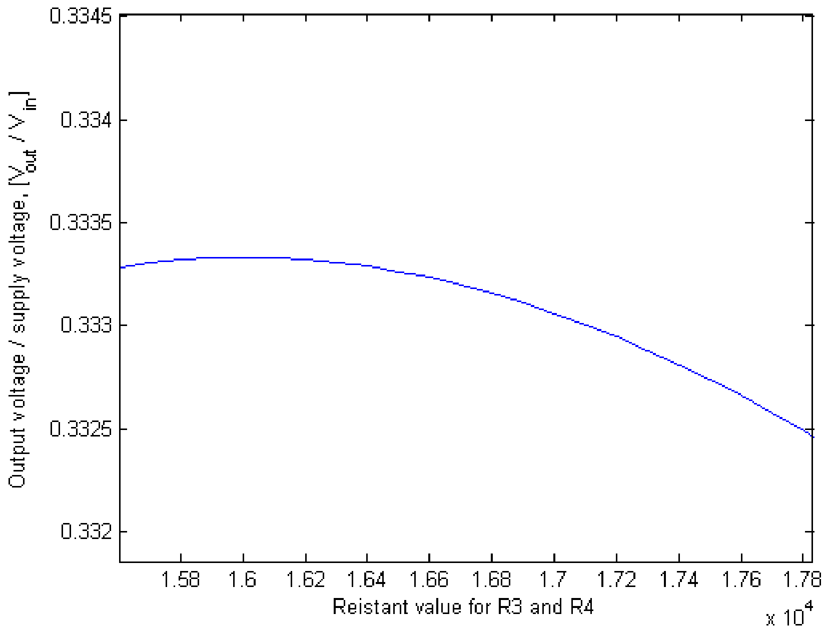


Figure 5.5: Here we have the zoomed version of figure 5.4. We have plotted the voltage difference in relation to the supply voltage as we vary R_3 and R_4.

We can see that we now got a better resolution than the one we actually wanted. This would have been the real resolution if we didn't consider the boundaries when calibrating the system. The final resolution will therefore be viewed after we have calibrated the overall cooling system.

When we calibrated the bridge, we found that the supply voltage actually needed to be 7,15V (the calibration will be discussed in chapter 6). We could also see fluctuations in the output voltage, not big ones, but enough to throw the microcontroller out of control. The digital value varied with about 60 digital values, in temperature this would correspond to 2 degrees. We therefore used an oscilloscope to check the noise on the bridge.

The bridge got noise at high frequencies; the peak to peak value were hard to read of the oscilloscope, but lied in the 300mV range. We see from this number that we should have a larger variation of the digital value.

$$\text{Variation in digital values} = \frac{\text{Peak to peak value}}{\text{Voltage per level}}$$

The voltage per level is the same we found in chapter 3. From the 300mV peak to peak value we should have had a variation of the digital value of about 125 digital values. The reason for this smaller variation is because we have written the software to take a mean value of 30 measured values (see Appendix C).

We could also see on the ocsilloscope that the bridge had the legendary 50 Hz noise. In the attempt to remove this noise we started to place an electrolyte on the bridge. This electrolyte is connected as shown in figure 5.6.

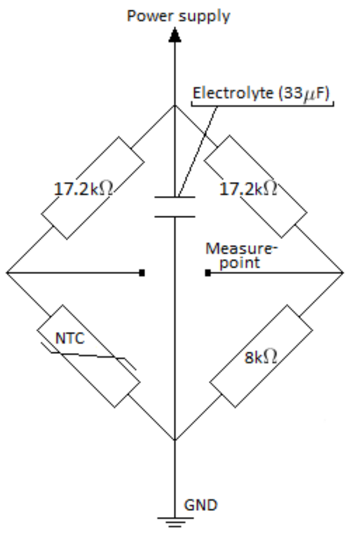


Figure 5.6: Here we can see how the electrolyte is connected. We have also set the value for R_2, R_3 and R_4

An electrolyte is supposed to remove the 50 Hz noise, but because of its design it is also eliminating some high frequent noise. We use a dry electrolyte, because it is more efficient to remove the high frequent noise than a wet electrolyte.

When we now connect the bridge to the oscilloscope, we see that the electrolyte has almost completely removed the 50Hz. The 50 Hz comes from antenna effects, which we can get from components and the bridge itself. To minimize the antenna effect, we have to make the bridge as small as possible, with as small components as possible.

After making a smaller version of the bridge we check the noise again. There was still some 50 Hz noise we had to get rid of. The high frequent noise is still on the new bridge, but the peak to peak value has now reduced to about 200mV. The solution to get rid of the last bit of 50Hz noise is to make an AC (Alternating Current) ground at the two measure points. See figure 5.7 for how the bridge is designed.

There is now only the high frequent noise left on the bridge. The high frequencies are removed by a capacitor that is placed over the output points. It works as a low pass filter. This low pass filter removed a lot of the high frequent noise. The peak to peak value of this noise is now 20 mV that will have some effect on the output voltage. These effects is eliminated in the software as we mentioned earlier (see Appendix C).

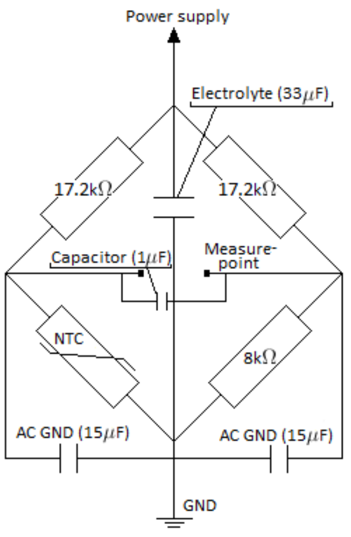


Figure 5.7: Here we see the final design of the Wheatstone bridge that will be used as a thermometer in this thesis.

Chapter 6

Cooling System

6.1 Cooling Element

The cooling element which will be used in this thesis is a peltier element (PE-127-10-13). When using a peltier element, it is critical to get all the energy away from the camera. The camera is using maximum 2 watt at a 12 volt power supply. We therefore need a peltier element that can handle a minimum of 2 watt. [1]

The peltier element we have chosen can handle up to 37,9 watt, we can therefore be sure not to destroy it. We chose the peltier to have the same dimentions as one of the camera sides so we get a big area covered. We then designed an aluminium sheet to embrance the camera on three sides, see figure 6.1 on the next page. This was to cool the camera from three sides in stead of just one. [1]

6.2 Switch

The switch is implemented as an electric component and works automaticly when it get control signals from the microcontroller. The switch is constructed with two MOSFET transistors, TIP3055 and BD137. We use the BD137 to turn on the TIP3055, as shown in figure 6.2 on page 29, because the microcontroller alone is not strong enough to turn on the TIP3055.

The reason why we have'nt implemented the BD137 alone, is because the current that will run through the peltier element is too great for this transistor to bear. The solution is to use our microcontroller to switch on the

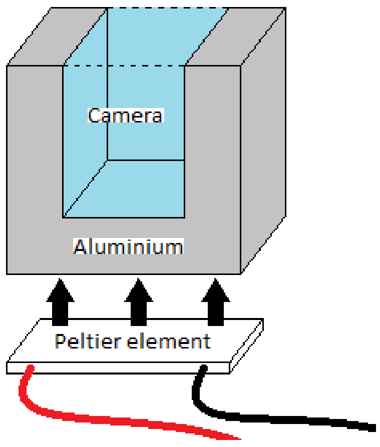


Figure 6.1: This is a sketch of how the aluminium is designed to embrance the camera, so the peltier element can cool the camera from three sides.

BD137 that will use the 12 volt power supply to switch on the TIP3055 transistor.

6.3 How the Cooling System Works

The microcontroller will get a voltage value in on one of the pins, ADC0. This voltage is coming from the Wheatstone bridge. The Wheatstone bridge has two outputs, so how does it work as a single ended input? If we use a floating power supply on the Wheatstone bridge we can use one of the outputs as a GND reference and send the other output to the ADC0 pin.

One of the outputs will be at a constant value; we send this to GND on the microcontroller and the power supply will now act as a floating one. The output that will vary is sent as input to the ADC0. When this voltage is getting to a certain value, we got to switch one pin at port D on the microcontroller to high. When the pin is high, the switch gets connected to ground, and this turns on the peltier element. When the peltier element has lowered the temperature on the camera, the microcontroller will turn off the peltier element.

If the microcontroller is to understand the voltage that is submitted from the measure bridge, it must be converted into a digital signal. This is where the built-in-ADC on the microcontroller is used. The analog voltage is con-

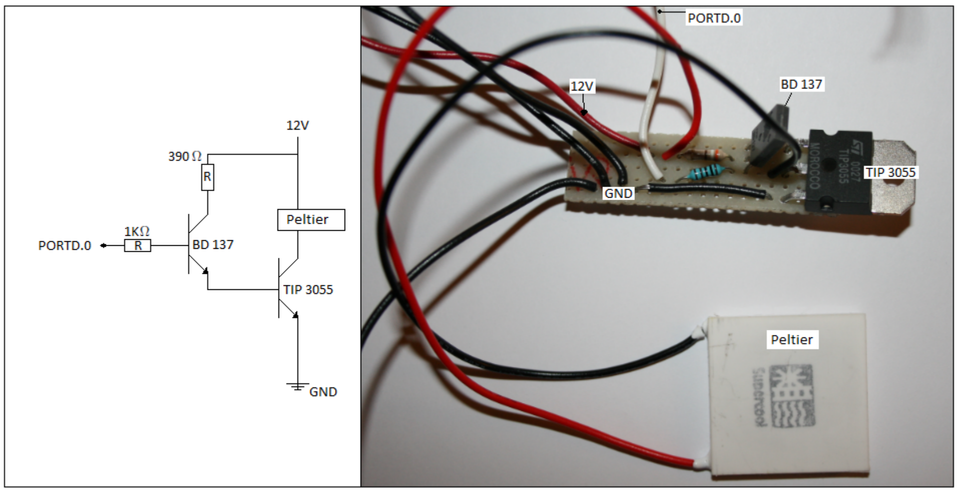


Figure 6.2: To the left is a sketch of the designed switch, and to the right is the switch in hardware.

verted to a digital value through the ADC. The relation between analog voltage and digital values is:

$$\text{Digital value} = \frac{\text{Analog voltage}}{\text{voltage/level}} \tag{6.1}$$

In table 6.1 on the next page we can see what voltage each digital value will be equivalent to. The "Digital value" column represent the expected digital value, while the "Displayed value" column describes what digital value the microcontroller has calculated. The "Offset" column will be discussed in section 6.4.

6.4 Offset

If we look at the "Offset" column in table 6.1 on the following page. The offset value is positive close to the digital value zero and slowly, but surely, it crawls to the negative side as the displayed digital value reach 1006. In chapter 3 we discussed ADC Accuracy, and here we clearly see that the microcontroller has accuracy issues.

Measured voltage	Digital value	Displayed value	Offset
0.004	1.6384	0	1.64
0.1	40.96	37	3.96
0.2	81.92	78	3.92
0.3	122.88	119	3.88
0.4	163.84	160	3.84
0.5	204.8	202	2.8
0.6	245.76	243	2.76
0.7	286.72	284	2.72
0.8	327.68	325	2.68
0.9	368.64	366	2.64
1.0	409.6	407	2.6
1.1	450.56	448	2.56
1.2	491.52	489	2.52
1.3	532.48	531	1.48
1.4	573.44	572	1.44
1.5	614.4	613	1.4
1.6	655.36	655	0.36
1.7	696.32	696	0.32
1.8	737.28	737	0.28
1.9	778.24	778	0.24
2.0	819.2	819	0.2
2.1	860.16	860	0.16
2.2	901.12	901	0.12
2.3	942.08	942	0.08
2.4	983.04	983	0.04
2.456	1005.9776	1006	-0.0224

Table 6.1: Digital values with known input voltage to the ADC.

We see that the offset from the first measurement have a deviation from the other offsets. The reason for this might just be that we are close to zero. If we look at the other offsets, we see that they decrease as we increase the voltage at the input on the ADC. We manage to use the three error sources, discussed in section 3.5, to describe the error in the microcontroller.

First the offset error occures, we have a deviation of about 4 from the real value at 0.1V. Since the deviation decrease as we increase the voltage at the input on the ADC, we have found gain error. If we look closer on the offset we can see that it permanently decrease by 0.04 digital values each time we increase the voltage 0.1V, but we got three exceptions.

We see that when we increase the voltage from 0.4 to 0.5, from 1.2 to 1.3 or from 1.5 to 1.6, the deviation decrease 1,04 digital values. These three steps can be associated with the quantization error.

6.5 Calibration of the System.

Before we can study how the cooling system is improving the pictures, we have to calibrate the system. To calibrate the system we have to check what voltages we got at the different temperatures. In figure 6.3 we see the calibration setup.

This setup will give us a voltage at the measure points on the bridge. These voltages will have a corresponding temperature which we get from the thermometers. The water will have temperatures at: 0, 5, 10, 15, 20, 25 and 30 degrees Celsius.

We get these temperatures by adding cold and warm water. The temperature of the water has to be stabilized, before we check the voltage at the measure points. To be sure of the temperature, we use three thermometers and calculate the average of the temperature.

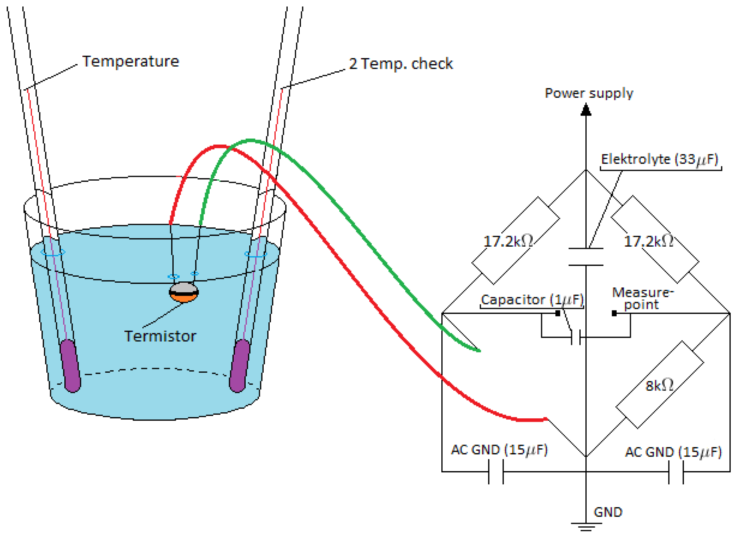


Figure 6.3: This is the setup we have used to calibrate the cooling system.

Temperature (average)	Voltage	Digital value	Displayed digital value
0.2 °C	2.4 V	983.04	983
5.0 °C	1.982 V	811.83	812
10.2 °C	1.533 V	627.92	627
15.0 °C	1.137 V	465.72	465
20.1 °C	0.735 V	301.06	300
25.0 °C	0.364 V	149.09	148
30.0 °C	0.013 V	5.32	4

Table 6.2: Digital value after calibrating the system.

First we need to set the voltage supply over the bridge. We have balanced the bridge and considered the boundary close to 0V. There are no problems there, but when we get close to 0 degrees Celsius we need the voltage to be 2.4V. We cool down the water to 0 degrees Celsius and change the supply voltage on the bridge until we have 2.4V at 0 degrees Celsius. This is achieved at $V_s = 7.15V$.

In table 6.2 we can see which digital value each voltage are converted into after we have set the supply voltage on the bridge. The values in the "Digital value" column are derived from equation 6.1 on page 29. We will use the "Displayed digital values" in our program code to get control over the temperature in the camera.

These values will be set as a limit value inside the microcontroller one at a time. The microcontroller will start the cooling process each time the calculated digital value gets smaller than the limit value. When the microcontroller has cooled the camera for a while, and the calculated digital value exceed the limit value it will stop the cooling process.

Since the system now is calibrated, and we got the final values from the calibration, we are able to recalculate the final resolution for our thermometer. We use the difference in degrees Celsius and the respective difference in digital values. We use equation 5.4 on page 23:

$$\begin{aligned} \text{Resolution} &= \frac{30.0 - 0.2}{983 - 4} \\ &= 0.03 \frac{\text{Degrees}}{\text{Digital value}} \end{aligned}$$

We clearly see that the final resolution is far better than the one we wanted.

Chapter 7

Quality of Pictures

We will now check the pictures as we use what we have learned and made throughout this master's thesis. Remember, all pictures are taken with the lens cap on the camera, since we want to eliminate all photons to view the noise problems with a camera.

When we were to take the pictures which were supposed to view the dark current we discovered a noise source. We discovered the source when we used 650 in gain, so the source must be inside the camera. When we use 650 gain it means that the pictures have been amplified with 22.75dB.

In figure 7.1 we can see one of the pictures taken. The picture is taken at 25 degrees Celsius with an exposure time of 65 seconds. We can clearly see that there is a lot of noise in the top left corner.

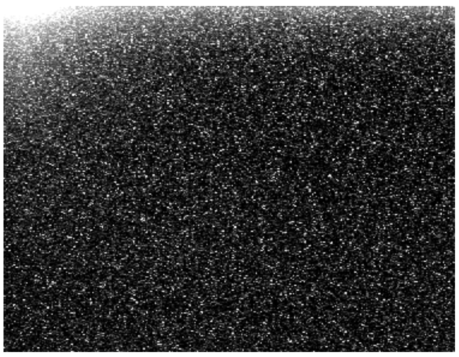


Figure 7.1: This picture is taken with an exposure time of 65 seconds and with 650 gain. It shows us the unknown noise source.

We had to find the source of this noise. At the back of the camera there is placed two LED-lights, there may be a leakage of light from the back to the front inside the camera. These LED's where covered to eliminate as much noise as possible. Then we took a new picture with the same settings. In figure 7.2 we can see the result.

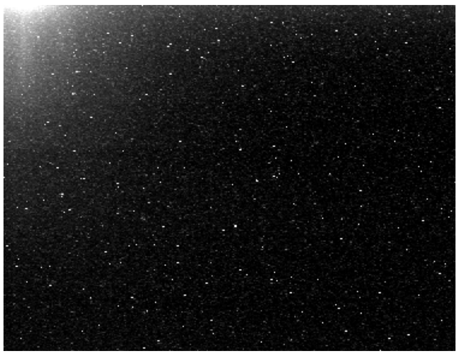


Figure 7.2: This picture had the same settings as figure 7.1, but this picture is taken after we have covered the LED-lights we find at the back of the camera.

We see that we have improved the picture a bit, but there is still a lot of noise. The light is eliminated as a noise source, so now we've got the temperature left. We therefore cool down the camera so that we can see the effect.

The camera is tested in two ways. One; we had the camera running for some time before we cooled it down and started the exposure. Two; we first cooled the camera and then turned it on and started the exposure as fast as possible. We can see the results in figure 7.3 and 7.4 on the facing page.

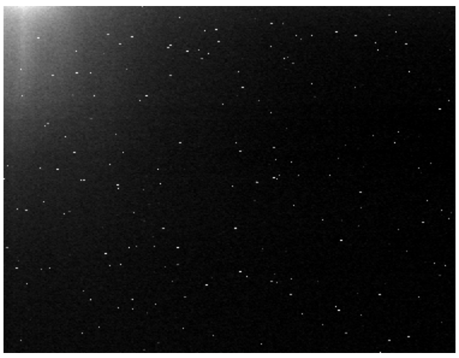


Figure 7.3: The camera had the same settings as before. The camera has been running for a while and then cooled down to 0 degrees Celsius. When the camera was cold we took this picture.

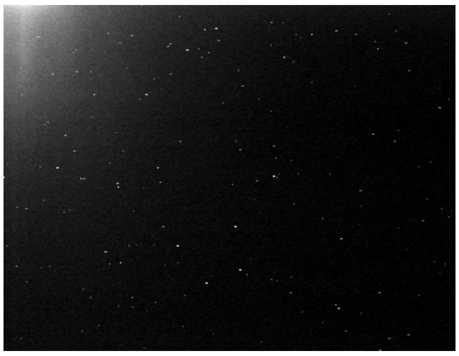


Figure 7.4: The camera had the same settings as before. The camera has been turned off while we have been cooling the camera. Immediately after turning the camera on, we start the exposure and we retrieve this picture.

As we see from the pictures it's not easy to get rid of this noise. To identify this noise source we contacted Allied Vision Technical support. They said that there is placed a CCD Vertical Clock Driver on the CCD PCB card (Card which holds the CCD). There might also be other noise sources on this card. Keep in mind that the camera is not built to take pictures with max exposure time and max gain.

There is no way to remove this noise. To calculate the dark current we therefore decided to use only the pixels at the right bottom corner of the pictures, as far away as we can get from the noise source. This thesis is discussing dark current in a general CCD camera, not in a spesific Guppy camera. Therefore we must study the dark current as if the Charged Coupled Device (CCD) in the camera is placed a distance from all the opperational components that are inside the camera.

We now check what difference it makes on the pictures to use the coolingsystem on the camera. Since the camera is highly sensitive near infrared ligth, we know that by cooling the camera it will give some effect on the pictures.

We use *"'imread"'* in MATLAB to calculate the mean value of each picture. The reason why we check the mean value is described in chapter 4 in section 4.2.2 Thermal and dark noise. The pixels in the pictures got values between 0 and 255 and by using *"'imread"'* we get these values put in an array that is as big as the number of pixels in the picture. The value 0 will correspond to completely black, while the value 255 will correspond to bright white (see Appendix E).

If we have a picture with a lot of noise, the mean value of that picture will be greater than a picture with less noise. We calculate this mean for 7 temperatures between 0 and 30 degrees Celsius, we also check what difference it makes to use 0 in gain rather than 650 in gain.

The first picture series are taken at the temperatures 0, 5, 10, 15, 20, 25 and 30 degrees with gain value set to 650. The second series are taken at the same temperatures, but the gain is now set to 0. Both series had an exposure time of 65 seconds.

The shot noise is removed in the same way as the dark noise, so if we removed the shot noise we would not be able to see the effect by cooling the camera. We will therefore not remove the shot noise from the pictures before we calculate the mean values. The readout noise however have been removed from the pictures before we have calculated the mean values of the pictures.

In figure 7.5 on the next page we see the mean value of each picture at the different temperatures.

We check the mean values from the second series and we see the same pattern, but the values are smaller. less gain less mean value. See figure 7.6 on page 38.

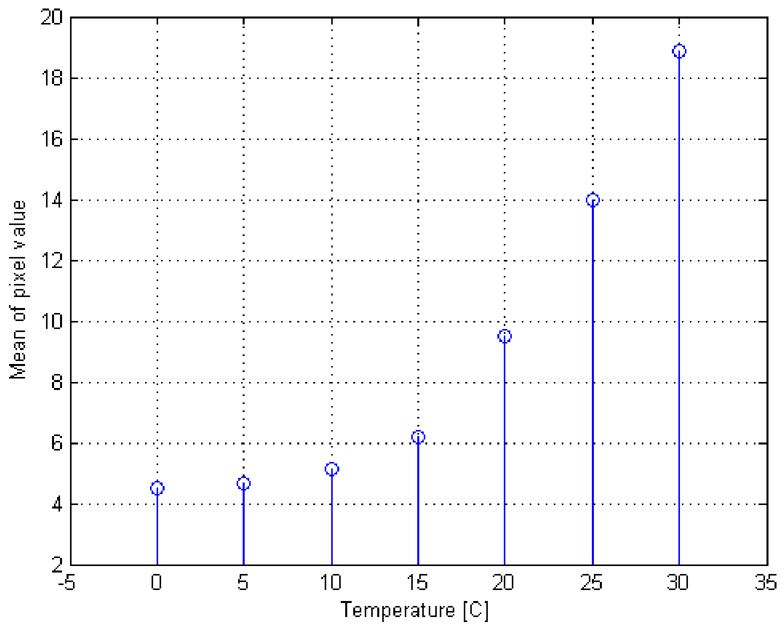


Figure 7.5: This is a plot of the mean pixel values of the dark frames taken at each temperature. The camera had 65 seconds exposure time and 650 gain when these pictures were taken.

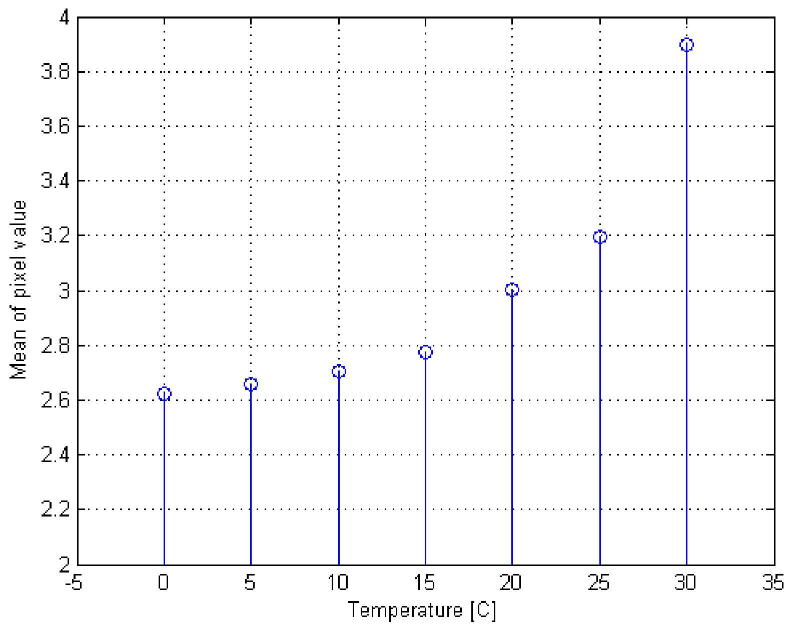


Figure 7.6: This is a plot of the mean pixel values of the dark frames taken at each temperature. The camera had 65 seconds exposure time and 0 gain when these pictures were taken.

Chapter 8

External Trigger

If you can decide manually when the camera should or should not take pictures, you would say that you got yourself an ordinary camera. If you want to do this in an automated way it isn't as easy as you may think. The AVT Guppy F044 NIR needs either a software trigger or an external trigger.

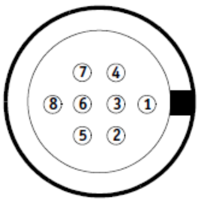
When we want to use the external trigger we need to program the camera to wait for this external trigger. The computer will program all the registers in the camera, and wait for the camera to take a picture.

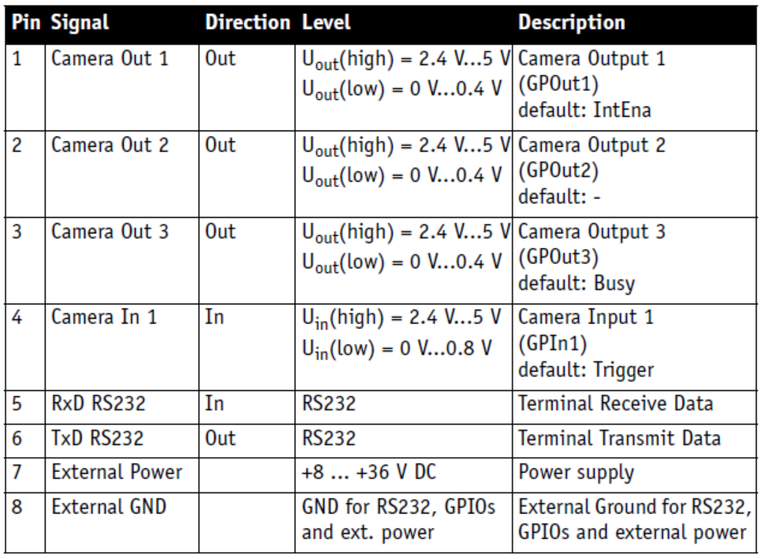
The camera won't start the exposure before it have retrieved an external trigger signal. When the camera gets the trigger signal the picture is taken. The computer is on standby and waits for the picture to arrive. The computer saves the incoming picture to its predefined place. In case of a software trigger we need to program the camera to use the software trigger.

In figure 8.1 on the next page we can see the pin configuration that is in the back of the camera. From the figure we can see that the trigger input is going in on pin 4 and the GND should be connected to pin 8. [14]

The Guppy cameras support three different external trigger modes. To control the camera with external trigger you need to program the camera so that it can understand one of these modes. The trigger signals needs to be presented in a way the camera can understand them.

Trigger_Mode_0 is also known as the edge mode. It sets the shutter time to the predefined shutter value which is set in one of the cameras registers. The shutter value is set when programmed. [13]





Pin	Signal	Direction	Level	Description
1	Camera Out 1	Out	U_{out}(high) = 2.4 V...5 V U_{out}(low) = 0 V...0.4 V	Camera Output 1 (GPOut1) default: IntEna
2	Camera Out 2	Out	U_{out}(high) = 2.4 V...5 V U_{out}(low) = 0 V...0.4 V	Camera Output 2 (GPOut2) default: -
3	Camera Out 3	Out	U_{out}(high) = 2.4 V...5 V U_{out}(low) = 0 V...0.4 V	Camera Output 3 (GPOut3) default: Busy
4	Camera In 1	In	U_{in}(high) = 2.4 V...5 V U_{in}(low) = 0 V...0.8 V	Camera Input 1 (GPIn1) default: Trigger
5	RxD RS232	In	RS232	Terminal Receive Data
6	TxD RS232	Out	RS232	Terminal Transmit Data
7	External Power		+8 ... +36 V DC	Power supply
8	External GND		GND for RS232, GPIOs and ext. power	External Ground for RS232, GPIOs and external power

Figure 8.1: The pin configuration at the back of the camera.

When the external trigger signal is applied to the input pin on the camera, the camera will take a picture with its programmed settings. It's triggered on falling edge. In figure 8.2 we can see how the trigger works.

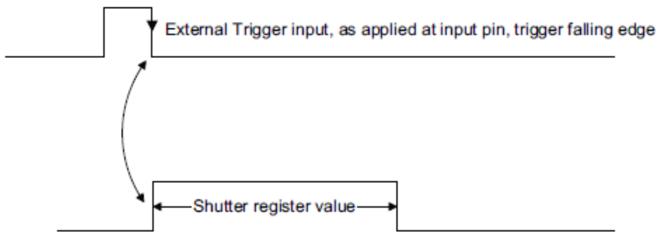

External Trigger input, as applied at input pin, trigger falling edge
Shutter register value

Figure 8.2: This shows us how the Trigger Mode 0 is working. When the camera register a falling edge it starts the exposure time which is set in one of the registers in the camera.

Trigger_Mode_1 is also known as the level mode. It sets the shutter time to the active low time of the pulse applied (or active high time in the case of an inverting input). If we use non-inverting input, the shutter will be on as long as the trigger input is high. In figure 8.3 on the next page we can see how the trigger works. [13]

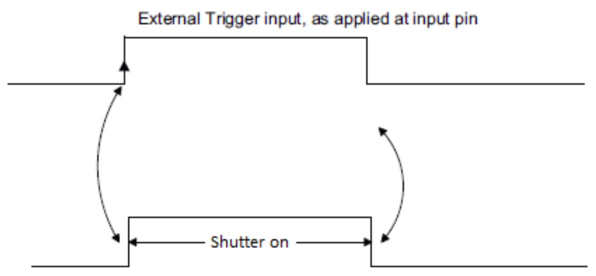


Figure 8.3: This shows us how the Trigger Mode 1 is working. As long as the trigger signal is high the camera will have its shutter turned on.

Trigger_Mode_15 is a programmable mode. It is a bulk trigger, which is triggered by an external trigger signal. When this mode is activated with the external trigger, it would be able to take continuous, one-shot or multi-shots with the internal trigger. This mode is activated on falling edge, just as the Trigger_Mode_0. See figure 8.4 for how it works. [13]

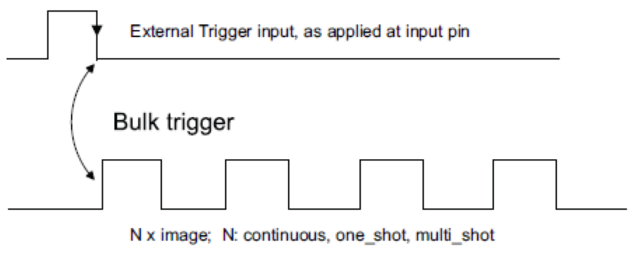

External Trigger input, as applied at input pin
Bulk trigger
N x image; N: continuous, one_shot, multi_shot

Figure 8.4: This shows us how the Trigger Mode 15 works. When the camera registers a falling edge it starts to take as many pictures as it is programmed to capture.

Guppy F044 NIR support only Trigger_Mode_0 and Trigger_Mode_15. It also support trigger delay, which you can compare with a self-timer. When the camera gets the external trigger it will be able to delay the actual internal trigger with the predefined time delay. This ability to delay the trigger would be a waste if we use the software trigger, since we are able to program a delay into the software. [13]

Chapter 9

Trigger System

When cameras like the AVT Guppy F044 NIR are used in scientific experiments, we rely on the precision and the stability in the triggers. This is to ensure that we get the data we need. In this chapter we will study the precision and stability in three different trigger sources, two external triggers and one software trigger.

9.1 Hardware Setup

The hardware that is used as trigger sources will be a microcontroller (ATmega 324PA), HP 33120A Function Generator and a stationary computer. The computer uses a MSI K9VGM-V mother board and a AMD Sempron^{TM} 3400+ processor. The firewire is connected to one of the mother boards PCI slots. The operative system we use on the computer is Ubuntu 9.04. The camera is set to have an exposure time at 62μs and use Trigger_Mode_0 in all three cases.

- The ATmega 324PA is the same microcontroller that we have been using throughout this thesis. This microcontroller is now used as an external trigger. It is programmed to create a square-pulse with a 10Hz frequency (see Appendix C). This means that the time interval between each falling edge is 100 ms. Since the camera has an exposure time of 62μs, it will be taken a picture for each pulse.
- The HP 33120A is a frequency generator, this is used as the second external trigger source. The operative frequency range for a square pulse is $100\mu Hz$ - 15 MHz. The frequency generator is using the same settings as the microcontroller, a square-pulse at 10Hz.

- We use the computer as the software trigger source. The computer is programmed to have a 100ms delay for each time a picture is taken. For this delay we use a while-loop inside the program. This will create a frequency on outcomming pictures at 10Hz.

9.2 Method

As we mentioned earlier, we have a computer rigged to this trigger system. The computer we use is the same as the software trigger source. The computer will program the camera to its specified settings. After the camera is completely programmed, it waits for the trigger signal that will be sent from one trigger source at a time.

We will take 5 series of 20 pictures for each of the trigger sources. We use the setup as described in chapter 2 figure 2.2 on page 4. The trigger source in this figure is exchanged for each test. The software trigger use the firewire to send its trigger signal. When the picture is taken the computer will store the picture at its predefined place inside the computer.

On these pictures there will be a timestamp. This timestamp is used to check the precision and stability of each trigger source. We have set the frequency to be 10Hz for all three sources. This means that we would expect the time difference between two timestamps to be 100ms.

9.3 Trigger Stability

We check the difference between each of these timestamps. We get 19 differences for each series. We check the mean value and variance for this difference for each series. The following formula is used to calculate mean, E(x), and variance, σ_x^2:

$$\begin{aligned} E(x) &= \frac{\Sigma_{i=0}^{N}(x_i)}{N} \\ \sigma_x^2 &= E(x^2) - E(x)^2 \end{aligned}$$

In table 9.1 on the facing page we see the results. We see that there is some variation in the variance. We use weighted averaging (see Appendix A) to get one mean value and one variance so that we manage to plot three Gauss curves for these three trigger sources.

Software	Mean $[ms]$	Variance $[ms^2]$
Serie 1	99.99	0.0036689
Serie 2	99.99	0.0037924
Serie 3	99.99	0.0037596
Serie 4	99.99	0.0037779
Serie 5	99.99	0.0038055
Frequency generator	Mean $[ms]$	Variance $[ms^2]$
Serie 1	100.00	0.0021527
Serie 2	100.00	0.0019839
Serie 3	100.00	0.0020702
Serie 4	100.00	0.0021526
Serie 5	100.00	0.0019980
Microcontroller	Mean $[ms]$	Variance $[ms^2]$
Serie 1	100.03	0.0036111
Serie 2	100.03	0.0034737
Serie 3	100.03	0.0035801
Serie 4	100.03	0.0035382
Serie 5	100.03	0.0036845

Table 9.1: This table show us the mean value and variance for five series with each trigger source.

In table 9.2 we can see the calculated values which we have used to plot the Gauss curves.

Trigger Source	Mean $[ms] = \bar{x}$	Variance $[ms^2] = \sigma_{\bar{x}}^2$
Software	99.99	0.000752
Frequency generator	100.00	0.000414
Microcontroller	100.03	0.000715

Table 9.2: This table show us the mean value and variance for the three trigger sources, which will be use to plot the Gauss curves.

The Gauss curves are plotted by using the following equation:

$$P_{\bar{x}}(x) = \frac{1}{\sqrt{2\pi\sigma_{\bar{x}}^2}} e^{-\frac{(x-\bar{x})^2}{2\sigma_{\bar{x}}^2}},$$

Here is $\bar{x}$ and $\sigma_{\bar{x}}^2$ from table 9.2.

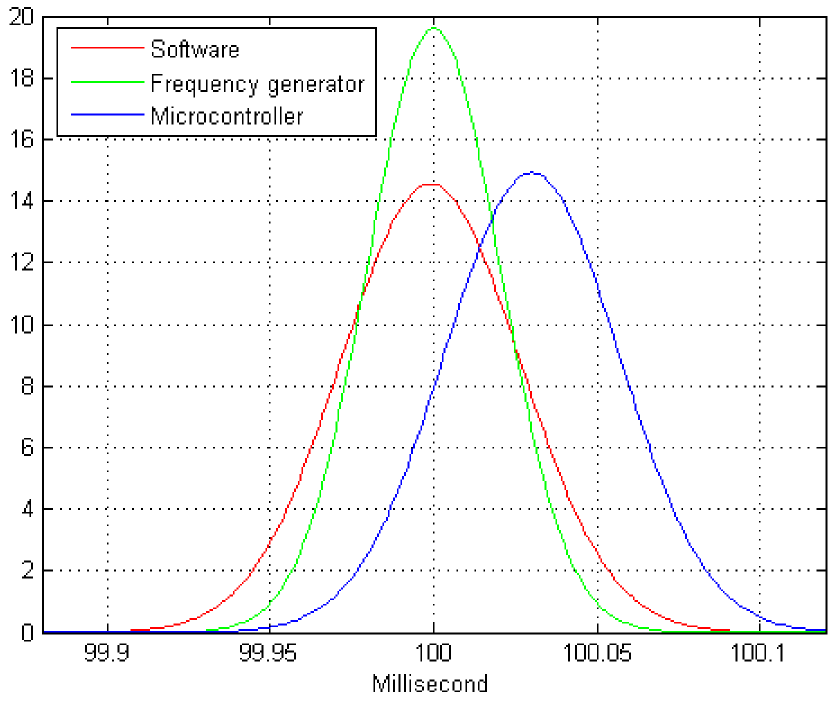


Figure 9.1: This is three probability density functions, one for each of the three trigger sources.

In figure 9.1 we see the Gauss curves, also called probability density functions. A probability density function will be tall when the variance is small.

A trigger source with a small variance will be more stable than a trigger source with a big variance. When we say stable we mean that the expected trigger signal is received when we expect it to be received.

9.3.1 Problem with the Software Trigger

When we were to check the variance of the software trigger we got extreme variations, values between $\sigma_x^2 = 0.0030$ and $\sigma_x^2 = 2.98$. To identify the problem we need to check what priorities the computer do.

The computer we are using is low on memory, so it is hard to do many things at the same time. The solution was to wait for the computer to be done with the startup sequence and make the computer do nothing else than taking pictures, this implies writing calculations out to screen when running the script for taking pictures.

The only thing the computer now do is to send the trigger signal and store all data from the camera. After we have discovered these sources we took 5 new series with 20 pictures and the result is written in table 9.1 on page 45.

Chapter 10

Discussion

The assembled system is now able to decrease the operational temperature in the camera. We want the temperature to be stable around 0-1 °C. It is also able to take pictures in the presence of an external trigger and we have showed that there is a difference in precision and stability, whether you use an external or a software trigger.

10.1 Pictures After Cooling.

To show the improvements of the cooling of the camera we have plotted the mean values of pictures versus the temperature of the camera. As we described in chapter 4 we would expect the mean value to decrease as we decreased the temperature of the camera.

To show the dark current noise we have removed the readout noise from the pictures. The shot noise is not removed from the pictures because it will eliminate the dark current as well.

From the theory we discussed in chapter 4 about the shot noise we can see that it is not dependent of temperature. The noise that is dependent of temperature is referred to as dark current. We can therefore say that the shot noise is not dependent of temperature, and will not contribute a lot to the total noise we can see in the mean value, see figure 7.5 on page 37 and figure 7.6 on page 38.

In these figures we see that the mean value decrease as the themperature decrease. Hence we see that the temperature is a visible factor for this dark current.

10.2 Stability in the Three Trigger Sources.

We study the figure 9.1 on page 46. From this figure we can see that the microcontroller had an offset from the wanted 100ms. We see this offset both in the figure and the two tables 9.1 on page 45 and 9.2 on page 45.

The reason for the offset in the microcontroller trigger source are the timing issues (chapter 3) we got when we use commands instead of clock signals. In Appendix C we can see that we have used a delay operation in the code.

Since the instruction fetch and instruction execute from figure 3.5 on page 13 is as long as a clock cycle we would think that the code would make a square pulse with a frequency of 10 Hz. This is not the reality.

There might be a lot of uncertainties for this offset, and one of them could be that it takes time for the microcontroller to change the voltage on the used pin from high to low or the other way around.

We see from figure 9.1 on page 46 that the frequency generator is the best trigger source out of the three tested sources. This is quite reasonable since the function generator is produced to make a precise pulse in its operative frequency range, either if it is a sine, triangle or a square pulse. Operative frequency range for a square pulse is $100\mu Hz$ - 15 MHz

The other two trigger sources are not as precise as the function generator, because they are produced to do other tasks in addition to make a square pulse at a frequency of 10Hz.

Chapter 11

Conclusion

We have developed hardware that have been used to measure and control the operational temperature inside a Guppy F044 NIR camera. We achieved a resolution = 0.03 degrees Celsius for each digital value on the thermometer. With this resolution it is easy to keep the temperature in the camera stable around 0-1 °C.

We conclude from the discussion that the cooling of the camera have a positive effect on the quality of the pictures.

The software that have been developed is attached as appendices(Appendix B, C, D and E). The first two softwares are developed to program the camera and the microcontroller (Appendix B and C). The last two softwares are developed to view the quality of the pictures as we vary the temperature in the camera and plot the Gauss curves to the three trigger sources (Appendix D and E).

We have succesfully used three different trigger sources on the camera, and by ploting the probability density functions we have been able to see the difference between them.

We conclude from the discussion and the measurements that the frequency generator is the most precise and stable trigger source for this camera.

Bibliography

[1] Supercool AB. Thermoelectric modules. http://www.ckitaly.com/pdf/controllo_temperatra/moduli_peltier/moduli_peltier.pdf, January 2005.

[2] Inc. Agilent Technologies. Agilent fundamentals of rf and microwave noise figure measurements. http://cas.web.cern.ch/cas/Denmark-2010/Caspers/Fundamentals%20on%20RF%20noise%20figure%20measurements%20AN%2057-1%20CAS2010.pdf, May 2010.

[3] John P Bentley. *Principles of Measurement Systems, 4th Edition.* Prentice Hall, 2005. Page 205-214, Deflection bridge.

[4] Atmel Corporation. 8-bit microcontroller with 16/32/64k bytes in-system programmable flash. http://www.atmel.com/dyn/resources/prod_documents/doc8011.pdf, January 2010.

[5] Damien Douxchamps. libdc1394 homepage. http://damien.douxchamps.net/ieee1394/libdc1394/, January 2008.

[6] M. Colleen Gino. Noise, noise, noise. http://astrophys-assist.com/educate/noise/noise.htm, September 2002.

[7] Philip Greenspun. Aerial photography. http://photo.net/learn/aerial/primer, February 2007.

[8] David M Pozar. *Microwave Engineering, 3rd Edition.* Wiley, 2004. Page 486-497, Noise Temperature.

[9] Sarah Cox Richard Barnett. *Embedded C Programming & the Atmel AVR, 2nd Edition Book/CD.* Delmar Publishing, 2006.

[10] Michael Richmond. Readout noise, and total noise. http://spiff.rit.edu/classes/phys559/lectures/readout/readout.html, December 2004.

[11] Inc. RTI Electronics. Precision temperature measurement and control devices interchangeables ntc thermistors. http://www.datasheetarchive.com/datasheet-pdf/046/DSA0052529.html, January 2005.

[12] Source Author: John R Taylor. English: Compendium, fys1003, introduction to experimental physics. norwegian: Kompendium, fys1003, grunnkurs i eksperimentell fysikk, 2006.

[13] Allied Vision Technologies. Technical manual v7.1.0. http://www.alliedvisiontec.com/fileadmin/content/PDF/Products/Technical_Manual/Guppy/Guppy_TechMan_V7.1.0_en.pdf, May 2009.

[14] Allied Vision Technologies. Hardware installation guide v6.0.0. http://www.alliedvisiontec.com/fileadmin/content/PDF/Products/All_Hardware_Installation_Guide/All_HardInst_V6.0.0_en.pdf, May 2010.

Appendix A

Weighted Averaging

Bibliography:[12]

Assume that we have calculated the mean value for a number of picture series, we have also found the standard deviation:

$$\begin{aligned} \text{Case A: } v &= v_A \pm \sigma_A \\ \text{Case B: } v &= v_B \pm \sigma_B \end{aligned}$$

Since these results will vary a little bit, we still would expect the results to represent the same mean value. How can we calculate the final mean value, v_{best}, and variance, σ^2_{vbest}?

We can use the principle of maximum probability to find these parameters. If we assume that the calculated values represent a normal distribution round a unknown, true value V, then the probability to find v_A is:

$$P_V(v_A) \propto \frac{1}{\sigma_A} e^{-\frac{(v_A - V)^2}{2\sigma_A^2}},$$

while the probability to find v_B is:

$$P_V(v_B) \propto \frac{1}{\sigma_B} e^{-\frac{(v_B - V)^2}{2\sigma_B^2}}.$$

The probability for finding v_A in case A and v_B in case B is given by:

$$P_V(v_A, v_B) = P_V(v_A)P_V(v_B) \propto \frac{1}{\sigma_B \sigma_A} e^{-\frac{\chi^2}{2}} \tag{A.1}$$

here is χ^2:

$$\chi^2 = (\frac{v_A - V}{\sigma_A})^2 + (\frac{v_B - V}{\sigma_B})^2 \tag{A.2}$$

The best estimate for V will be where we have maximum in equation A.1 on the preceding page, which is where we have a minimum in A.2 on the previous page.

We derive equation A.2 on the preceding page with respect to V and set the equation equal to zero, to find the best estimate to our v_{best} value

$$2(\frac{v_A - V}{\sigma_A}) + 2(\frac{v_B - V}{\sigma_B}) \quad = \quad 0$$

If we solve this equation for V, we find the best estimate v_{best}.

$$v_{best} = V \quad = \quad \frac{\frac{v_A}{\sigma_A^2} + \frac{v_B}{\sigma_B^2}}{\frac{1}{\sigma_A^2} + \frac{1}{\sigma_B^2}}$$

We simplify our v_{best} equation by defining weights:

$$w_A = \frac{1}{\sigma_A^2}, w_B = \frac{1}{\sigma_B^2}$$

We get:

$$v_{best} \quad = \quad \frac{w_A v_A + w_B v_B}{w_A + w_B}$$

This equation is a weighted avering of our input cases. We compare this equation to the center of gravity of two bodies and see that w_A and w_B is equivalent to the weights of the two bodies, and v_A and v_B is their positions.

This result could be generalized to combine even more measurements. Assume that we got N separated calculations of the mean value x.

$$\begin{aligned} x_1 &\pm \sigma_1, \\ x_2 &\pm \sigma_2, \\ &\ldots, \\ x_N &\pm \sigma_N, \end{aligned}$$

here is σ_1, ..., σ_N the corresponding standard deviations.

The best estimatad value, x_{best}, will be the weighted averaging:

$$x_{best} \quad = \quad \frac{\Sigma_{i=1}^N w_i x_i}{\Sigma_{i=1}^N w_i}$$

The expression for the standard deviation, σ_{xbest}, is derived from the basic formula for error propagation.

$$\sigma_{xbest} \quad = \quad (\Sigma_{i=1}^N w_i)^{-\frac{1}{2}}$$

Appendix B

Software to Program and Control the Camera

```
/**********************************************************************
**      Bibliography:[5]
**      Title: Operate the Guppy F044 NIR camera
**      by: Anders Kristiansen
**      $Revision: 23. juni 2010
**
**-----------------------------------------------------------------
**
**********************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <fcntl.h>
#include <inttypes.h>
#include <sys/time.h>
#include <sys/mman.h>
#include <sys/ioctl.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>
#include <signal.h>
#define _GNU_SOURCE
#include <getopt.h>
#include "sys/timeb.h"
```

```
#include <libraw1394/raw1394.h>
#include "dc1394/dc1394.h"
#include "dc1394/vendor/avt.h"

#define DROP_FRAMES DC1394_RING_BUFFER_LAST
#define MAX_PORTS 9
#define NUM_BUFFERS 9

#define GUPPY1_ID 0x000a47010f08727fLLU // This is the true
// address of the Guppy F044 NIR
#define CAMERA_OFF 0
#define CAMERA_ON 1

/* Declarations for libdc1394 */
dc1394camera_t *camera;
dc1394featureset_t features;
dc1394video_frame_t *Videoframe;

/* Other declarations */
int res;
char *frame_buffer=NULL, datafil[17];
int gain;
int exp_time;
int loop, imageloop;
struct timeval start, end;
int shotresult=1; //DC1394_SUCCESS = 0!
int camera_status=CAMERA_OFF;
FILE *fd;

void cleanup(void) {
    if (frame_buffer != NULL)
        free( frame_buffer );
}

/* Trap ctrl-c */
void signal_handler( int sig) {
    signal( SIGINT, SIG_IGN);
    cleanup();
    exit(0);
}
```

```
/*This function writes the picture to the computer*/

int skriv_datafil(dc1394video_frame_t *Vframe, char *filnavn)
{
    int i, j, k, imax, jmax;
    char output_buffer[1024];
    char buf_0[] ="#0000 000 ";
    char buf_1[] ="00000 000 ";
    char buf_2[] ="#0000000000000000 ";
    int utfil;

    imax = (int) Vframe->size[0];
    jmax = (int) Vframe->size[1];
    k = 0;

    // The line below writes the timestamp to the file.
    fprintf(fd, "%llu\t", Vframe->timestamp);
    utfil = open(filnavn, O_WRONLY | O_CREAT, S_IRUSR | S_IWUSR);
    write(utfil, "P5\n", 3);

    sprintf(buf_2, "#%llu\n", Vframe->timestamp);
    write(utfil, buf_2, 18);
    sprintf(buf_0, "#%05d %03d\n", exp_time, gain);
    write(utfil, buf_0, 11);
    sprintf(buf_1, "%03d %03d\n", imax, jmax);
    write(utfil, buf_1, 8);

    write(utfil, "255\n", 4);
    for (j=0;j < jmax;j++) {
        for (i=0;i < imax;i++) {
            output_buffer[k] = Vframe->image[i + j*imax]; // writing lsb
            k++;
            if (k == 1024) {
                write(utfil, output_buffer, 1024);
                k = 0;
            }
        }
    }
    if (k > 0)
      write(utfil, output_buffer, k);
    close(utfil);
    return 0;
}

int main(int argc,char *argv[])
{
    unsigned int speed;
    int t = 0;
    int y = 1;
    dc1394error_t err;
    dc1394_t * d;
    dc1394camera_list_t * list;
    uint32_t extexp;

    // Make the program user friendly.
    if(argc != 3){
        printf("usage: %s Shutter gain\n", argv[0]);
        exit(0);
    }

    exp_time = atoi(argv[1]);
    gain= atoi(argv[2]);
    imageloop = 20;
    extexp = (uint32_t) exp_time*1000;

    char serie[10];
    serie[0] = '0';
    serie[1] = '\0';

    // Checks if we got a file already open. This file
    // is where we write the timestamps to.
    while(fopen(serie, "r")){
        (*serie)++;
    }
    fd = fopen(serie, "w");

    d = dc1394_new ();
    err=dc1394_camera_enumerate (d, &list);
    DC1394_ERR_RTN(err,"Failed to enumerate camera");
    // Checks if we got a camera connected
    if (list->num == 0) {
         dc1394_log_error("No camera found");
         return 1;
    }

    camera = dc1394_camera_new (d, list->ids[0].guid);
    if (!camera)
        dc1394_log_warning("Failed to initialize camera with guid %llx", list-
>ids[0].guid);

    dc1394_camera_free_list (list);

    /* Setup camera for capture */
    if(dc1394_feature_get_all(camera, &features) !=DC1394_SUCCESS){
        printf("unable to get feature set\n");
    } else {
        // dc1394_print_feature_set(&features);
    }

    if (dc1394_video_get_iso_speed(camera, &speed) != DC1394_SUCCESS)
    {
         printf("unable to get the iso speed\n");
         cleanup();
         exit(-1);
    }
    /*Settings*/
    camera_status = CAMERA_ON;
    res=DC1394_VIDEO_MODE_FORMAT7_2; // 2x2 binning
    dc1394_video_set_operation_mode(camera, DC1394_OPERATION_MODE_LEGACY);
    dc1394_video_set_iso_speed(camera, DC1394_ISO_SPEED_400);
    dc1394_video_set_mode(camera,res);
    dc1394_format7_set_color_coding(camera, res, DC1394_COLOR_CODING_MONO8);
     dc1394_avt_set_timebase(camera,9);  //  9=1ms,  8=500us,  7=200us,
6=100us, 5=50us, 4=20us
    dc1394_avt_set_extented_shutter(camera, extexp);
    dc1394switch_t pwr = DC1394_ON;
    dc1394_feature_set_value(camera, DC1394_FEATURE_GAIN, gain);
    // The line below needs to be commented if we use an external trigger.
    dc1394_software_trigger_set_power(camera, DC1394_ON);
    //dc1394_external_trigger_set_mode(camera, 0);
    //dc1394_external_trigger_set_power(camera, pwr);

    // The two lines above are used when we got an external trigger
    // rigged to the system.

    if (dc1394_capture_setup(camera, 4,DC1394_CAPTURE_FLAGS_DEFAULT)
!= DC1394_SUCCESS) {
        fprintf(stderr, "unable to setup camera- check line %d of %s to make
sure\n",__LINE__,__FILE__);
        perror("that the video mode,framerate and format are supported\n");
        printf("is one supported by your camera\n");
        cleanup();
        exit(-1);
    }

    //If we use an external trigger we need to comment out the
    // five lines below marked with: "//comment"

    /* main event loop */
    gettimeofday(&start, NULL); //comment
    loop = 0;
   while(loop<imageloop){

       while(t<100000*y){ //comment
           gettimeofday(&end, NULL); //comment
           t = (end.tv_sec*1000000 + end.tv_usec)-(start.tv_sec*1000000
           + start.tv_usec); //comment
       } //comment
       y=y++;
      // The line below starts the exposure in the case of a
      // software trigger, if else it waits here for the trigger
      // signal in the case of an external trigger.
       shotresult = dc1394_video_set_one_shot(camera, DC1394_ON);

        if (shotresult == DC1394_SUCCESS) {

            if (dc1394_capture_dequeue(camera, DC1394_CAPTURE_POLICY_WAIT,
&Videoframe)!=DC1394_SUCCESS)
                printf("Error: Failed to capture from GUPPY\n");

            if (Videoframe) {
                sprintf(datafil, "\n", loop);
                skriv_datafil(Videoframe, datafil); // here we write to file.
                dc1394_capture_enqueue (camera, Videoframe);
            }
        }
        loop++;
    }
    exit(0);
}
```

Appendix C

Software to Program and Control the ATmega324PA

Bibliography: [9]

```
#include <stdio.h>
#include <mega324.h>
#include <delay.h>
#define ADC_VREF_TYPE 0x00


unsigned int i, j, max_temp, min_temp;
unsigned int x, x_min, x_max, x_measure, x_temp;

// Read the AD conversion result
unsigned int read_adc(unsigned char adc_input)
{
    ADMUX=adc_input | (ADC_VREF_TYPE & 0xff);
    // Delay needed for the stabilization of the ADC input voltage
    delay_us(10);
    // Start the AD conversion
    ADCSRA|=0x40;
    // Wait for the AD conversion to complete
    while ((ADCSRA & 0x10)==0);
        ADCSRA|=0x10;
    return ADCW;
}

void main(void)
```

```
{
    // Crystal Oscillator division factor: 1
    #pragma optsize-
    CLKPR=0x80;
    CLKPR=0x00;
    #ifdef _OPTIMIZE_SIZE_
    #pragma optsize+
    #endif

    // Input/Output Ports initialization
    // Port A initialization
     // Func7=In Func6=In Func5=In Func4=In Func3=In Func2=In
Func1=In Func0=In
     // State7=T State6=T State5=T State4=T State3=T State2=T
State1=T State0=T
    PORTA=0x00;
    DDRA=0x00;

    // Port B initialization
     // Func7=Out Func6=Out Func5=Out Func4=Out Func3=Out
Func2=Out Func1=Out Func0=Out
    // State7=0 State6=0 State5=0 State4=0 State3=0 State2=0 State1=0
State0=0
    PORTB=0x00;
    DDRB=0xFF;

    // Port C initialization
     // Func7=In Func6=In Func5=In Func4=In Func3=In Func2=In
Func1=In Func0=In
     // State7=T State6=T State5=T State4=T State3=T State2=T
State1=T State0=T
    PORTC=0x00;
    DDRC=0x00;

    // Port D initialization
     // Func7=In Func6=In Func5=In Func4=In Func3=Out Func2=In
Func1=In Func0=Out
     // State7=T State6=T State5=T State4=T State3=T State2=T
State1=T State0=0
    PORTD=0x00;
    DDRD=0x09;
```

```
// Timer/Counter 0 initialization
// Clock source: System Clock
// Clock value: Timer 0 Stopped
// Mode: Normal top=FFh
// OC0A output: Disconnected
// OC0B output: Disconnected
TCCR0A=0x00;
TCCR0B=0x00;
TCNT0=0x00;
OCR0A=0x00;
OCR0B=0x00;

// Timer/Counter 1 initialization
// Clock source: System Clock
// Clock value: Timer1 Stopped
// Mode: Normal top=FFFFh
// OC1A output: Discon.
// OC1B output: Discon.
// Noise Canceler: Off
// Input Capture on Falling Edge
// Timer1 Overflow Interrupt: Off
// Input Capture Interrupt: Off
// Compare A Match Interrupt: Off
// Compare B Match Interrupt: Off
TCCR1A=0x00;
TCCR1B=0x00;
TCNT1H=0x00;
TCNT1L=0x00;
ICR1H=0x00;
ICR1L=0x00;
OCR1AH=0x00;
OCR1AL=0x00;
OCR1BH=0x00;
OCR1BL=0x00;

// Timer/Counter 2 initialization
// Clock source: System Clock
// Clock value: Timer2 Stopped
// Mode: Normal top=FFh
// OC2A output: Disconnected
```

```
    // OC2B output: Disconnected
    ASSR=0x00;
    TCCR2A=0x00;
    TCCR2B=0x00;
    TCNT2=0x00;
    OCR2A=0x00;
    OCR2B=0x00;

    // External Interrupt(s) initialization
    // INT0: Off
    // INT1: Off
    // INT2: Off
    // Interrupt on any change on pins PCINT0-7: Off
    // Interrupt on any change on pins PCINT8-15: Off
    // Interrupt on any change on pins PCINT16-23: Off
    // Interrupt on any change on pins PCINT24-31: Off
    EICRA=0x00;
    EIMSK=0x00;
    PCICR=0x00;

    // Timer/Counter 0 Interrupt(s) initialization
    TIMSK0=0x00;
    // Timer/Counter 1 Interrupt(s) initialization
    TIMSK1=0x00;
    // Timer/Counter 2 Interrupt(s) initialization
    TIMSK2=0x00;

    // Analog Comparator initialization
    // Analog Comparator: Off
    // Analog Comparator Input Capture by Timer/Counter 1: Off
    ACSR=0x80;
    ADCSRB=0x00;

    // ADC initialization
    // ADC Clock frequency: 156,250 kHz
    // ADC Voltage Reference: AREF pin
    // ADC Auto Trigger Source: Free Running
    // Digital input buffers on ADC0: On, ADC1: Off, ADC2: Off, ADC3:
Off
    // ADC4: Off, ADC5: Off, ADC6: Off, ADC7: Off
    DIDR0=0xFE;
    ADMUX=ADC_VREF_TYPE & 0xff;
    ADCSRA=0xA7;
    ADCSRB&=0xF8;

    j = 0;

    while (1){
        i=0;
        x_temp =0;
        x_min = 10000;
        x_max = 1;
        for (i = 0; i < 30; i++){
            x_measure = (read_adc(PINA.0));
            x_temp = x_temp + x_measure;
        }

        x = (x_temp/(i));
        delay_ms(100);
        PORTB = x;
        delay_ms(1000);

        min_temp = 984;
        max_temp = 982;
        if (x > min_temp){
            PORTD.0 = 0;
        }

        if (x < max_temp){
            PORTD.0 = 1;
        }

        //The following four lines are used when we use the
        //microcontroller as an external trigger source,
        //all other lines inside the while-loop then
        //needs to be commented out.
        //PORTD.3 = 0;
        //delay_ms(50);
        //PORTD.3 = 1;
        //delay_ms(50);
    };
}
```

Appendix D

Calculations of the PDFs to the Trigger Sources

```
% Here we plot the probability density functions for the three
% different trigger sources.

clc

% The following values have been calculated with weighted
% averaging from the values listed in table 9.1.

% Software
sforv = 99.999;
svari = 0.000752;
% Frequency generator
fforv = 100;
fvari = 0.000414;
% Microcontroller
eforv = 100.03;
evari = 0.000715;

j=1;
min = 99.88;
max = 100.12;
x = min:1/1000:max;

for i=1:((max-min)*1000)+1
    sy(j) =(1/(sqrt(2*pi*svari)))*exp(-(((x(i)-sforv)^2)/(2*(svari))));
    fy(j) =(1/(sqrt(2*pi*fvari)))*exp(-(((x(i)-fforv)^2)/(2*(fvari))));
    ey(j) =(1/(sqrt(2*pi*evari)))*exp(-(((x(i)-eforv)^2)/(2*(evari))));
    j=j+1;
end

figure(1)
plot(x,sy,'r'),hold on;
plot(x,fy,'g'),hold on;
plot(x,ey,'b');
legend('Software','Frequency generator','Microcontroller',3);
xlabel('Millisecond');
ylabel('p_x(x)');
grid on;
```

Appendix E

Software to Calculate the Mean Value of Pictures

```
% Find the mean value of pictures.

clc
M = 7; % Number of pictures
mA = zeros(1,M);

% Naming vectors
n = 1:M;
name = num2str(n);
name(name==' ') = [];
l = 9;
readout = num2str(l);
readout(readout==' ') = [];

for j = 1:M
    A = imread([name(j),'.bmp'],'bmp'); % Loading bottom right pixels
    B = imread([readout(1),'.bmp'],'bmp'); % Loading readout noise
    a = double(A);
    b = double(B);
    s = b(:);
    t = a(:);
    v = t-s;
    mA(j) = mean(v);
end

temp = [0 5 10 15 20 25 30];
```

```
figure(1);
stem(temp,mA,'o')
axis([-5 35 2 20])      % Axis used when we plot the
        % serie taken with gain = 650.
% axis([-5 35 2 4])     % Axis used when we plot the
        %serie taken with gain = 0.
xlabel('Temperature [C]')
ylabel('Mean of pixel value')
grid on
```