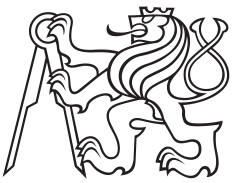
Master's Thesis



Czech Technical University in Prague

F3

Faculty of Electrical Engineering
Department of Measurement

Measurement Camera for Teaching Labs

Bc. Jakub Vodseďálek
Cybernetics and Robotics
Sensors and Instrumentation

2019

Supervisor: doc. Ing. Jan Fischer, CSc.

MASTER‘S THESIS ASSIGNMENT

I. Personal and study details

Student's name: **Vodseďálek Jakub** Personal ID number: **434677**

Faculty / Institute: **Faculty of Electrical Engineering**

Department / Institute: **Department of Measurement**

Study program: **Cybernetics and Robotics**

Branch of study: **Sensors and Instrumentation**

II. Master’s thesis details

Master’s thesis title in English:

Measurement Camera for Teaching Labs

Master’s thesis title in Czech:

Měřicí kamera pro výukové laboratoře

Guidelines:

Design and realize a measurement camera for educational laboratories. The camera will serve for calculations verification experiments associated with the choice of the optical system. Also, it will serve for contactless dimension measurement experiments and evaluation of lenses geometry flaws.
The camera will use area CMOS image sensor and microcontroller series STM32H7xx (or other from the series STM32xxx). The microcontroller will be used to set the sensor and transfer image data via the USB interface to the PC. The camera allows both linear and area operation modes so it can replace existing cameras in currently used modules.
Create the necessary software for the microcontroller as well as the parent PC concerning the use of the camera in the laboratory tasks.

Bibliography / sources:

[1] STMicroelectronics: DS12117 STM32H753VI Data
[2] STMicroelectronics: AN5020 Application note Digital camera interface (DCMI) for STM32 MCUs
[3] STMicroelectronics: RM0433 Reference manual STM32H7x3 advanced ARM-based 32-bit MCUs

Name and workplace of master’s thesis supervisor:

doc. Ing. Jan Fischer, CSc., Department of Measurement, FEE

Name and workplace of second master’s thesis supervisor or consultant:

Date of master’s thesis assignment: **12.02.2019** Deadline for master's thesis submission: **24.05.2019**

Assignment valid until:
by the end of summer semester 2019/2020

doc. Ing. Jan Fischer, CSc. Supervisor’s signature	Head of department’s signature	prof. Ing. Pavel Ripka, CSc. Dean’s signature

III. Assignment receipt

The student acknowledges that the master’s thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the master’s thesis, the author must state the names of consultants and include a list of references.

.	
Date of assignment receipt	Student’s signature

Declaration

I declare that the presented work was developed independently and that I have listed all sources of information used within it in accordance with the methodical instructions for observing the ethical principles in the preparation of university theses.

This thesis was created in the laboratory of videometry, Department of Measurement CTU FEE in Prague under the supervision of doc. Ing. Jan Fischer, CSc.

Prague, May 22, 2019 .....

Acknowledgement

I would like to thank all the people that helped me not only in my studies but also in my whole life. Special thanks go to my supervisor doc. Ing. Jan Fischer, CSc., who guided me and provided valuable advice during my academic studies. My parents deserve absolutely the biggest thanks and my dearest gratitude for all the support I received along my life journey. I would also like to appraise the patience of my girlfriend Maruška and thank her for enduring all the hardships with me. Last but not least, I would like to thank all my family and friends because without all of them I would not be where I am now.

Abstrakt

Tato diplomová práce se zabývá návrhem a realizací měřící kamery pro výukové laboratoře. Kamera podporuje několik CMOS obrazových senzorů řízených mikrořadičem z řady STM32. Snímky jsou přenášeny do PC s běžící vytvořenou aplikací pomocí USB. Podpora high speed USB je přidáná pomocí externího USB PHY na navržené propojovací desce. Kamera je schopna přenosu snímku v reálném čase, takže rozlišení podporovaných CMOS senzorů není omezeno. PC aplikace s multi platformní podporou umožňuje plnou kontrol CMOS senzoru. Přítomen je také simulovaný mód řádkového senzoru se základními nástroji pro analýzu řádku. V rámci práce je vytvořena propojovací deska mezi desku CMOS senzoru a desku mikrořadiče a také programy pro mikrořadič a PC.

Klíčová slova: CMOS obrazový sensor, kamera, STM32, STM32H743, USB, externí USB phy, libusb, platforma pro laboratorní výuku, videomterie, embedded programování, Qt, C++, C

Abstract

This master's thesis deals with the design and realization of a measurement camera for educational laboratories. The camera supports multiple CMOS image sensors controlled by STM32 series microcontroller. Images are transferred to PC running the developed application using USB. High speed USB support is added by external USB PHY on designed interfacing board. The camera is capable of real-time image data transfer, hence the resolution of supported CMOS sensors is unlimited. PC application with multiplatform support allows full CMOS sensor control. Simulated linear image sensor mode with simple line analysis tools is also included. Interfacing board between CMOS sensor board and microcontroller board and programs for microcontroller and PC are created within the work.

Keywords: CMOS image sensor, camera, STM32, STM32H743, USB, external USB phy, libusb, labs teaching platform, videometry, embedded programming, Qt, C++, C

Contents

Figures

Tables

Chapter 1
Introduction

Contactless measurement employing image sensors is an integral part of modern measurement systems. Thus it is also necessary to educate students in this particular field. Appropriate equipment is then necessary for educational laboratories. It is possible to use commercially available measurement cameras. However, their high price and not fulfilling all requirements does not make them an ideal solution.

This thesis deals with the design and realization of custom made measurement camera using a CMOS image sensor for educational laboratories. Unlike commercially available cameras, it can give us access to all sensors modes and settings while maintaining all the core functionalities. Using non-standard settings can help with the explanation of CMOS sensors properties and function, resulting in students better understanding of given problematic. The camera will have a construction similar to commercially available solutions and will use similar tools, thus it can help with understanding its operation as a whole (e.g., the process of capturing an image and sending to another system). The methodology of camera development is also described in this thesis and can help in the future with resolving problems encountered during the development.

Historically it was proven that single board camera might be the most elegant solution, however, it is not the most suitable one. Assembly of a single board containing a microcontroller, image sensor, and all other necessary components proves to be quite a challenging task. Hence it is problematic to replace the camera in case of hardware failure. For this reason, a modular solution using existing microcontroller development board is presented. This approach allows for more simple camera replacement. Multiple supported CMOS sensors also make it usable for comparing different CMOS sensor types or using it for different tasks.

Chapter 2
Analysis

The goal of this thesis is to create a camera for videometry teaching laboratories. The camera should be appropriate for calculations verification experiments associated with the choice of the optical system. Also, it should provide tools for contactless dimension measurement experiments and evaluation of lenses geometry flaws. A brief analysis of requirements and possible solutions is performed in this chapter.

2.1 Requirements

All of the entry requirements go around usability for teachings labs. That comes from the goal to replace more than one outdated system currently in use for Videometry and contactless measurement course. The current state is that every single laboratory measurement uses a different system. Some of them are, as of today, irreplaceable, and this particular problem should be addressed. The main requirements for the camera can be summed into the following points:

- Reliability
- Replaceability
- Usage in both area and line mode
- Changeable image sensors
- Expandability
- Complementing PC application

Now when the main requirements are defined, we can break down each one of them in more detail.

2.1.1 Reliability

Reliability is listed as the first for a good reason. It is one of the essential attributes for all systems. In this case, we have minimal tools to affect the reliability of used hardware. So the goal should be to at least not make it worse. What can be affected is software reliability, as firmware and PC application will be developed. In our hands is the choice of correct tools, including IDEs, libraries, compilers and the last but not least implementation itself. Developed software and firmware does not have to be completely bulletproof, because it will not be in commercial use. However, it should work reliably in correct use cases.

Communication between PC application and camera has always been problematic in the past. Possible problems and their solutions will be described in this thesis, as it will be the most focused part of reliability.

2.1.2 Replaceability

In the case of hardware failure, it might be crucial to have the ability to replace damaged parts quickly. This case is especially true for industrial application, but it also applies to teach purposes. So when designing the system for our cause, it is vital to keep it in mind.

Replaceability concerns are the main reason for not developing a single board camera solution. From the past, it was proven that these boards are hard to assemble; thus, the replacement is problematic.

2.1.3 Expandability

This point sums up three of mentioned requirements. In this case, expandability means specifically easy addition of another supported image sensor. The most important precondition for this is having the same physical interface on used image sensors. That goes in hand with changeable sensors, as accomplishing one will get us closer to other condition. It also relates to usage in both area and line mode. Using both modes with a single sensor would not be as practical as having the ability to change sensor depending on the use case.

All of this relates to our goal to create a camera for teaching labs. As mentioned, it should replace more than one system, and for that, we need some flexibility in configuration. This flexibility can be achieved by a changeable image sensor. Easy addition of another supported sensor is the primary concern for the future because the system can then be used even for new tasks.

2.1.4 Complementing PC application

To complete the whole system, the PC application needs to be developed to complement the camera itself. It should serve as an interface for the user to control the camera. Image data will also be streamed to PC. The application needs to include GUI because camera settings might be complicated, and visual representation of image data is required. Alongside camera control application should also include tools for laboratory measurements needed in teaching labs. Support for multiple operating systems would also be wanted addition for system adaptability.

2.2 Image sensors

Image sensors can be divided into two categories based on the used technology. The first technology is known under abbreviation CCD and the other one under abbreviation CMOS, which is also well known from different use cases. As the primary source of information for this section is used [1].

2.2.1 CCD image sensors

CCD stands for Charge-Coupled Devices, and it is historically older technology of the two mentioned. However, it still proves to have its advantages. Generally, they can have higher light sensitivity and dynamic range than CMOS sensors. Their noise characteristics are also usually better. From the disadvantages of CCD sensors must be named their price and also power consumption. They are also prone to effect called “blooming”, which can occur in case of oversaturation. What happens is that charge is “spilling” to neighborhood pixels, creating image distortion. This effect can be prevented by creating an anti-blooming gate between pixels (basically bigger insulation barrier). It is, unfortunately, increasing space between pixel results in lower pixel density.

Principle of CCD image sensors lies in MOS capacitor. Under electrode with a positive charge is created space charge (potential well). The positive charge on the electrode is then compensated by the accumulation of free charge carriers released by incident radiation.

Amount of accumulated charge can be expressed by formula:

$$Q = \frac{et\eta ES}{hv} \tag{2.1}$$

Where e is electron charge, t is time of accumulation, η is quantum efficiency, E is irradiance, S is pixel area, h is Planck constant and v is frequency of photon. From this relationship, we see that accumulated charge is linear dependant on the irradiation, meaning CCD image sensor response is linear before oversaturation.

To transfer charges form the structure are used charge shift registers. Different types of charge shift registers with 2,3 or more phase transfer exists. The charge is transferred by moving potential well created by a row of electrodes. Example of 3 phase transfer is shown in figure 2.1.

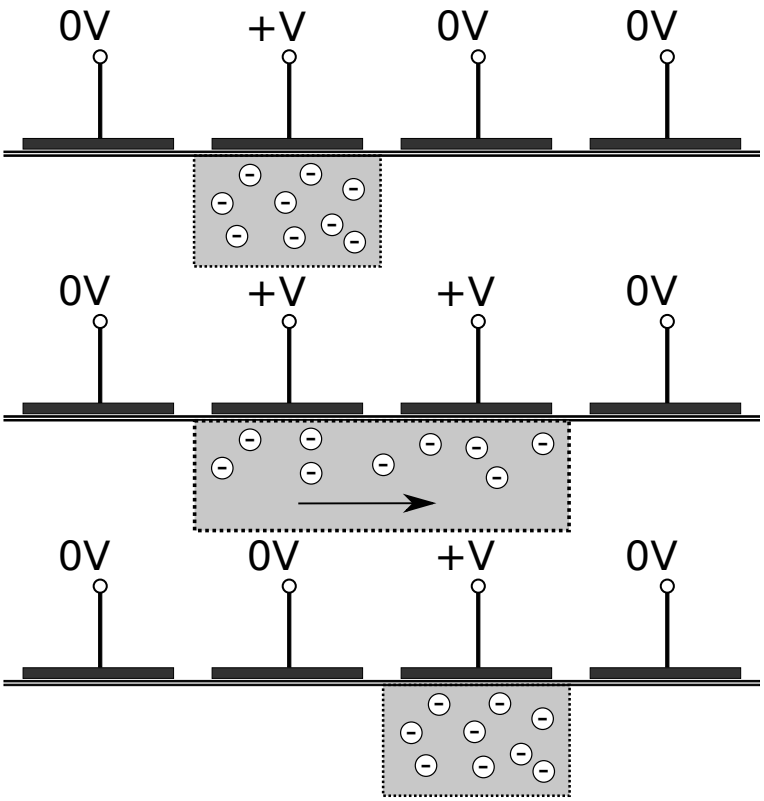


Figure 2.1. 3 phase CDD charge shift register functions

The efficiency of such a transfer is not 100 %, so some part of the charge is lost during the transfer. Transfer efficiency can be expressed as CTE (Charge Transport efficiency), which is a ratio between transferred charge Q' and charge before transfer Q.

$$CTE = \frac{Q'}{Q} \tag{2.2}$$

In conclusion, CCD image sensors are these days mostly used in specialized fields or high-end cameras. They still have some advantages thanks to their superb parameters. However, CMOS image sensors have been gradually improving in the past and are now the by far more used option.

2.2.2 CMOS image sensors

CMOS technology is very widely used also in other fields. Memories, processor or other integrated circuits often uses CMOS technology, hence the manufacturing process is widespread. Thus it lowers manufacturing cost and creates probably the most significant advantage of CMOS sensors - their price. Usability in mobile devices is also greatly improved, as power consumption can be up to ten times lower than for CCD sensors. Used CMOS technology allows simple integration of other circuits directly on the same chip (ADC, communication circuits, JPEG encoder, etc.). For CMOS image sensors, it is easier to achieve higher resolutions, as it is possible to have higher pixel density and thanks to the more tuned manufacturing process. From other advantages can be named faster readout speed, resulting in higher frame rates or a simple implementation of windowing, thanks to directly accessible individual pixels.

As a disadvantage can be counted lower sensitivity, caused by a smaller photosensitive area of a single pixel compared to its size. CMOS image sensor pixel area also includes a few transistors, reducing the photosensitive area. Noise characteristics of CMOS sensors are generally worse than of CCD sensors, making them less suitable for precise applications. However, during the latest years, CMOS sensors have improved, and more expensive ones can match CCD sensors.

CMOS image sensor function is based on a photodiode. From the function of photodiode comes sensors nonlinear dependence of output on irradiance. The photodiode can be embedded in the MOS transistor, which then serves as an integrated switch. Schematic drawing of such a transistor is shown in figure 2.2.

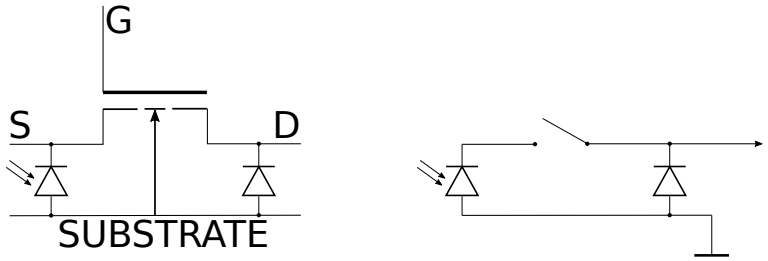


Figure 2.2. Photodiode in MOS transistor

The photodiode is working in the 3rd quadrant, and the principle schematic of CMOS image sensor work is shown in figure 2.3. The whole process can be simplified to a few steps:

- Charging of the capacitor in structure (reset)
- Exposition - discharging of the capacitor by the current generated by the photodiode
- Recharge of capacitor - the amount of delivered charge is proportional to irradiance
- Exposition
- Recharge
- etc.

In this case, the reading of values is destructive, and the structure is called a passive. However passive structures are these days used only in photodiode rows. CMOS image sensors use active structures were phase of reading and reset is divided. Thus the reading is nondestructive. Connection of individual elements to array is similar to DRAM or flash technologies. Individual elements are accessible with row and column selection.

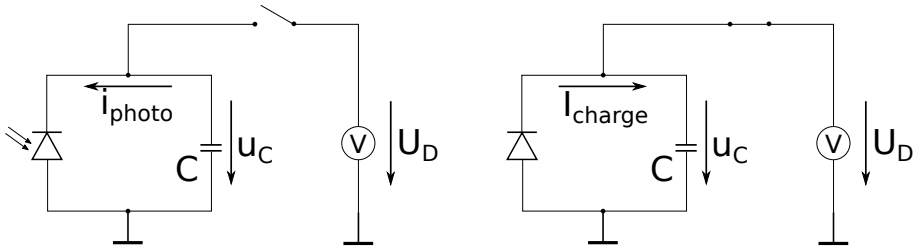


Figure 2.3. Principle of CMOS image sensor

These days CMOS image sensors have a dominant position on the market. First, they started to get popular thanks to their price. However, over the years, their parameters improved and are now very close to CCD sensors.

2.2.3 Shutter control

The image sensor can also be divided by used shutter control. Two options are well known - global and rolling shutter. Both CCD and CMOS image sensors can fundamentally use both of them. As sensors technology, shutter control types bring their advantages and disadvantages.

- **Global shutter**

  While using global shutter control, exposure of all photosensitive cells of the sensor is done at once. It means the whole shutter always has the same state - open or close. We can expect only movement distortion (blur) in case of moving object and too long exposure time. Thanks to the exposure of all cells at once, a global shutter sensor is suitable for capturing moving objects or other high-speed applications.

  Global shutter is usually used on CCD type sensor. However, it is also introduced in CMOS image sensors. While still maintaining its advantages mentioned above, it brings some disadvantages to CMOS sensors. First of all, the additional storage component for each cell is needed, as the whole image needs to be read at once. Image is then read sequentially line by line, as usual for CMOS image sensors. The delay between storage and read out may, unfortunately, lead to an increase of noise elements. Before starting the next exposure, the sensor needs to be cleaned from previously accumulated charges. All the additional steps mentioned above can lead to reducing the effective speed of sensor up to half compared to rolling shutter.

- **Rolling shutter**

  This type of shutter is mostly used in CMOS image sensors. Faster speed of the sensor is achieved by progressive exposure of individual lines. Before the exposure and readout of one image are finished, another can be already started. Process of rolling shutter exposure control is shown in figure 2.4

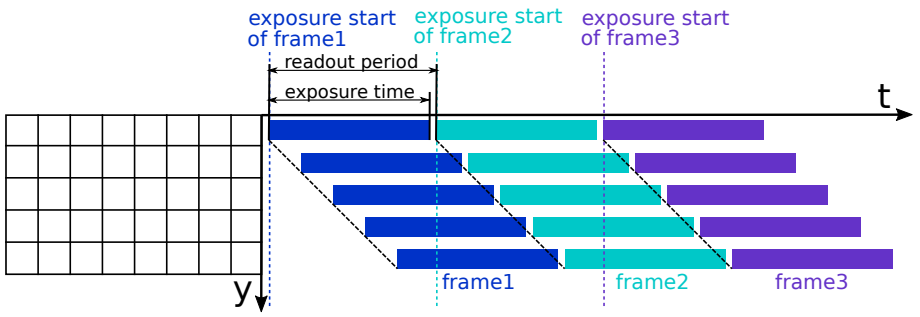


Figure 2.4. Rolling shutter exposure control

Readout time is not negligible and causes a delay between the exposure of individual lines. It can lead to image overlap, as seen in fig 2.4. Capturing a moving object can then result in distortion special for rolling shutter sensors (sometimes called spatial distortion). As the lines are captured at different times, a captured image of moving object can look as shown in figure 2.5.

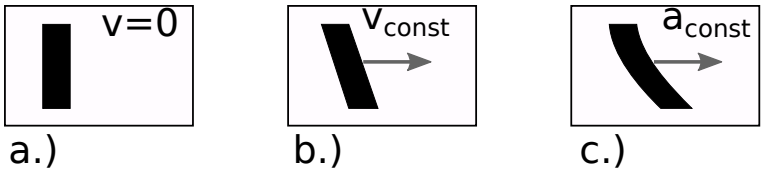


Figure 2.5. Rolling shutter spatial distortiion
a.) static object b.) constant speed c.) constant acceleration

However, the spatial distortion of a rolling shutter can also be used as an advantage. While detecting movements, we focus on finding distortion artifacts caused by a rolling shutter. Using this method cheaper and slower rolling shutter sensor can in some cases do the same job as more expensive fast global shutter sensor.

2.3 Controlling the sensor

A mean of controlling the sensor and also driving the communication with PC is needed. One of the possible choices would be microcontroller based on ARM® core. Two solutions are at hand for MCU usage. Custom MCU board could be designed, or already existing development board can be used. Custom PCB would probably be smaller, contain only necessary components also resulting in lower price. A physical interface for CMOS sensors could be integrated directly to this PCB so that no interfacing board would be necessary. On the other hand, using existing development board has its huge advantages as well. Most notably it preserves easy replaceability, as the damaged board could be easily swapped for new without a need of assembly. Not negligible is also a time-saving factor. Development of quite complex board would take a considerable amount of time, that can be invested in other areas of work. In the end, it was decided to go the easier path and use existing development board, as the replaceability factor is also important.

2.4 Communication

Camera system complemented with PC application needs various communication types. Two types of communication protocols are usually needed just between the image sensor and the controller. Another different communication protocol is then typically used for communication with the PC application. Regarding sensor communications, we are limited by sensors capabilities. There are not that many choices for communication with PC either, but there at least two meaningful possibilities.

2.4.1 Image sensor control

Image sensors commonly support only two types of communication protocols. One is SPI, sometimes called four-wire serial bus. The second one is then I^2C, so-called two-wire serial bus. Just from that, we can say, that they share many similarities and are used in similar

applications. Following is brief description of these buses, with focus on latter one (as it is used in our case).

1. **SPI**

   SPI is a synchronous serial communication bus. As already said, SPI is sometimes called a four-wire serial bus. That comes from the definition of four signals. SPI protocol is using master-slave architecture, typically supporting only a single master on a bus (unlike I^2C). Multiple slave devices can be connected to a single master. Another huge difference from I^2C is that SPI supports full duplex mode, which can not be achieved using only a two-wire interface. The following naming for defined signals is usually used:

   - SCLK - Serial Clock
   - MOSI - Master Output Slave Input
   - MISO - Master Input Slave Output
   - SS - Slave Select

   Single defined master node on bus controls SCLK signal frequency, which must be set to value supported by slave devices. Communication is initiated by master setting SS signal low to select a single slave node on bus. Following is always full duplex data on MOSI, MISO lines even when single direction data transfer is intended.

   Additionally, SPI supports configuration of SCLK polarity and phase by the master node. The most common names for these options are CPOL for clock polarity and CPHA for clock phase. CPOL 0 means SCLK is idle at 0 and cycles with pulses to 1. That also means the rising edge is the leading one. On the other hand, CPOL 1 means SCLK is idle at 1 and cycles with a pulse to 0. The leading edge is then the falling one. When setting CPHA to 0 one data cycle consists of first clock idle, and clock asserted. In opposite CPHA 1 means one data cycle consists of first clock asserted and then clock idle. Combinations of these options are referred to as modes. Established convention for mode numbering is captured in the following table 2.1.

SPI Mode	CPOL	CPHA
0	0	0
1	0	1
2	1	0
3	1	1

Table 2.1. Possible SPI modes

   Difference between SPI modes can be seen on figure 2.6.

   The most common SPI configuration with a single master and two slaves is shown in figure 2.7

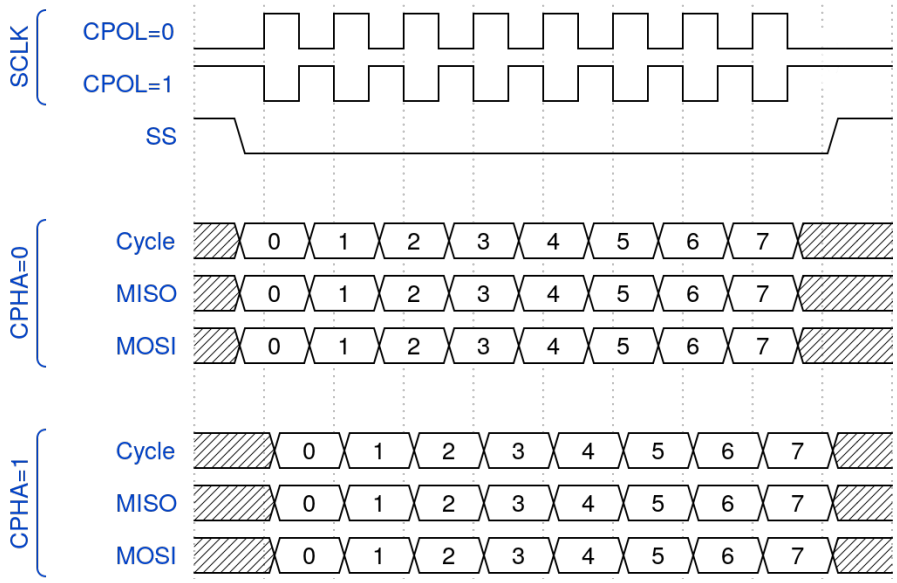


Figure 2.6. SPI modes timing

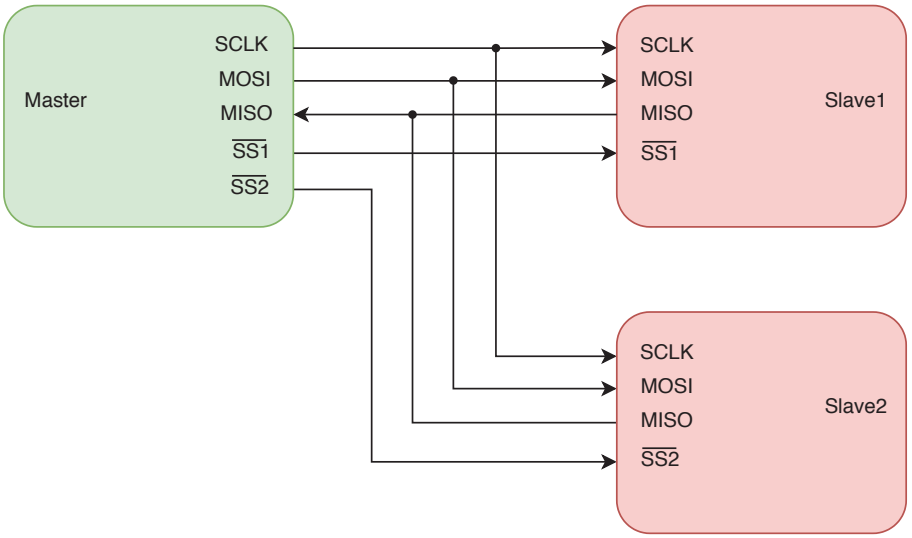


Figure 2.7. Typical SPI configuration with two slave devices

2. **I²C**

I²C bus uses only two wires. Same as SPI it is also synchronous serial communication bus. Its communication protocol is based on master-slave architecture too. With using only two wires, full duplex communication is not possible for synchronous transfer. So unlike SPI I²C is using bidirectional lines in half-duplex mode. It is possible to have multiple masters and multiple slaves on single I²C bus. Master selects different slaves by unique 7-bit (or 10-bit in some cases) address. Communication is limited to only 8-bit data packets. Another essential thing to note is that both buses lines needs to be connected to a pull-up resistor. I²C defines the following signals:

- SCL - Serial Clock
- SDA - Serial Data

Both SCL and SDA are high in idle state. A master node initiates communication by setting SDA to low and starts clock signal SCL. Follows 7-bit slave address with

8th bit choosing between writing to slave and reading. If the 8th bit is 1, it means read, and 0 means write. Then slave sends acknowledge bit by setting SDA to low. Following are 8 bits of data itself and acknowledge bit again. Communication can then continue by repeating the start condition or ended by stop condition - set SDA to high and stop clock signal. A clock signal is always controlled by the master. Typical clock frequencies are up to 100 kHz or in fast mode up to 400 kHz. The communication timing diagram for master reading data from a slave is shown in figure 2.8. Typical bus configuration with a single master and two slaves is shown in figure 2.9.

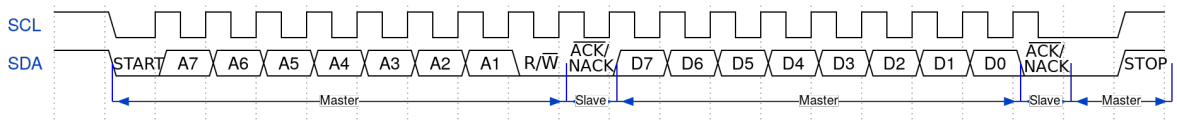


Figure 2.8. I²C communication timing

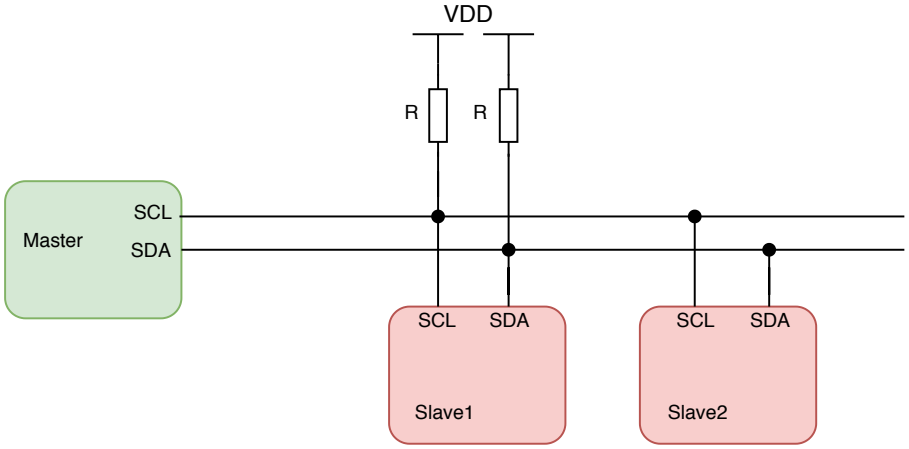


Figure 2.9. Typical I²C configuration with single master and two slaves

Both of those two have their advantages and disadvantages. The most notable ones can be summed into the following table 2.2.

SPI	I²C
Full duplex communication	Only half duplex capability
No pull-up resistors needed	Requires pull-up resistor
Potentially higher speed	Limited maximum clock speed
Transfer data length flexibility	Transfer data limited to 8-bit packets
Requires at least 3 wires	Requires always only 2 wires
No slave acknowledge	Acknowledge embedded in protocol
Only single master node	Supports multiple master nodes

Table 2.2. SPI and I²C comparison

As already mentioned, the choice of image sensor control bus is limited by hardware capabilities. If we omitted this limitation, the better choice would be I²C. There is no need for high data throughput, as all used transmits consist of few bytes. High data

throughput is needed only for image transfer, for which different interface is used. Needed is a connection of only a single slave to a single master with at least as possible wires. Connection of one slave to one master can be done of both SPI and I^2C. For I^2C speaks use of 2 wires, which can save us free pins on microcontroller package. Slave acknowledge embedded in communication protocol is just another point why choose I^2C over SPI in this particular project.

2.4.2 Image data transfer

For transfer of image data itself, another interface is used. CMOS sensors usually use a parallel interface with three synchronization signals. Parallel interface is used, because high data throughput is required (up to around 40 MB/s). Some microcontrollers from ST Microelectronics provides DCMI for this purpose. Focus in this chapter is on DCMI (for more detail see [2]), but other similar interfaces will have many similarities, as sensors interface is still the same.

Image data are transferred using DCMI by 8, 10, 12, or 14-bit data channels. More parallel data channels mean higher data throughput. Generally, the data throughput of DCMI can be changed by two factors. The first is already mentioned (number of parallel data channels) and the second is the frequency of PX clock signal. With that we get to data synchronization. For DCMI there are two possibilities.

1. **Hardware synchronization**

   Alongside PX clock signal, two other signals are used for hardware synchronization. PX clock signal serves as pixel data valid signal. The second synchronization signal is horizontal synchronization (HS). Its purpose is to distinguish horizontal blanking from valid pixel data. The last signal used signal is vertical synchronization (VS). This signal works similarly as HS, but for vertical blanking. Simplified it can be said that PX clock marks individual pixels, HS marks image lines, and VS marks whole images. How these signals might look like in reality is shown in figure 2.10. Polarities of all these three signals are programmable on the side of DCMI and also on the side of some CMOS sensors.

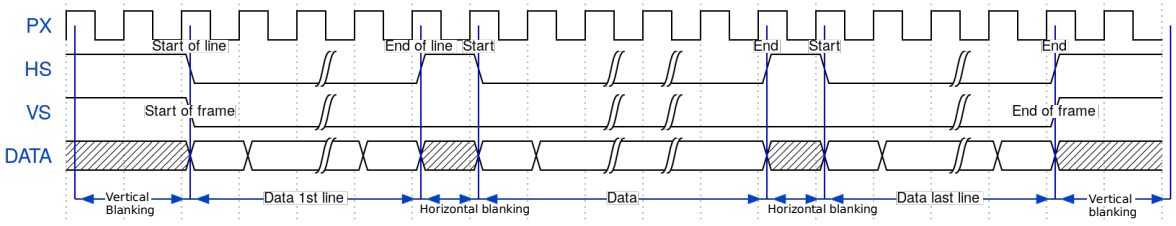


Figure 2.10. DCMI data transfer timing

2. **Embedded synchronization**

   Another option for data synchronization is to use only one dedicated synchronization signal PX clock. Frame and line synchronization are then coded directly into the data stream. By using embedded synchronization, we eliminate two additional connections, but it brings a few disadvantages. First of all, it is supported only in 8-bit parallel data mode. We also limit the number of possible data values from 256 to 254, as 0x00 and 0xFF are used for identification purposes. Synchronization code always consists of 4-byte value sequence starting with 0xFF 00 00. The last byte then codes the corresponding event.

DCMI supports two different capture modes. In snapshot, mode capture is controlled by setting capture bit. After setting capture bit of DCMI control register to 1, interface waits for the next start of the frame. When the frame capture is finished capture bit is automatically set to 0. The timing diagram for snapshot mode is shown in figure 2.11.

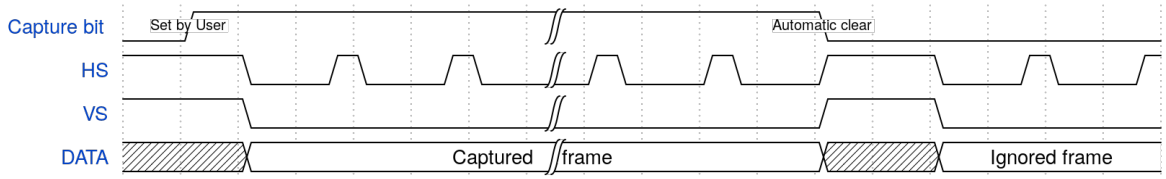


Figure 2.11. DCMI snapshot mode timing

In continuous grab mode user enables capture by setting capture bit and DCMI interface then continuously grabs following frames. For disabling capture, the user must manually set capture bit to 0. DCMI then continues to grab data until the current frame is finished. The timing diagram for a continuous grab mode is shown in figure 2.12. Additionally, setting for continuous grab mode is possible in the form of not capturing all available frames. Captured can be each frame, every 2nd or every 4th frame, effectively reducing bandwidth to 50% respectively 75%.

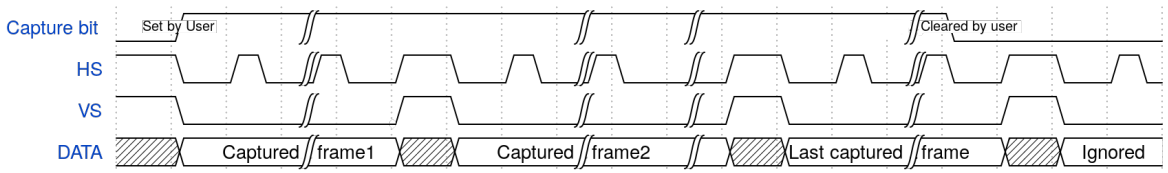


Figure 2.12. DCMI continuous grab mode timing

Multiple data formats are supported by DCMI:

- 8-bit monochrome or raw Bayer
- YCbCr 4:2:2
- RGB565
- JPEG

The most notable format for this application is 8-bit monochrome or raw Bayer, as it is the only one used. In this data format, every pixel is represented by 8-bit value. The same format can be applied for both monochrome and raw Bayer data, as RGB image is obtained from raw Bayer data by post-processing (described in section 6.1.1). Data register of DCMI is 32-bit and uses little-endian format (the least significant byte is stored at the smallest address). It means 4-pixel data can fit the DCMI data register at a time. Structure of the data register is shown in figure 2.13.

Figure 2.13. DCMI data register structure with monochrome data

The following formats represent directly full RGB image, unlike raw Bayer data. YCbCr format divides image data to three components. The most important being Y representing luminance (luma). Importance of luminance is represented by twice as dense raster than

for color differences (for one Cr and Cb sample two Y samples). Other two components are color differences (chromas), where Cb stands for the blue difference and Cr for the red difference. By simply omitting chromas monochrome image can be obtained directly from the luma component. Conversions between RGB and YCbCr formats can be simply expressed by two matrix equations:

$$\begin{bmatrix} Y \\ Cb \\ Cr \end{bmatrix} = \begin{bmatrix} 0 \\ 128 \\ 128 \end{bmatrix} + \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ -0.168 & -0.3313 & 0.5 \\ 0.5 & -0.4187 & -0.0813 \end{bmatrix} * \begin{bmatrix} R \\ G \\ B \end{bmatrix} \tag{2.3}$$

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1.402 \\ 1 & -0.3441 & -0.7141 \\ 1 & 1.772 & 0 \end{bmatrix} * \left(\begin{bmatrix} Y \\ Cb \\ Cr \end{bmatrix} - \begin{bmatrix} 0 \\ 128 \\ 128 \end{bmatrix} \right) \tag{2.4}$$

RGB565 uses a standard RGB format for representing color data. Each pixel is represented by 16 bits, where the first 5 bits encode blue component, next 6 bits encode green component, and last 5 bits encodes red component. The difference from raw Bayer data is that each pixel already has all three components. In the raw Bayer data format, each pixel has just one of these components.

Scan order of pixel raster for monochrome, YCbCr, and RBG format is shown in figure 2.14. Each word is 32-bit value in little-endian format (from DCMI data register format).

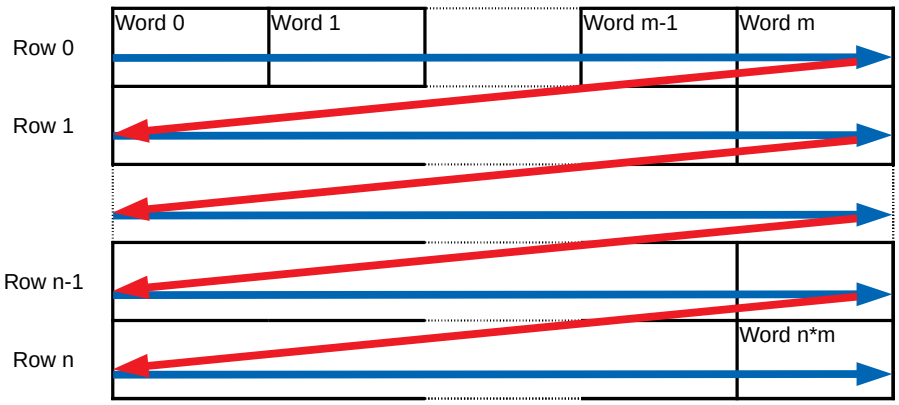


Figure 2.14. DCMI pixel scan order

The last supported data format by DCMI is compressed format JPEG. While using JPEG data format, we are limited to use only hardware synchronization, and DCMI crop feature is not available.

Other DCMI features as crop and Image resizing can substitute these functions if they are missing on CMOS sensor side. Crop feature allows as to select only part of the image to transfer (rectangular window). With that, we can effectively substitute the CMOS sensors windowing feature. Naturally, the frame rate of the sensor will not be increased, as in the case of windowing in the sensor. We choose relevant data from the whole frame on the DCMI side.

Skip and even binning feature can be substituted with image resizing capability of DCMI. It allows capture of all, only even or only odd lines. While capturing only one line out of two vertical resolution is reduced twice. For reducing horizontal resolution similar setting for individual bytes is available. All received data, every odd or even byte, two bytes out of four or one byte out of four can be captured. Using this setting, horizontal

resolution can be reduced twice or four times. Reducing horizontal resolution using DCMI might be tricky while using different data formats than monochrome.

Data transfer from DCMI data register is then usually done using DMA. Using DMA can save a lot of processor time, as much data needs to be transferred. After initialization DMA controller works independently on the processor core, saving processor time. However, DMA access to DCMI is limited to 32-bit alignment and can transfer maximally 0xFFFF 32-bit words on single configurations. To transfer the whole image, more DMA configurations usually needs to perform.

2.4.3 Camera and PC communication

Wireless communication (Wi-Fi or BlueTooth) has its advantages. However, as we would still need a power cable, it would make no sense to use wireless and deal with all of its disadvantages. Wired connections are always more reliable and usually also more comfortable to implement. From wired connections, the choice between Ethernet and USB must be made. Both of these have fast enough variants for image data transfers (at least few MB/s). We can also found both of these on most PCs.

In the end, it was chosen to use USB. It is the more versatile option, and a huge advantage is its availability. Ethernet connector is usually also present on most PC, but most of the time only once and is used for network connections. On the other hand, multiple USB ports are available. Different supported physical connectors also make it a more widely used option.

Many different options for USB speed or connectors are available at this point. Maximum data rates ranges from 1.5 MBit/s for Low Speed USB 1.0 up to 20 Gbit/s for SuperSpeed+ USB 3.2. With such a range of data rates and also different physical connectors it is a very versatile communication bus, that can be used in many applications.

From the first version USB 1.0, at least four pins are used in the USB connector. For backwards compatibility these 4 original connections are still employed 2.3.

1	2	3	4
+5V	D-	D+	GND

Table 2.3. Standard USB pinout

USB uses differential pair D- and D+ for data transfer. Two other pins, +5V and GND, can be used for supplying slave device and common ground connection. Only these four necessary pins are employed on USB A and B connectors. Micro and Mini versions of connector introduce the fifth pin called ID. It moved GND to the fifth position and is on the 4th pin. It these days it is only seldom used and serves for host/slave negotiation for USB OTG.

While using only one differential pair for communication, USB works in half-duplex mode. That changed with the introduction of USB 3.0. This standard introduces five additional pins, in summary having nine pins. It includes two more differential pairs, one for transmitting and second for receive. So from USB 3.0 standard USB can work in full duplex mode. The last added pin in the middle is ground.

In 2014 USB-C connector was standardized and is the latest addition to the USB connectors family. Total count of pins for USB-C is 24. USB-C connector can be unlike its predecessor connected independently on its orientation. So it has two sides, each containing the same 12 pins. One pin on each side is dedicated for mode negotiation. Two of them are dedicated to power delivery and two for ground. Thanks to that, USB-C can have a higher power rating than its predecessor. Two high-speed differential pair are

present on each side and also one additional low-speed link. The last two pins are then old differential pair for backward compatibility. USB-C connector can support the highest data rates thanks to 4 high-speed differential pair.

USB can also be divided using different standard versions, where each has a different maximal data rate. These version with respective data rates are shown in table 2.4. From the table we can see, that USB has come a long way from its beginnings to the current form, as it is more than 1500 times faster.

USB 1.0	USB 2.0	USB 3.0	USB 3.1	USB 3.2
1.5 MB/s	60 MB/s	625 MB/s	1250 MB/s	2500 MB/sr

Table 2.4. USB standard maximal data rates

2.5 Development tools

Proper choice of development tools is one of the crucial parts of every project. This work can be divided into two independent parts from development tools point of view. It is not only possible but instead expected to choose different tools for FW and PC application development. On both sides, few important things need to be chosen, including programming language, IDE, and compiler.

2.5.1 Firmware development

Possibilities for FW development would be nearly infinite, but our choices are already limited to a single platform. 32-bit ARM® processor from Cortex®-Mx lineup will be used as a controller. More specifically it will be a processor from STM32xxx series by ST Microelectronics.

Choice of hardware already substantially limits our choice of programming language. In the case of microcontrollers, the first choice might be assembly language. However, assembly is suitable only for small and simple project (or for parts of the code, when the best possible optimization is desired). For a project with scale such as this, it would make no sense to try implementing all of the necessary code in assembly. Discarding assembly, we are left with the choice between C and C++ programming language. As C is most widely used for programming microcontrollers, it has better support in libraries, development tools, etc. The objective approach of C++ is also not necessarily needed for this project. It might make a few things easier, but in more cases, it would bring only problems in the form of C to C++ binding. Another possibility may be the usage of FreeRTOS. As its name suggests, it is free RTOS, which can be used on ARM® microcontrollers. One of the main reasons to use FreeRTOS is the implementation of threading. However, FreeRTOS is more suitable for faster microprocessors such as ARM® higher series processors. Fortunately, we do not need to use threads, and it is possible to implement everything without the use of FreeRTOS. In conclusion, the best approach will be to use the C programming language for firmware implementation.

There are quite a few supported IDEs (or better said whole toolchains including IDE, compiler, linker, programmer, etc.) for STM32 family of ARM Cortex-M based microcontrollers 2.15. Paid IDEs such as IAR-EWARM or Keil provides better debuggers and run-time analysis tools. However, their free versions are always limited in some ways. The most common limitation is limit to maximum program size. Some of them also does not support other operating systems than Windows. So rather than to run into program size limit, it might be better to use some of the free, supported IDEs. We can not be

wrong with the choice of SW4STM32 (System Workbench for STM32). It is a free IDE developed by ac6 with direct support from ST Microelectronics. Thanks to being on based Eclipse IDE it also supports all Eclipse plugins. Support for all most widely used operating systems (Windows, Linux, OS X) is a nice additional feature. As it is developed with support directly from ST Microelectronics comprehensive support for STM32 based microcontrollers is present. Support for ST-Link debugging is just another critical feature (for code debug). For compilation is used GCC C/C++ compiler. The used compiler can be of course changed, but there is no need for that.

Figure 2.15. Supported IDEs for STM32 based microcontrollers development [1]

From other development tools, one handy tool must be mentioned, when working with STM32 based microcontrollers. CubeMX is configuration tool developed by ST Microelectronics. Clocks, peripherals, and GPIOs can be configured in an obvious manner using this tool. It also provides simple power consumption computation. The most useful feature is then the generation of code to project of the most used IDEs. Desired libraries (HAL or LL) are also automatically linked to the generated project. So this tool can be used for generating all the necessary configuration code, which can save a lot of time.

2.5.2 PC application development

For PC application development, programming language and framework needs to be chosen. The most constraining requirement is multi-platform support and tools for GUI development. Existing library for low-level work with USB devices would also be a preferred option. All commonly used programming languages provides multi-platform support to some extent.

These days very popular option for GUI multi-platform applications is Qt. It is a free widget toolkit using C++ as its main programming language. Free IDE Qt Creator is distributed with Qt toolkit an also has multi-platform support. Python can be used for Qt development too, as there are existing python bindings (PySide2). However for this particular problematics is better to use standard C++, because of low level operations will be commonly needed. For compilation under Linux GCC compiler distributed with OS can be used. The same way can be used MinGW (substitute for GCC) for Windows compilation (other compilers such as MSVC are also supported). Only disadvantages can be the need to do separate executables for each supported operating system. However, as it has a very wide user base and also meets all the requirements, it was decided to use Qt with C++ for PC application development.

[1] `https://www.st.com/en/development-tools/stm32-ides.html`

Chapter 3
Camera realization

This chapter describes camera realization in detail. Hardware parameters and development workflow are mentioned to help reproduce the final product. One of the main goals of camera realization is to make parts of the product easily replaceable, as it might be needed in case of HW failure.

For the sake of replaceability development board from ST Microelectronics is used rather than a custom board. Development board STM32H743 from Nucleo-144 lineup was chosen. From the main parameters, we can name the following:

- ARM® Cortex®-M7 core at 400 MHz
- 2 MB of Flash memory
- 1 MB of SRAM
- Ethernet and USB connectivity

Another critical property is standard support for I^2C and DCMI interfaces. It also features fast enough processor for all of our needs (even for possible onboard image processing). The only concern comes from a combination of 1 MB internal SRAM and the presence of only full speed USB physical layer on board. Internal SRAM consists of different parts with different purposes (described in section 4.1). So it might be better to use just the largest part of SRAM (512 kB) for image storage. However, that is not enough space for a single image from bigger sensors (which are used in this work). This problem can be resolved with fast enough data transfer to PC (real-time image transfer). Unfortunately, full speed USB has a maximum theoretical speed of 1.5 MB/s. For the real-world application, we have to count with speed maximally around 1 MB/s. Such a speed is usable but is far from the ideal case for image data transfer. A possible solution for slow communication exists in the form of external high speed USB PHY. Connection of external USB PHY is supported using the ULPI interface. As we have a microprocessor in 144 pin package, there are enough pins for I^2C, DCMI and ULPI connection. While using high speed USB maximum theoretical speed is 60 MB/s. Reachable speed will be probably more around a few MB/s, but that should be sufficient for this application.

3.1 Development workflow

In this chapter development workflow of the whole project is described. Crucial breakpoints are defined to understand the sequence of development steps better. This chapter contains useful tips and hints usable for development of a similar project. Other parts of the thesis are focused on describing the final product.

At first, it is important to analyze the problem and define requirements. With defined requirements, suitable tools, technologies, or hardware can be chosen, and development started. The analysis itself deserves its dedicated chapter and can be found at the begging of this thesis in chapter 2.

3.1.1 The first stage of development

At the first stages of development, CMOS sensor board was connected to the Nucleo board just with wires (Appendix C.5). Fortunately, even this type of connection was sufficient for sensor control and image data transfer. It made the early development a bit faster as no additional interfacing board was not needed. The camera is uses USB CDC implementation so communication can work through emulated serial ports. Basic functionalities were validated using simple Matlab scripts for communication with PC. It proves as one of the best options for early development, as it provides a simple interface for a serial port. A free alternative that might be suited as well is using python. For more simple data, it might be better to use some terminal application (e.g. PuTTY). However, using Matlab scripts, we can easily visualize or save image data for validation. At this point, the camera could only capture an image, transfer it using DMA to internal RAM, and then send it using full speed USB. Simple Matlab script for connecting to VCP and capturing one image can be written as:

```
clc
clear all

width = 752; %image width
height = 480; %image height

delete(instrfindall); %removes all serial port objects from memory
instrfind; %finds all non hidden serial devices

s = serial('/dev/ttyACM1'); %get reference to serial port on /dev/ttyACM1
set(s,'BaudRate',115200); %set ports baud rate

len = width*height; %set expected receive data length
s.Timeout = 3; %set ports timeout
s.InputBufferSize = len; %set ports input buffer size
fopen(s); %open port

%evoke image transfer

RAW=fread(s, [width,height]); %receive data
imshow(RAW'./256); %show received image
```

With the addition of more settings capabilities to FW, it was more convenient to start developing a base of PC application. Application using Qt framework gives us better debugging options than Matlab scripts. Also, it can be a lot more versatile, which is crucial for following the development of sensors functions. The serial port implementation in Qt was still used as a communication interface. The camera should support multiple CMOS sensors in its final form, and it would be inconvenient always to flash different FW to MCU with sensor change. Universal programming interface 4.2 for CMOS sensors was designed and developed at this stage. However, it was only tested with MT9V034 sensor. To successfully capture and send an image, it had to fit in internal RAM. Sending offline data (first capture then send the whole image) is not limited by communication speed. Continuous online image data transfer in its full meaning is not possible with FS USB.

In conclusion, at the end of the first stage of development following was finished:

- Working MT9V034 connected with wires
- Main core of FW
- Basic skeleton of PC application
- Offline image data transfer
- Communication with PC using FS USB (VCP)

3.1.2 The second stage of development

The capability of using multiple CMOS sensors enforces the development of interfacing board between Nucleo kit and CMOS sensor board. It can be useful even to save time. Connecting sensor with wires is a faster option if it is needed to connect the sensor just once. However for numerous sensor swaps (which are needed in development), it is more time efficient to spend more time of interfacing board, that saves precious time every connection/disconnection. The second thing forcing development of interfacing board is the need for faster communication. To capture and send images with other presumably supported sensors at full resolution online data transfer to PC is needed. Moreover, that is simply because full resolution images from these sensors will not fit internal RAM. To use faster communication in form HS USB external USB PHY is needed. As interfacing board is already required, the PHY can be placed here.

For the prototype of the interfing board was used 90x70 mm solder breadboard. As HS USB PHY was used USB3320 from Microchip. Unfortunately, this PHY is manufactured only in a QFN32 package. It is not possible to solder QFN32 directly on solder breadboard with 2.5 mm raster. So QFN32 adaptor was used and then soldered to breadboard. Additionally was needed just 1.8 V voltage regulator, USB connector and few capacitors and resistors (full documentation is provided for final interfacing board 3.2). No external power supply was used for the prototype of the interfacing board. Both CMOS sensor and USB PHY were powered from Nucleos 3.3 and 5 V outputs. If we count pins just from CMOS sensor header and USB PHY, we get to number 62. As the majority of these pins is used, it means quite a lot of connections needs to be routed. To do so without beforehand plan might not end well. Avoiding unnecessary crossing of wires is crucial as some of the signals are running at frequencies about few hundreds MHz. Pseudo PCB was designed 3.1 and used as a pattern for routing. It must be noted that it is not real PCB design and it served just as a pattern for routing. That might explain a few things, just as the USB3220 represented by headers of QFN32 adaptor. Also three 100 nF capacitors are not visible on this design, as they are soldered directly on the adaptor (from below). Photo documentation for finished prototype of interfacing board can be found in Appendix B C.6 C.7. Please refer to 3.2 for final PCB design.

Even with pseudo PCB design used as a pattern for routing and no crossing wires, there is still one issue that was probably caused by not optimal routing. While using faster px clocks (more than 25 MHz), image data sometimes contained artifacts. Example of such artifacts is shown in figure 3.2. This problem could be easily countered by lowering px clocks to 20 MHz. Nearly no performance was lost, as the system was mostly bottlenecked by USB communication at this point.

From the FW side, most of the work was done in this part of development. Support for two more CMOS sensor was added. Interface using queues for CMOS parameters setting and parameters sending was created as well (more in section 4.4.4). However, the most important update was the introduction of HS USB support, that was enabled by external HS USB PHY on the interfacing board. The capability of using onboard FS USB was

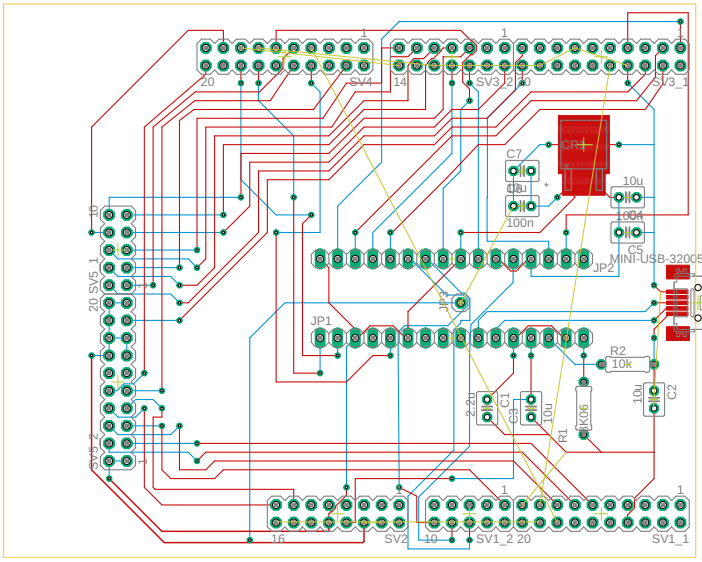


Figure 3.1. Design of handwired pseudo PCB for interfacing board

Figure 3.2. Image acquired from the camera: artifacts showing with higher px clock using the prototype of interfacing board

preserved. HS USB can be used by connecting via a mini USB port on interfacing board and FS USB using a micro USB port on Nucleo kit. New representation enabling working with images bigger than internal RAM was needed because faster USB communication enabled online image data transfer to PC. A form of linked list implementation was chosen for this task 4.4.3.

All of the core functionalities were implemented in PC application in this part of development. As core functionalities finished in this development part can be counted following:

- Dynamic loading of connected sensor data
- Sensor setting through GUI
- Support for both monochrome and RGB image
- Support for images that do not fit in internal RAM
- Registers read/write
- Image saving

With the introduction of HS USB support, current implementation utilizing serial port started to limit performance. As there were also issues with a serial port on Windows, it was decided to use a different approach. New implementation of communication classes using libusb was developed. With this new implementation, communication performance noticeably improved. The whole chapter is dedicated to USB communication implementation 5, as much effort was put in this part of work.

In summary, the following was finished at the end of the second stage of development:

- Prototype of interfacing board with HS USB PHY
- Support for 3 CMOS sensors
- Majority of FW finished
- Core functionalities of PC application
- Working HS USB communication using libusb

3.1.3 The final stage of development

Design of PCB for interfacing board can mark the final stage of development. PCB was designed, sent for manufacturing and after arrival, it was assembled and finally proven functional. The performance was better than with prototype, as we got rid off the artifacts with faster px clocks. For detailed design informations see following section 3.2.

On the FW and PC application side mostly optimizations and other improvements were implemented. Most notably, USB communication was furthermore optimized for better performance. To PC application was added linear mode usage of CMOS sensor and sensors settings saving.

3.2 Interfacing board

A connection is done through ST Zio connectors (female headers on Nucleo board). Another possibility would be to use ST morpho connectors. These allow full access to all STM32 I/O, unlike ST Zio connectors with limited access. Unfortunately, no headers are soldered on Nucleo-144 boards on ST morpho connectors places. However, St Zio connectors still provide enough connections to the microprocessor. Moreover, while using St Zio connectors, no need Nucleo board modification is present. Connection pinout for both prototype and final interfacing board is shown in figure 3.5.

Just a few changes from prototype were made in the final design of interfacing board. Most changes were made in power delivery. On prototype of interfacing board, the whole camera was powered from ST-Link on Nucleo board. Board then provides 3.3 and 5 V voltages, so just one 1.8 V regulator was needed for supplying USB PHY. This option was preserved even in final interfacing board design. However, another option in the form of an external DC power supply was added. DC power supply is connected using a standard 2 mm barrel jack connector, and voltage ranges are 5-7 V. In case of using the DC power supply option, both USB PHY and CMOS sensor are powered from this source and not through Nucleo kit. Meaning an additional 3.3 V voltage regulator is needed (alongside 1.8 V regulator). In both cases, linear low-drop voltage regulators[3] are used. CMOS sensor is powered by 3.3 V, and USB PHY needs both 3.3 and 1.8 V for proper function.

For powering Nucleo board, there are four options, including power through St-link. So 3 options for externally powering Nucleo board are available, as seen in table 3.1. The best option would be to use E5V input with 5 V power supply. Unfortunately, this input is only on ST morpho connector and is not available on used ST zio connectors. The +3.3 V input is not ideal either, as we would like to use the same power delivery circuit while

using both external power supply and ST-link power. With 3.3 V as input to the LF33 voltage regulator, the output does not always have to be 3.3 V as dropout voltage is rated up to 0.7 V. In the end, only one option is left in the form of VIN input with a declared range of 7-12 V. However, using more than 7 V as input to LF33 might not be ideal either. Schematics 3.3 shows how power delivery circuits look on used Nucleo boards.

Input name	Voltage range
VIN	7-12 V
E5V	4.75-5.25 V
+3.3V	3-3.6 V

Table 3.1. Nucleo board external power options overview[4]

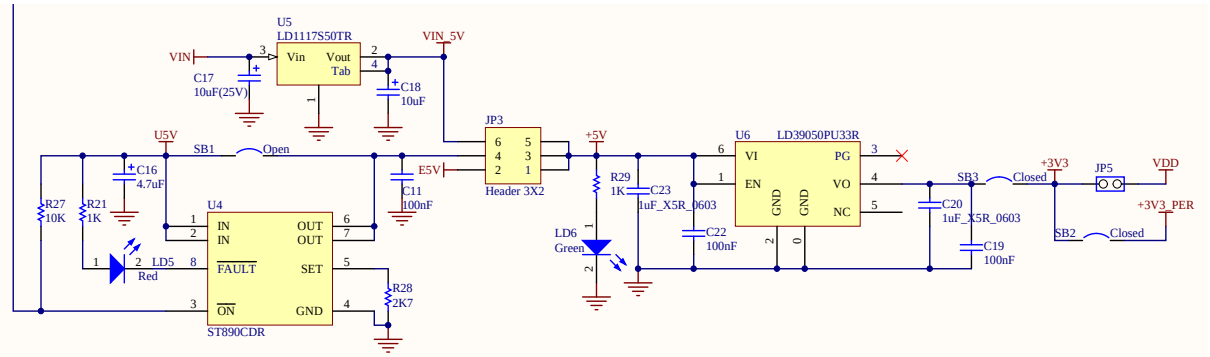


Figure 3.3. Power circuits on Nucleo board (taken from[4])

From this schematics and datasheet of used regulator LD1117 [5] we see why VIN input lower range is rated at 7 V. LD1117 has dropout voltage rated up to over 1 V, so at least over 6 V should be on the input. To make sure there is always 5 V on the output of LD1117, 7 V rating was used as minimal VIN input. However these 5 V are used just as output to boards connectors and are not actually not needed for the proper function of the board. Necessary is just 3.3 V output from the following voltage regulator. For that to be true, 5 V is not necessarily required on the input of the 3.3 V regulator. With that, we can conclude, that Nucleo board will just fine (while not using 5 V output) with 6 V (or even 5 V) on VIN input.

To ensure correct polarity of input voltage, a diode is placed directly after the DC connector. There is also LED indicating power on. Selecting power source (ST-link or external) is done by a jumper with instruction printed on the interfacing board. It is essential to change the jumper position on the Nucleo board as well when changing the power supply. Jumpers configuration can be seen in figure 3.4.

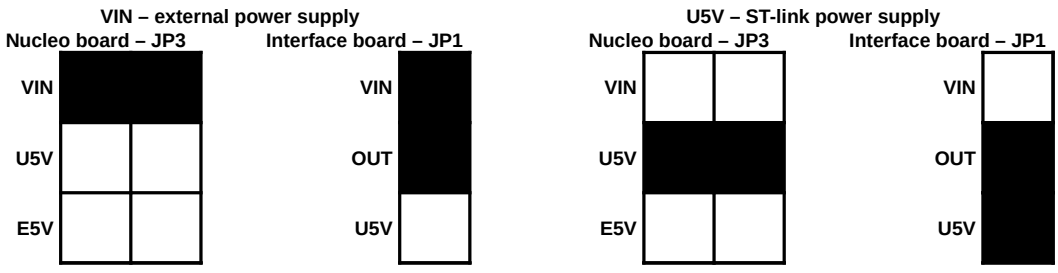


Figure 3.4. Jumpers configuration for power supply selection

Additionally, four user programmable LEDs were added. They are connected on GPIOs PF15, PG14, PE8, PE7 (printed on interfacing board as well). These are all the relevant changes from the prototype. The PCB was designed using KiCad software. Detailed documentation for final version can be found in appendix B and photos C.8, C.9, C.10, C.11 or attached CD.

3.3 Supported Image sensors

The flexibility of the developed camera is created by interchangeable CMOS image sensors. For some teaching labs, it would be better to use a small global shutter image sensor. On the contrary for simulation of a linear image sensor is more suitable bigger sensor without concerns with rolling shutter. Camera firmware is implemented with the thought of adding more support image sensors in the future. Adding an image sensor requires only writing a few necessary functions and recompiling the project (more in chapter 4.5). That is the case if the sensor is interfaced using a standardized board used in CTU FEE laboratory of videometry 3.6.

On figure 3.6 top view of connector (female headers facing down to interfacing board) is presented. Power delivery is marked in red (3V) and black (GND). In yellow color we can see signals from sensor to DCMI interface (for detailed description see chapter 2.4.2). Signals D0-7 are for parallel image data transfer. Three other signals then serve for synchronization:

- VS - vertical synchronization
- HS - horizontal synchronization
- PX - pixel clock

In blue color are marked other CMOS sensor signals. Their active levels may vary for different sensors, but their utilization is generally the same.

- CLK - Clock - master clock signal for sensor (supported ranges may vary)
- STB - Standby - shutting sensor down for power saving
- EX - Exposure (or Trigger) - stars exposure in snapshot or slave mode (or similar manual modes)
- RE - Reset - asynchronous hard reset of a sensor to default settings
- LD - Led out (or Strobe) - indicates ongoing exposure or end of exposure (may widely vary for different modes and sensors)

Lastly, in green color are marked I^{2}C signals SDA and SCL used for sensor setting.

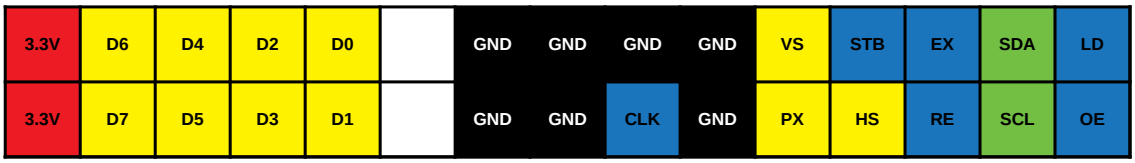


Figure 3.6. CMOS interfacing board pinout (Top view)

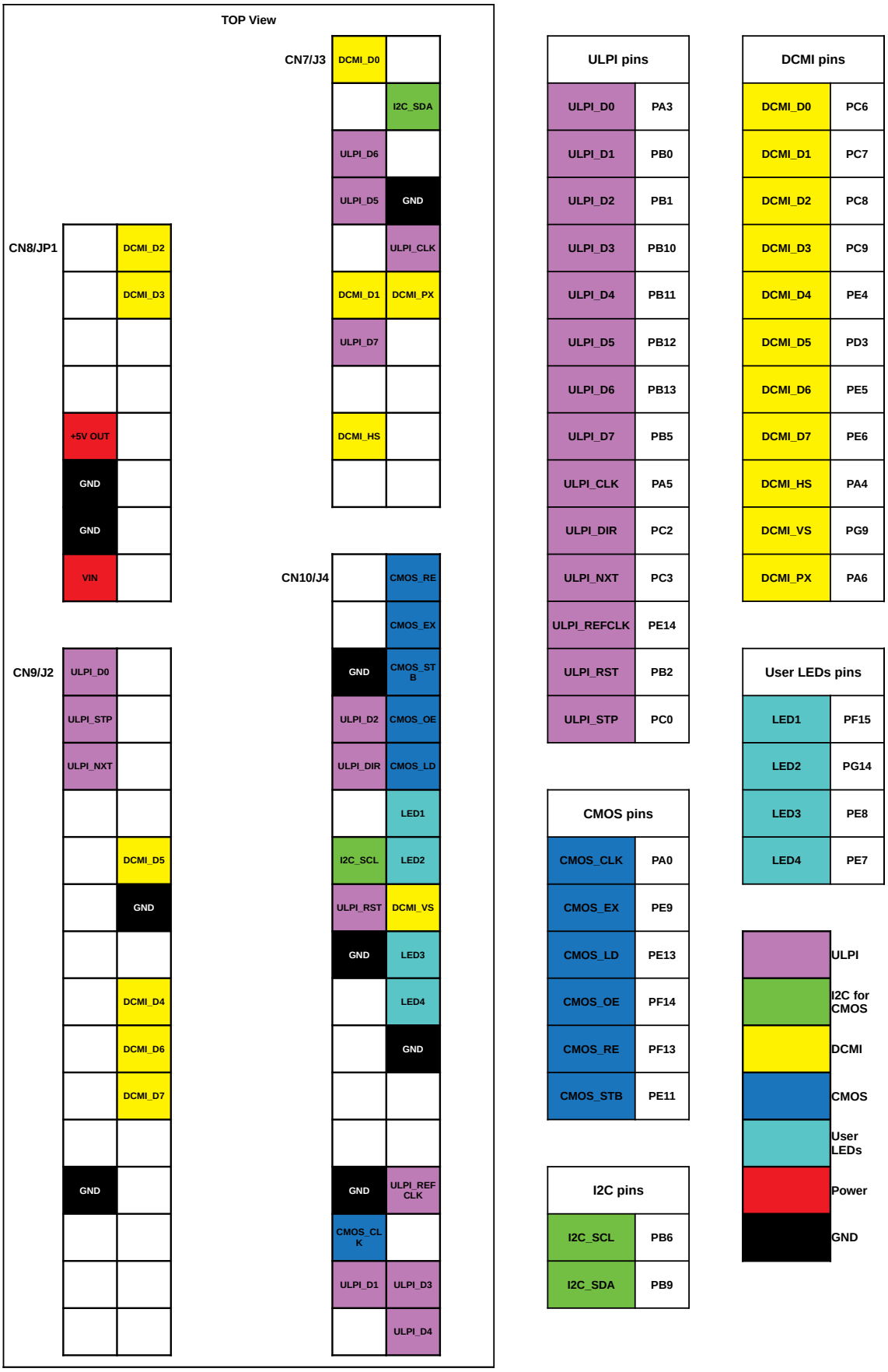


Figure 3.5. Pinout of interfacing board

3.3.1 Image sensor MT9V034

The smallest supported CMOS sensor. Also, it is the only one with a global shutter. At least one sensor with global shutter was required to be able to show differences between shutter types in laboratories. From standard features windowing, a setting of exposure time and gain settings are supported. It also features pixel binning and I^2C communication interface. Apart from standard features, this sensor also supports additional features as AEC (automatic exposure control), AGC (automatic gain control) and HDR (high dynamic range), that other supported sensors do not. Main sensor features are presented in table 3.2.

Optical format	1/3-inch
Active imager size	4.51 mm (H) x 2.88 mm (V)
Active pixels	752 (H) x 480 (V)
Pixel size	6.0 x 6.0 μm
Color filter array	Monochrome
Shutter type	Global shutter
ADC resolution	10-bit
Responsivity	4.8V/lux-sec
Dynamic range	>55 dB linear >100 dB in HDR mode
Features	AEC, AGC, HDR Px binning, Row/Column flip, ROI

Table 3.2. MT9V034 parameters [6]

3.3.2 Image sensor MT9M001

MT9M001 is a bigger sensor with fewer features than MT9V034. It was mainly chosen to be used for simulated linear mode, due to its bigger size. From additional features windowing, a setting of exposure time and gain and pixel skip are present. For sensor setting required I^2C interface is present. Interfacing board used for MT9M001 had a small problem with missing pull-down resistor on the standby signal. That resulted in the undefined state for that pin, which caused occasional sensors power downs with low illumination. Main sensor features are presented in table 3.3.

Optical format	1/2-inch
Active imager size	6.66 mm (H) x 5.32 mm (V)
Active pixels	1280 (H) x 1024 (V)
Pixel size	5.2 x 5.2 μm
Color filter array	Monochrome
Shutter type	Rolling shutter
ADC resolution	10-bit
Responsivity	2.1V/lux-sec
Dynamic range	68.2 dB
Features	Px skip, Row/Column flip, ROI

Table 3.3. MT9M001 parameters [7]

3.3.3 Image sensor MT9T001

The latest supported sensor is very similar to previous MT9M001. It has just higher resolution and Bayer RGB mask. This 3-megapixel sensor has the same capabilities as MT9M001 (windowing, exposure time, and gain setting, pixel skip, I^2C). Additionally, it features pixel binning as the second option to image size reduction. As a specialty, it also features GRR (Global reset release). When combined with a controllable mechanical shutter, it can then simulate global shutter. Main sensor features are presented in table 3.4.

Optical format	1/2-inch
Active imager size	6.55 mm (H) x 4.92 mm (V)
Active pixels	2048 (H) x 1536 (V)
Pixel size	3.2 x 3.2 μm
Color filter array	RGB Bayer mask
Shutter type	Rolling shutter
ADC resolution	10-bit
Responsivity	>1V/lux-sec
Dynamic range	61 dB
Features	GRR, Px skip, Px binning, ROI

Table 3.4. MT9T001 parameters [8]

Chapter 4
Camera Firmware

Developed camera FW is one of the crucial parts of this thesis. It was designed to be easily expanded for support of another sensor. In this chapter, design choices and implementation details of FW are discussed.

4.1 Memory usage

Correct use of available memory is an essential part of any FW design. It gets more complicated with high-performance microcontroller featuring different parts of RAM with different functionality. Used microcontroller H743 features 2MB of flash memory used for storing an executable program (FW). More interestingly four types of embedded SRAM dedicated for a different purpose are featured:

- 864 kB of system SRAM
- 128 kB of DTCM RAM
- 64 kB of ITCM RAM
- 4 kB of backup RAM

System SRAM is further divided to 3 different power domains and 5 blocks as seen in table 4.1.

Domain	block	start	end
D1	SRAM	0x24000000	0x2407FFFF
D2	SRAM1	0x30000000	0x3001FFFF
D2	SRAM2	0x30020000	0x3003FFFF
D2	SRAM3	0x30040000	0x3004FFFF
D4	SRAM4	0x38000000	0x3800FFFF

Table 4.1. System SRAM block in H743

Division to more power domains might be useful, as not all of them needs to be powered at the same time. In some cases, we might want to turn on standby mode on D1 and use, for example, D3 to retain some smaller amount of data. However, this is not necessary for this particular application. In fact, only SRAM in D1 is used. Whole block SRAM is dedicated for image data, transferred using DCMI through DMA. It might be possible to use other parts for the same purpose, but it would be a bit more complicated for implementation and would not bring that much of an advantage. Even with just using SRAM block with size 512 kB, more than half of system SRAM is utilized for image data.

More interesting might be the use of the two following parts of SRAM. DTCM and ITCM stand for data/instruction tightly coupled memory. It means that the core can access these memories with minimal delay times. DTCM with a size of 128 kB is big enough to fit all non-static program data, including stack and heap. So there is no reason to use other SRAM parts for data, except for image data stored in SRAM. On the other

hand, ITCM can be used for time-critical routines. In this case, it is used for DCMI and DMA manipulation for image capture. As not all functions are stored in ITCM, separate ITCM section needs to be declared in the linker script. This section then also has to be copied from flash to ITCM in the startup script. Functions to store in ITCM must also be declared with specific identifiers. According to [9] using ITCM and DTC part of RAM leads to optimal performance. Declaration of ITCM section in a linker script can look like the following:

```
  _siitcm = LOADADDR(._itcm);
  ._itcm :
  {
    . = ALIGN(4);
    _sitcm = .;
    *(._itcm)
    *(._itcm*)
    . = ALIGN(4);
    _eitcm = .;
  } >ITCMRAM AT> FLASH
```

Part of start up scrip to copy form flash to ITCM can than be written in assembly as:

```
 movs  r1, #0
 b  LoopCopyDataInitItcm
/* load data to itcm */
CopyDataInitItcm:
  ldr  r3, =_siitcm
  ldr  r3, [r3, r1]
  str  r3, [r0, r1]
  adds  r1, r1, #4

LoopCopyDataInitItcm:
  ldr  r0, =_sitcm
  ldr  r3, =_eitcm
  adds  r2, r0, r1
  cmp  r2, r3
  bcc  CopyDataInitItcm
```

And the last step is to declare the function with section attribute as following:

```
void fuctionName(void)  __attribute__((section ("_itcm")));
```

The last backup part of RAM is most of the time used while using batteries as power supply. It is possible to use this part of RAM to retain data during low power modes. That is not necessary for this application. Thus backup RAM is not used.

4.2 CMOS sensor representation

Internal CMOS sensor representation is what enables multiple sensors support and simple expandability of another sensor. It is represented by one data structure, containing all information about sensor alongside functions for sensor control. With this design object-oriented programming is simulated. Entire structure is shown in figure 4.1.

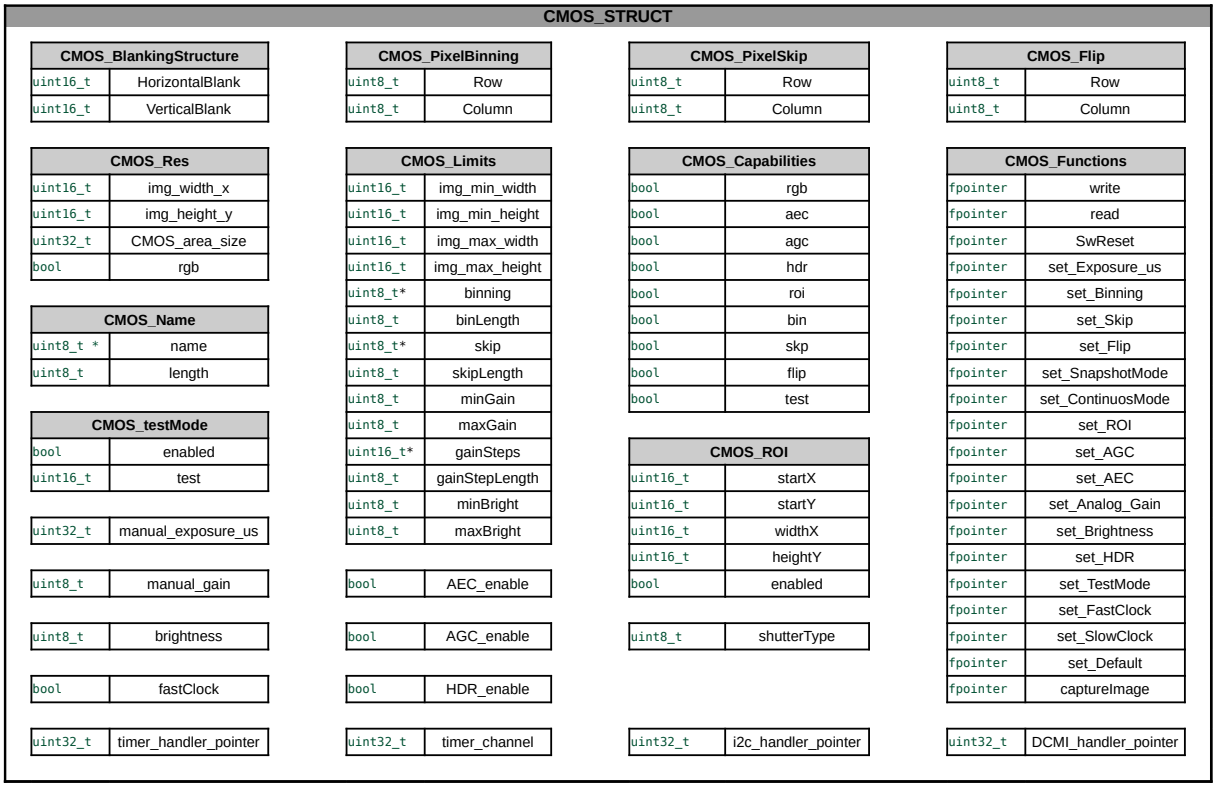


Figure 4.1. Internal FW CMOS sensor representation

Individual structures and parameters included in CMOS structure and their purpose are the following:

1. CMOS_BlankingStructure
   The internal representation of the current blanking parameters of the connected CMOS sensor. These parameters are never sent to a PC application, as a direct setting through PC application is not supported.
2. CMOS_PixelBinning
   A structure representing the current state of pixel binning. It is not relevant when the sensor does not support pixel binning.
3. CMOS_PixelSkip
   A structure representing the current state of pixel skip. It is not relevant when the sensor does not support pixel skip.
4. CMOS_flip
   A structure representing the current state of image flip. It is not relevant when the sensor does not support image flip.
5. CMOS_Res
   Includes the current image resolution produced by the sensor. That means it includes image reduction done by ROI, binning, or skip. It includes the size of the image and information if the image is monochrome or RGB.
6. CMOS_Name
   A simple structure containing the sensors name with its length. Internally it has no meaning - sensors name is just sent to PC application.
7. CMOS_testMode A structure representing the current state of test mode. It is not relevant when the sensor does not support test mode.

8. CMOS_ROI

   Contains information about the current state of windowing - meaning image start and its width and height. The difference from CMOS_Res is that it contains raw image sizes without pixel binning or skip.

9. CMOS_Limits

   A structure containing the limits for available settings. It is mainly used for sending limit parameters to a PC application, so values out of bounds are not even presented. Most of them are pretty straightforward, and no explanation is need. However, few parameters use an array with array length variable. More simple ones are binning and skip, that contains a list of available settings. The most complex is the gain limits. Individually saved are minimal and maximal gain values. An array then represents steps between these values. However, to reduce the amount of unnecessary data, it is not simple lists of available values. The array has structure, where there are always three values for a single gain area (gain interval where gain increment is the same). The first from these three values is gain increment in its real form (usually floating point, so it is this value·10000), the second is an increment in internal sensors register settings and the last is the maximal limit for this particular interval. All the available values are computed on the PC application side using this single array.

10. CMOS_Capabilities

    A structure containing the information about sensors capabilities. It is vital, as it says, which function are mandatory to implement for a given sensor.

11. CMOS_Functions

    Contains pointers to the given sensors implementations of CMOS control functions. Always mandatory functions are:

    - write - I^2C write
    - read - I^2C read
    - set_Exposure_us - sets exposure time
    - set_Analog_Gain - sets analog gain
    - set_FastClock - sets px clock or other setting for fast image capture
    - set_SlowClock - sets px clock or other settings to enable online image data transfer while capturing
    - set_Default - sets CMOS sensor to its default settings
    - captureImage - captures Image and transfers it to internal RAM

    If it is mandatory to implement other function is determined by CMOS_Capabilities structure. If the given capability is set to true, it is mandatory to implement the relevant function. By using pointers to function, it is possible to mimic polymorphism capability of object-oriented programming to some extent.

12. Other non-structure parameters

    Most of the parameters not included in any structure are quite simple. They just represent the current state of sensor (e.g. manual_exposure_us, manual_gain, etc.). ShutterType then says what shutter is current sensor using (global or rolling). Remaining parameters are mandatory for correct sensor function. Two of them - timer_handler_pointer and timer_channel defines controls for the timer, that is used for generating px clocks with adjustable frequencies on the go. For I^2C communication handler for given peripheral is needed and a pointer to it referenced by i2c_handler_pointer. The last parameter, DCMI_handler_pointer, is necessary for image capturing.

4.3 Image representation

Suitable image representation is crucial, especially when working with images that will not fit internal RAM. In this case, online data transfer to a PC is required. However, it would still be convenient to have a representation of the whole image in a single structure. Passing just one structure pointer is the cleanest way of handling image passing between different parts of the program (functions).

It was determined that using a type of linked list implementation might be a suitable solution. Mainly because it is quite a simple data structure (meaning faster append and delete operations than over more complex data structures) moreover, in case of an image, it will always be iterated over the whole structure. Just from these two reasons, it seems linked list implementation might be the ideal solution for this particular problem. Basic linked list representation can be seen in figure 4.2. In its most simple form linked list structure can include just pointer to head node. Every node then contains some data and pointer to the following node (linking nodes together - linked list). Other additional variables, such as tail (pointer to the last node) or a number of nodes, can be added to list structure if necessary.

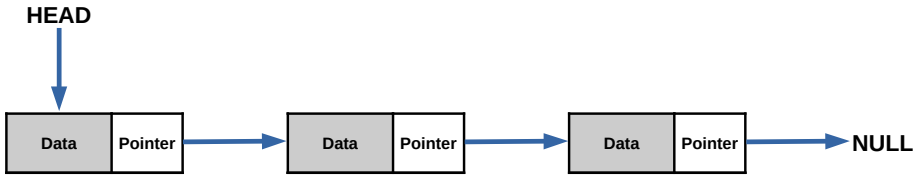


Figure 4.2. Basic representation of a linked list

Every node contains the address where part of the image begins, size of the part, information if the part is ready to send and of course pointer to the next node. ImageArea structure itself contains a pointer to the first node (head), the total number of nodes and pointer to the current node indicating currently processed node by DCMI/DMA. Both used structures for linked list are shown in figure 4.3.

ImagePart	
partBegin	4 bytes
part_Size	4 bytes
readyToSend	1 byte
next *	4 bytes

ImageArea	
begin *	4 bytes
current *	4 bytes
numOfParts	1 byte

Figure 4.3. Internal image area representation

At the beginning of the program and every time image size changes, imageArea needs to be defined again. It also expects that image size is already aligned to 4 bytes, as DCMI and DMA work only with 4-byte alignment. Process of setting imageArea is shown using flowchart in figure 4.4.

At first, it is decided if will the image fit to a dedicated part of internal RAM. If it does not fit, online transfer of data will be needed. For that CMOS sensor is slowed, so image transfer to RAM is not faster than image sending. Follows free of currently allocated imageArea list, as it is dynamically allocated. Next new lists parameters are computed. As DCMI/DMA works with 4-byte alignment, image size is first divided by 4 and set as partSize. Also, maximal size for single DMA transfer is 0xffff times 4 bytes, so partSize is halved until it is less than this value. At the same time, the number of parts is

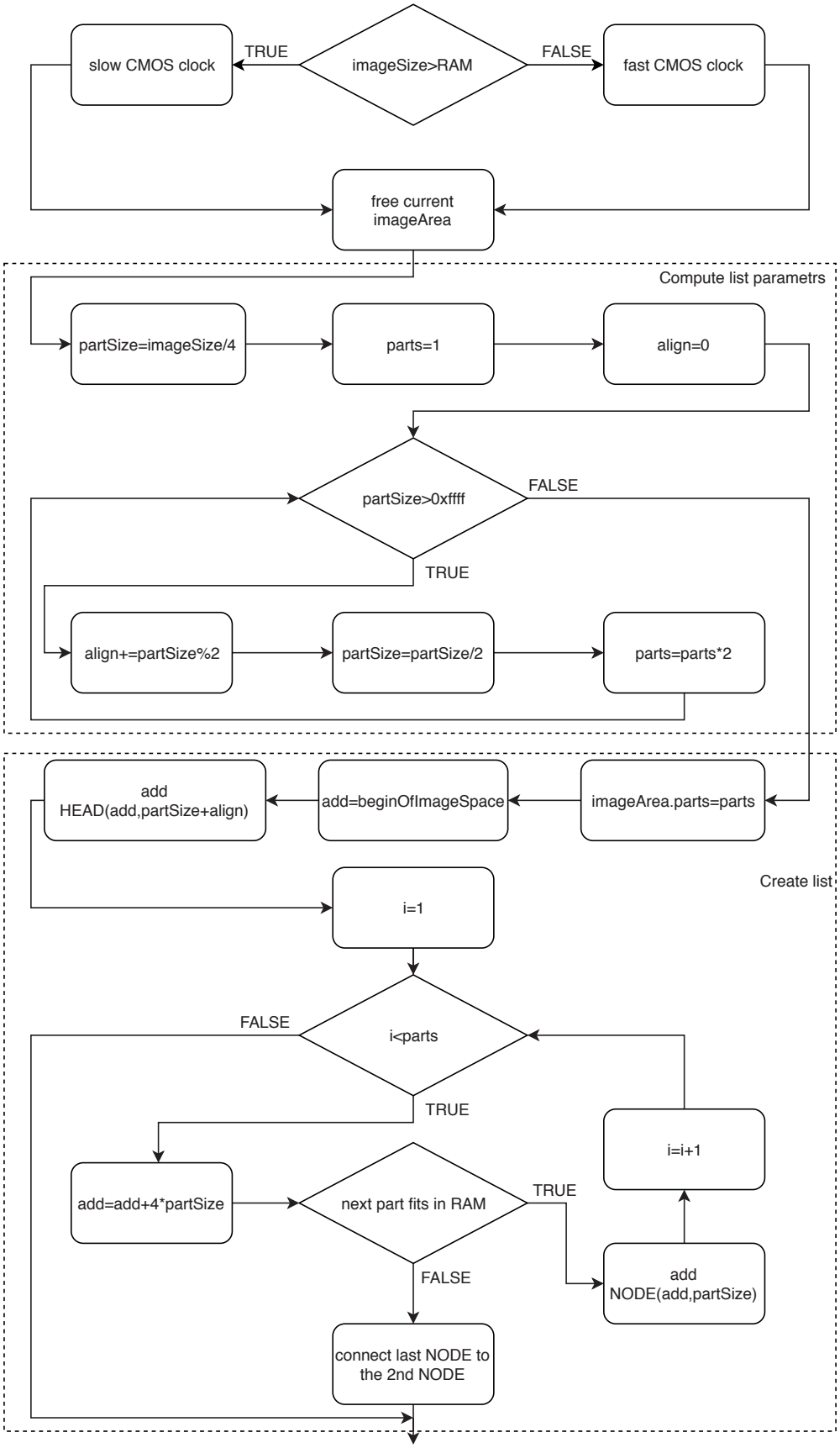


Figure 4.4. Flowchart of setting imageArea

counted by multiplying by 2. When rounding is needed after division by 2, align variable is incremented, so no bytes are lost in the process. With list parameters computed, it can be created. At first head of the list is added with computed part size + align from division, starting at the begging of dedicated part internal RAM. If all parts fit dedicated part of internal RAM, they are all just added with incremented address and computed part size. If the next part will not fit RAM, the last added part next pointer is pointed to the 2nd item in the list. The 2nd is selected simply because the head can have different size (alignment) and all other parts share the same size. With pointing to a node already in the list, the cycle is created (meaning infinity length in linked list point of view). However, the number of parts are saved as one of the parameters of the list and are used as limit while iterating over the list. By cycling the list this way unnecessary operation and waste of RAM is prevented because there is no need for creating new nodes with duplicate parameters.

4.4 Firmware functionality

In this chapter, firmware functionalities are described. Whole can FW can be seen as individual modules that interact with the main loop. A simplified model of modules and their interactions is shown in figure 4.5.

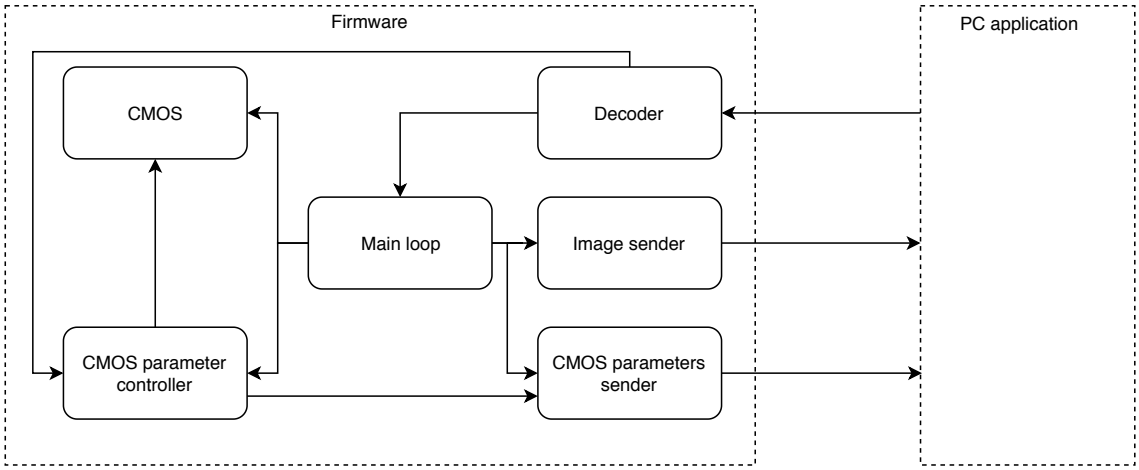


Figure 4.5. Schematics of FW modules and their interactions

- **Main loop**
  The main loop runs in endless cycles and calls all other modules except decoder. Its states and more are described in detail in section 4.4.2.
- **Decoder**
  Decoder module is called whenever new data from PC application are received. After it decodes received data, it can change the main loop state. The second thing it can do is forward parameters to CMOS parameters controller, which will be set next time controller is called from the main loop.
- **CMOS**
  CMOS sensor module is called from the main loop to capture images. CMOS parameter controller can then can also call CMOS sensor module to change sensor settings.

- **CMOS parameter controller**
  As already said, the CMOS parameter controller gets information about what to set directly from the decoder. However, the setting is executed only after the call from the main loop.
- **Image sender**
  This module sends provided image using USB peripheral after the call from the main loop.
- **CMOS parameters sender**
  Purpose of this module is to inform PC application about the current state of CMOS sensor parameters and their changes. CMOS parameter controller can modify what parameters will be sent because after the change of CMOS parameter, the new parameter is sent to the PC application.

Available commands for decoder module and also format of other used messages is described in section 5.4.

4.4.1 Initialization

Before going to programs main loop, initialization of necessary parts is performed. The first thing to initialize is system clocks, following by used peripherals. Everything is going in the usual fashion, except for USB peripheral initialization. A timer is used for generating 24 MHz clocks for used external PHY, so this particular timer needs to be initialized and started before USB initialization. With all the necessary microcontroller parts initialized, CMOS sensor initialization can be performed. Whole CMOS identification and initialization process is shown in figure 4.6 using flowchart.

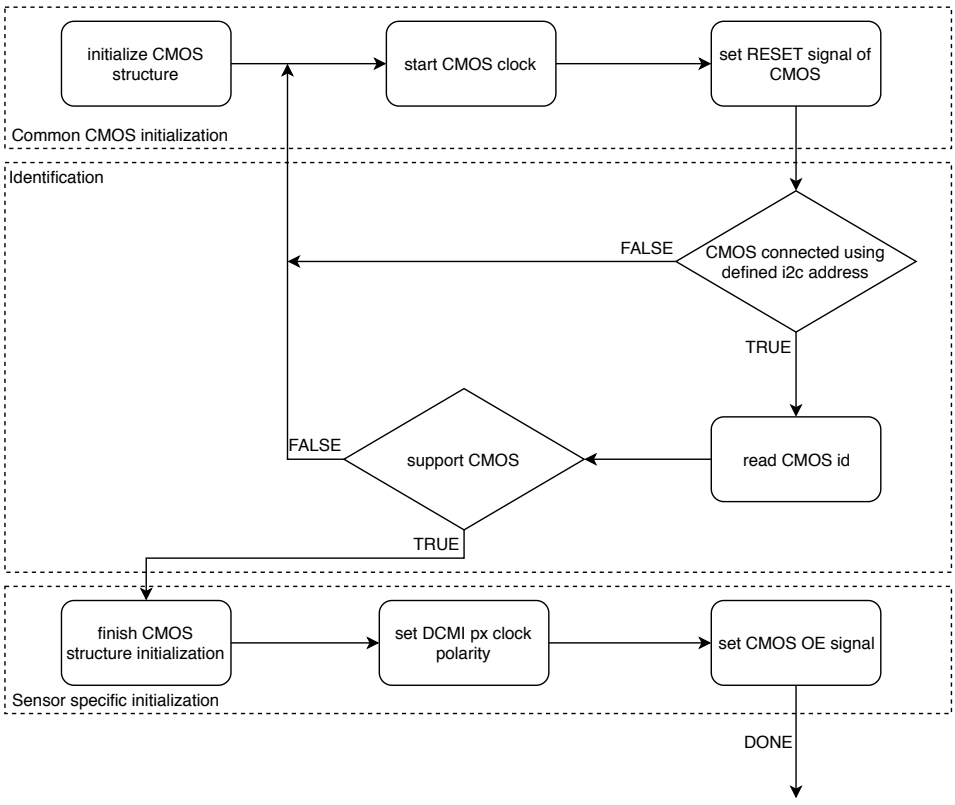


Figure 4.6. CMOS identification and initialization process flowchart

The first part of CMOS initialization is the same for all supported sensors. To CMOS structure are passed pointers to handlers for required peripherals (timer, DMI, I^2C). Next timer generating clocks for the sensor is started so the sensor can start communicating. For that, it is also necessary to set RESET pin of the sensor, so CMOS is not in standby/reset state. If there is a CMOS sensor connected, it should be able to communicate with microcontroller using I^2C. FW includes saved list of possible I^2C address to search on. In the identification phase, these addresses are searched by trying to read sensors ID register (most of the time 0x0). If the read is successful, it means the sensor is conned. When no sensor is found on available addresses initialization, begin anew. With acquired sensor ID, it is determined if the sensor is supported. For supported sensor is then initialization finalized. CMOS structure is filled with sensor-specific information, DCMI initialization is finalized, and last is setting OE pin of sensor. At this point, the CMOS sensor should be ready for usage. If no sensor or unsupported sensor is found user LED4 is turned on to let, the user know of the given issue.

4.4.2 Main loop

The main loop works as a state machine. To get into another state than do nothing, the camera needs to be conned to PC application first. In a default state is then still done nothing. Only after decoder decodes incoming data from PC application, the state is changed. Following state are possible:

- **Default**
  Nothing is done in this state, only awaiting the change of state.
- **Single image**
  In this state single image is captured and sent to PC application. After the send is finished, the state is changed to default.
- **Continuous capture**
  Images are continuously captured and sent to PC application until the state is changed. After the state change, the currently processed image is always finished before moving to the new state.
- **Info**
  If the send parameters buffer contains any items, they are sent to PC application. The state machine is returned to the previous state after going through the whole buffer. That is mainly because when we need to set and then send parameters while continuously capturing images.
- **Set**
  Work similarly to `info` state, with the difference, set parameters buffer is read. It contains parameters to set for CMOS sensor. Parameters that are set are also immediately sent back to PC application to confirm their new value. After finishing this agenda, the state machine is returned to the previous state.
- **Registers**
  The state somewhere between info and set state. It can read CMOS sensors register and send its value to a PC application, or it can set register value. So both send and set operations can be performed, but not over CMOS structure like info and set states. Return to the previous state is also done after finishing send or write operation.

4.4.3 Image capture and sending

As already mentioned capture of the image is done using DCMI peripheral 2.4.2. Transfer of data from DCMI peripheral to SRAM is then done through DMA. By using DMA processor time is saved as data are after DMA peripheral set up transferred without processor assistance. That enables us to send parts of image data, while the image is still captured. It not only makes continuous image capture faster but also enables processing of an image that would not normally fit internal SRAM. DCMI is used with hardware synchronization mode as all supported CMOS sensors provide all necessary synchronization signals. Only monochrome/raw Bayer data format is used for DCMI, as all supported sensors use it. The whole process of image capture and send is shown in figure 4.7.

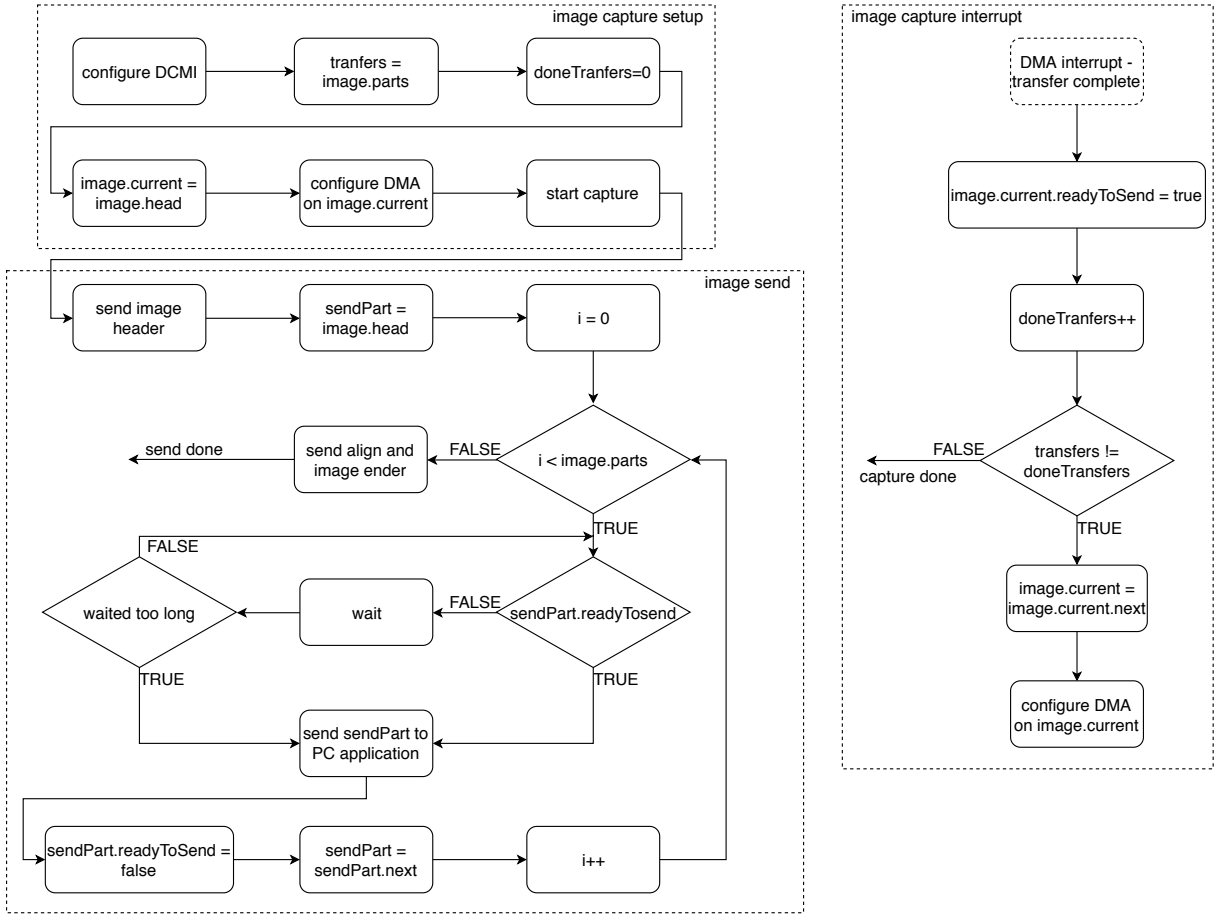


Figure 4.7. Image capture and send to PC application flowchart

The process of capturing an image starts with the setup of the DCMI peripheral. Noteworthy is also setting of image linked list head as currently processed part of the image to the image structure. Afterward, DMA can be configured according to the current part of the image. Capture is started by setting DCMI capture bit. DCMI then wait for VS marking new image start to begin capture.

Immediately after capture start, sending of the image to PC application is also started. Without delay, the image header can be sent. After the header, it is iterate over the image linked list using a saved number of parts. Every image part is marked if it is ready to be sent. If the current part is not ready to be sent, a certain amount of time is waited before forcing the sent of the current part.

Set up of DMA for another part of the image is done in interrupt after the previous part transfer is finished. Transfer of data for the next part can be done simultaneously with sending previous part, thanks to DMA working independently. Transferred image part is also marked as ready to be sent in the interrupt. By using the concept of simultaneously sending to PC application and capturing the image (transferring data to internal SRAM) arbitrary sized images can be processed.

To do simultaneous transfer it is usually needed to slow down the CMOS sensor, so DMA does not overwrite section not yet sent. Slowing down can be done in two different way, and it depends on the sensor, which way is more suitable. One way is to slow down px clocks controlled from FW, which slows down readout of every single pixel (every pixel is valid for an extended period of time). The second way is to increase horizontal blanking time on sensors side. By increasing horizontal blanking, the delay between individual lines readout is increased, effectively slowing down whole image readout.

4.4.4 CMOS parameters setting and sending

Both settings of CMOS parameters and their sending to PC application works similarly. Actions are first added to buffer and then executed at an appropriate time, meaning currently not capturing or sending an image. Buffer for parameter setting is filled from the decoder after incoming data are received. It includes a list of parameters to set and also parameters itself, as they are not stored anywhere else. The maximum length of the buffer is 50 and that should more than enough, as it is about four times more than the number of settable parameters. All the actions from the buffer are then executed after a call from the main loop. Parameters sender need only the buffer of demanded actions, as it reads values directly from CMOS structure. Its maximal length is the same as the one of parameters setter buffer, meaning 50 items. Format of used messages is described in section 5.4.

4.5 Adding support for another CMOS sensor

With the assumption that the CMOS sensor uses the same physical interface as described in section 3.3, adding its support should be quite straightforward. It requires a small modification to only two files and of course adding new files for a given sensor. Library for a given sensor can be created similarly as for three already supported sensor. That means header file defining sensor capabilities, limits, and registers. Source files then contain the implementation of functions for CMOS control. The most important would be initialization function filling CMOS structure with sensors parameters and functions. All always mandatory function to implement would be:

- initialize CMOS structure
- set to default state
- functions for I^2C communication
- image capture
- set exposure time
- set analog gain

If the CMOS sensor maximal resolution image will not fit internal SRAM, it is also mandatory to implement functions for slowing down the sensor to perform online data transfer. Function for faster readout when the image will fit SRAM is then also mandatory. Other functions implementation need depend on defined capabilities. In the case of defined capability, a respective function needs to be implemented as well.

Few modifications need to be done in CMOS header and source file. In header I^2C address of new sensor needs to be added to address list to search sensor on if it is not already present. If it is added, the length of this list needs to be incremented. Modification in CMOS source file is only in CMOS_identify function. New case with sensor ID finalizing initialization needs to add (using the same pattern as already supported sensors). No modifications should be necessary to a PC application, and new CMOS sensor should be supported with just these few changes.

Chapter 5
USB communication in detail

For communication between the camera and the PC application was chosen USB. It is still probably the most common option for data transfer from MCU powered device to PC. Every computer has at least a few USB ports, and USB cables are also very commonly available. However, focus in this chapter is mostly on used implementations of USB, rather than USB itself. Communication protocol specific for the developed camera is also described.

5.1 Firmware side implementation

Before implementation, it must be decided what functionality is expected from the camera, to choose USB role and class. In USB communication host can communicate with the device (slave), while the host always has to initialize the data transfer in both directions. The unit can usually act only as a host or as a device. There is one exception of UBS OTG that allows both roles, depending on the connected device (it is mainly used in phones or tablets, that acts as a device while connected to PC, but acts as host while connected to for example keyboard). However, in this particular case, the camera will always need a connection to the PC. PC will act as host, and thus camera needs to be defined as a device only. USB OTG functionality could be theoretically implemented but is not needed.

With the decision, that the camera will be defined as a USB device, the correct USB device class needs to be chosen. In total, 20 device classes are defined, alongside class for unspecified device class. Defining device class allows the host to load appropriate drivers to use the device automatically. In some specific cases, it might be necessary to use 0xFF vendor-specific device class. It indicates that no standard driver can be used and the vendor-specific driver is needed. However, it is better to avoid own driver development, as driver signing might be a bit unnecessary issue. If the device can be written using existing device class, drivers development can be skipped entirely.

Only two of the defined UBS device classes would make sense for our purpose. The first to come in mind might be 0x0E video class. However this class is primarily designed just for video streaming and is single purposely defined, hence it is not exactly suitable for out camera specific communication protocol. USB communication device class (CDC) is ideal precisely for this purpose. It uses bulk transfers, so high data rates can be achieved. Common USB CDC host drivers act as virtual com port and device can then be easily accessed as a serial port. It leads to the fact that USB CDC defined device acts as a serial device to outside and is expected to have a pipe like connection opened after connection to host (it is up to host to initialize transfers to be ready for receptions). When using USB CDC drivers and connecting to the device using serial port implementation, an application can act to the device as it would to a real serial device (USB routines are performed by the driver). Thanks to this development of host-side communication for USB CDC defined device is very simple. On top of that ST Microelectronics provides USB CDC stack implementation for their products, so device side implementation is not problematic as well.

The final camera device includes implementation of both full speed and high speed USB. If the camera is connected to PC by the micro USB port on Nucleo board, it uses internal PHY that has only full speed capabilities. However mini USB connector on the interfacing board can also be used. In this case, external high speed PHY is used, and the camera is then capable of high speed USB utilization. Type of used USB connection is determined after connecting to the host. This connection is utilized for the rest of the session. Both implementations use the same USB CDC definition with a slight change in descriptors. Descriptors parts used for device identification are defined as:

- FS descriptor
  - PID = 0x1
  - VID = 0x1
  - Manufacturer string = "Vodsedalek_FEE_CTU"
  - Product string = "Meassure Cam FS"
  - Serial number string = "00000000001A"
- HS descriptor
  - PID = 0x1
  - VID = 0x1
  - Manufacturer string = "Vodsedalek_FEE_CTU"
  - Product string = "Meassure Cam HS"
  - Serial number string = "00000000001A"

Bulk transfer with maximal respective bulk size packets is used for data transmit to maximize data rate. For full speed USB, maximal bulk size is 64 bytes, and for high speed USB, it is 512 bytes. Also, only a single interface can be used at a time, so it is not possible to connect to the camera from two sources (e.g. two camera PC applications). Used endpoints from camera view in default mode are:

- 0x01 - read endpoint
- 0x81 - write endpoint

5.2 PC application side implementation

For PC application side implementation controller - worker scheme is used. This scheme and other PC application designs are described in chapter 6. In this section, the focus is on port implementation and its problems. Available devices shown in the PC application are filtered using both PID and VID, so other devices will not show in connection menu. Two versions of Port class were created and are described in the following sections.

5.2.1 Serial port implementation

The most straightforward solution is to use a serial port class (if it is implemented in the used framework). In this case, Qt toolkit is used, which includes implementation of a serial port. As Qt is multiplatform, its serial port implementation can also be used on multiple operating systems. The most significant advantage of this option is its simplicity. Qt serial port class is very easy to use. No modification in the form of a manual install of driver or others is necessary. The connection between the camera and PC application using this implementation should work out of the box.

Unfortunately, this approach has its downfalls as well. For full speed USB, its performance is sufficient, but it seems to limit high speed USB data rate 5.5. The second

problem that is faced applies only to Windows USB CDC drivers. While sending a larger amount of data (such as images), certain parts of data are lost. Because of these problems, another solution was searched and is described in the following section. Serial port implementation was preserved in PC application and can be switched to by defines in the portController class header file. However, it is not recommended to use this option, as it might not be reliably functional.

After investigation, it was determined, that it happens on driver level. Later possible reason was discovered in UBS 2.0 specification [10]. The specification says the following:

"A bulk transfer is complete when the endpoint does one of the following:

- *Has transferred exactly the amount of data expected*
- *Transfers a packet with a payload size less than wMaxPacketSize or transfers a zero-length packet"*

For USB CDC, it would most likely mean that the second condition needs to be met to end bulk transfer. Until the transfer is ended, the driver holds the data and will not emit them further. So when a larger amount of data of full bulk packets is sent, some internal driver buffer overflows and data gets lost. There might also be some internal timeout to release the data, which would explain why this problem disappears with the slower data rate. This problem could be resolved by periodically sending zero-length bulk packets. For Linux USB CDC drivers it appears not to be the case. It seems that Linux UBS CDC driver takes NAK when the device has nothing to send, as the end of bulk transfer as well.

Small issue was also found on Linux systems. Some Linux distributions still include `modemmanager` service, that was historically used for dial-up internet access. This service periodically accesses newly connected serial ports for 15 s. During this period the serial port is nearly not usable. So after connecting the camera, it must be waited for these few seconds or simply uninstall this service, as it is nearly useless these days (`sudo apt-get remove modemmanager`).

5.2.2 libUSB implementation

It was desired not to change camera USB configuration, so there were not many options to work with USB CDC class device under Qt without developing driver. Fortunately exist widely used cross-platform library for USB device access called `libusb`[11]. This library allows accessing a USB device on driver level, without a need to develop the driver. It can disconnect currently used the driver a connect directly to the device itself. Basic initialization and connection to device can be done as following:

```
libusb_init(nullptr); // initialize libusb library
// open device and get device handle
libusb_open(libusb_device *, libusb_device_handle **);
// check if kernel driver connected
if(libusb_kernel_driver_active(libusb_device_handle **, INTERFACE)==1){
  // detach kernel driver
  libusb_detach_kernel_driver(libusb_device_handle **, INTERFACE);
}
// claim interface
status = libusb_claim_interface(libusb_device_handle **, INTERFACE);
```

After claiming the interface, both synchronous and asynchronous transfers are available. According to documentation asynchronous transfer interface is a bit more complicated, but more powerful, so it was decided to take this route. To create and submit an asynchronous transfer, it needs to be allocated, filled, and submitted. Exemplary it can be done as:

```
// allocate transfer
libusb_transfer * readTransfer = libusb_alloc_transfer(0);
// fill transfer
libusb_fill_bulk_transfer(
     readTransfer, this->dev_handle, READ_ENDPOINT,
     &this->inBuff[BUFFER_LEN - this->bufferLenAvailable],
     this->bufferLenAvailable, readDone, this, READ_TIMEOUT);
libusb_submit_transfer(readTransfer); // submit transfer
```

To make asynchronous transfers function correctly, somewhere in the code must be called `libusb_handle_events()` function to process libusb events. On this purpose separate thread is running and executing only libusb event handling. Without event handling in this way, transfer callback would never be called. Another important thing is that transfer does not have to be allocated over and over again. They can be allocated after connection and then being reused by just calling fill and submit. For detailed function description see official libusb documentation[11].

To mimic serial port implementation, two control transfers need to sent after connecting and before disconnecting the camera. These control commands are used in camera FW for connection and disconnection detection, so they are essential. On connection `SET_CONTROL_LINE_STATE` and then `SET_LINE_CODING` control commands are sent. On disconnection, it is the other way, and first is sent `SET_LINE_CODING` and then `SET_CONTROL_LINE_STATE` command.

- **Enabling libusb usage on linux**

  To allow libusb both read and write access to USB, permissions for that particular device needs to be changed. It can be done simply by adding new rule file to `/etc/udev/rules.d`. File needs to have name in format `[0-99]-NAME.rules`. In the rule file devices to apply rules to can be filtered through numerous attributes. Following example of the rule file sets read and write access for all users on devices with PID=1 and VID=1 (to make sure both USB and serial subclasses are searched).

```
file: /etc/udev/rules.d/90-usbpermission.rules

SUBSYSTEM=="serial", ATTRS{idVendor}=="0001",
ATTRS{idProduct}=="0001", MODE:="0666"

SUBSYSTEM=="usb", ATTRS{idVendor}=="0001",
ATTRS{idProduct}=="0001", MODE:="0666"
```

- **Enabling libusb usage on Windows**

  To enable operation under Windows, a supported driver must be installed for the device. Currently three different drivers are supported - `WinUSB`, `libusb-win32` and `libusbK`. The most straightforward way to install one of these drivers is to use `Zadig` [1] utility, that can detect all connected USB devices and install the selected driver for them. It is recommended to use WinUSB driver, as it was tested. One simple trick to avoid additional driver installation can be done. Device descriptor can be modified to correspond to WinUSB descriptor. The device would be than plug and play on

[1] `https://zadig.akeo.ie`

Windows system, as WinUSB driver would be automatically loaded for such a device (WinUSB driver is available in Windows from Windows XP times). However, as this project was started with the USB CDC class device and main development platform was Linux, it was not implemented.

5.3 Usage limitations

The limitation of camera use is present only while using just full speed USB connection. Because the data rate is too low for online data transfer to a PC application, it is not possible to process images bigger than the size of SRAM in the D1 domain (0x7FFFF bytes). Other functionalities should work as expected.

The second limitation is more general, but it is happening only with high speed USB, due to its nature. It is tied to PC application capability of current image redraw speed. Showing the image in its received form (100% scale and no interpolation) should lead to no problems. However, scaling the image in combination with high fps can lead to image stuttering and worsened GUI responsiveness. Same can happen with turned on interpolation for RGB image. To resolve this image shuttering simply turn off scaling and interpolation.

5.4 Camera communication protocol

This section focus on the communication protocol between the camera and PC application. Communication can be divided into two parts by the direction of data flow. Data emitted by the camera makes the camera to the PC application part. On the other hand, messages from PC application to the camera can also be seen as commands supported by the camera. It should be possible to use a different application to communicate with the camera if it complies to the described communication protocol.

5.4.1 Camera to PC communication

Data send from camera to PC application needs to be always aligned to 8 bytes, because the port implemented in the application reads data with this alignment. Data sent by camera can be divide to 3 simple categories. It can send image data, CMOS parameters and registers data. All possible messages emitted by the camera are shown in figure 5.1. The first two bytes of command always serves for command type identification.

Sending of the image is divided into three parts. Image header containing information about the image is sent first. It contains image width and height and also information about image format - RGB or monochrome. After the header comes image data itself, always in a format where one byte represents one-pixel data. Sending of the image is then ended by packet marked as image end, which contains the same information about the transferred image as the header. Only after image end packet is received by PC application, processing of the image is started.

CMOS parameters sending uses the same basic pattern for all parameters. After the two command type bytes comes the respective parameter code, followed by its data length and data itself. Available parameters codes and their data formats are shown in figure 5.2. Parameters 0-1 are only for informing PC application about connected CMOS sensor type. CMOS sensor capabilities are packed in single parameter with number 2. Numbers 3-8 mark sensors parameters limits. These parameters are needed for the PC application to not show unavailable values. Rest of the parameters 9-23 are for sending the current state of CMOS sensor.

image header							
0	1	2	3	4	5	6	
's'	'I'	'c'/'g'	uint16		uint16		align to 8 bytes
send	image	color/grayscale	image width		image height		

image payload	
image data length	
uint8[data length]	align to 8 bytes
image data	

image end							
0	1	2	3	4	5	6	
's'	'e'	'c'/'g'	uint16		uint16		align to 8 bytes
send	end	color/gray	image width		image height		

sensor data					
0	1	2	3	3+data length	
's'	's'	uint8	uint8	data	align to 8 bytes
send	sensor	param code	data length	param data	

register data					
0	1	2	3	4	
's'	'r'	uint8	uint16		align to 8 bytes
send	register	add	data		

Figure 5.1. Messages possibly emitted by camera

param code	param name	data length	data			
0	sensor name	nameLen	char * name			
1	shutter type	1	uint8 type			
2	capabilities	9	uint8 cap [9] = {rgb, aec, agc, hdr, roi, bin, skp, flip, test}			
3	width limits	4	uint16 min	uint16 max		
4	height limits	4	uint16 min	uint16 max		
5	binning steps	stepsLen	uint8 * steps			
6	skip steps	stepsLen	uint8 * steps			
7	gain limits	2+stepsLen	uint8 min	uint8 max	uint16 * steps	
8	brightness limits	2	uint16 min	uint16 max		
9	image width	2	uint16 width			
10	image heigth	2	uint16 height			
11	image color	1	uint8 color			
12	ROI enabled	1	uint8 roiEn			
13	ROI params	8	uint16 beginX	uint16 beginY	uint16 width	uint16 height
14	AEC	1	uint8 aec			
15	AGC	1	uint8 agc			
16	HDR	1	uint8 hdr			
17	brightness	1	uint8 bright			
18	gain	1	uint8 gain			
19	exposure time	4	uint32 exp			
20	binning	2	uint8 row	uint8 column		
21	skip	2	uint8 row	uint8 column		
22	flip	2	uint8 row	uint8 column		
23	test mode	1	uint8 test			

Figure 5.2. CMOS parameters code table

The last type of data coming from camera to PC application is registers data. Using these messages PC application can acquire raw CMOS sensor registers data. A message consists only from two command type bytes, one byte of sent register address and read register value with a size of 2 bytes.

5.4.2 PC application to camera communication

For communication from PC application to the camera, no alignment of data is required. Compared to communication in the other direction only a fraction amount of data is sent, as only control commands are used. The camera control protocol is very simple, and all available commands are shown in figure 5.3.

single image		
0	1	2
'r'	'l'	'o'
receive	image	one

continuous capture begin		
0	1	2
'r'	'l'	'b'
receive	image	begin

continuous capture end		
0	1	2
'r'	'l'	's'
receive	image	stop

sensor data			
0	1	2	2+data length
'r'	's'	uint8	data
receive	sensor	param code	param data

write register					
0	1	2	3	4	5
'r'	'r'	'w'	uint8	uint16	
receive	register	write	address	data	

read register			
0	1	2	3
'r'	'r'	'r'	uint8
receive	register	read	address

set to default	
0	1
'r'	'd'
receive	set default

Figure 5.3. Messages emitted by PC application

Messages can be again divided into three categories - image capture control, sensor parameters setting, and register control. Three different commands are available for image capture control. The first is for a request of a single image, triggering single image capture and sent to PC application. Other two are for continuous capture control. Continuous image capture and send are started by one of them, and the second one stops the continuous transfer. After the stop is received by the camera, processing of the current image is finished, and then the capture is stopped.

Commands for sensor parameters setting contains parameter code and data itself beside two command identification bytes. Same parameters table as for the camera to PC communication 5.2 is used, although only parameters with numbers 13-23 are available. All the other parameters are defined only in the other direction communication as they are only informing about sensor parameters. In this category of commands can also be counted set to default command, as it resets CMOS sensor to its default settings. It uses

its format different from parameters set commands. Only two command identification bytes are needed for this command because it is otherwise unique.

Independently are then defined commands for CMOS registers control. One serves for register setting and consists of write flag byte, register address byte and two bytes containing value to set. The second is used for register read and contains only read flag byte and register address to read.

5.5 Communication speed measurements

During the development of the camera, both full speed and high speed USB were employed. Both of them were preserved to the final solution, and each is usable through the different connector and is using different PHY. Two different implementations were also created for the host side (PC application). One using internal QT serial port library using USB CDC drivers and second using the libusb library to substitute driver level implementation. The PC application is also supported by two different operating system - Linux and Windows. Counting all of them we have eight different combinations for measurement. Comparing all eight options might lead to some interesting conclusion on different USB implementations performance.

5.5.1 Maximal data rates

Before the measurements itself, we might want to define absolute limits for our communication chain. The first part would be the maximal data output of supported sensors. The maximal data output of the CMOS sensor is usually with maximal supported resolution. One look on maximal data output of the sensor might be from the resolution to fps ratio. That can be expressed using the formula:

$$DataRate_{CMOS} = ActivePixels \cdot FPS \tag{5.1}$$

On the other hand, the data rate is limited by the px clock signal, as it defines how fast are individual pixels read. If 8-bit DCMI interface is used, every px clock period 1 byte of data is transferred. The total data rate is also lowered by HS and VS, but for simplification can be omitted (just noted that actual maximal data rates would be lower). If we look to datasheets of supported sensors and compute both maximal data rates, the latter one should be always higher, as it does not account HS and VS times. However maximal px clock frequencies are not used. Because of need for 24 MHz for external PHY, clock to timers peripherals has been lowered from 200 MHz to 192 MHz. Clocks for CMOS sensor are then also generated by timer, so we are limited to frequencies:

$$f = \frac{192}{2 \cdot (n+1)} \quad n \in N$$

Maximal data rates for supported sensors are shown in table 5.1. As relevant maximal data output of sensor will be furthermore used the values computed from real used px clocks.

	ActivePx · FPS	px clock (max)	px clock (used)
MT9V034	21.66 MB/s	27 MB/s	24 MB/s
MT9M001	39.32 MB/s	48 MB/s	32 MB/s
MT9T001	37.75 MB/s	48 MB/s	32 MB/s

Table 5.1. CMOS sensors maximal data output

The second limiting factor can be the transfer of data from DCMI peripheral to SRAM using DMA. Unfortunately, no new documentation like [12] is provided by the manufacturer, so we will have to assume that DMA controller works the same way on H7 series as on F2, F4, and F7. It is the only document, where DMA timing are described in detail. For AHB peripheral access time for DMA is 4 AHB cycles. Similar access to memory is also 4 AHB cycles. Working with 32-bit values, one transfer from DCMI to memory takes 8 AHB cycles. With AHB clock being 192 MHz, maximal data rate can be expressed as:

$$DataRate_{DMA} = \frac{4 \cdot 192}{8} = 96 \text{ MB/s} \tag{5.2}$$

USB itself has different maximal data rates for different releases. Two of them were implemented, and their maximal data rates are 1.5 MB/s for full speed USB 1.0 respectively 60 MB/s for high speed USB 2.0 (actual theoretical maximum data rate is lower because of protocol overhead, etc.). Other limiting factors might be processing times on both FW and application side. These will affect final maximal measured speeds. Maximal data rate limitations for communication chain parts are shown in table 5.2. For CMOS maximal data rate is chosen maximal reachable data rate of supported sensors.

CMOS	DMA	USB 1.0	USB 2.0
32 MB/s	96 MB/s	1.5 MB/s	60 MB/s

Table 5.2. Maximal data rates for communication chain parts

From table 5.2 can be easily seen that using full speed USB would marginally limit whole communication chain. On the other hand, it should be possible to match the maximal data rate of CMOS sensors using high speed USB. With fast enough USB communication, it is possible to process images, that does not fit to internal RAM.

5.5.2 Data rates measurements

Special function for measuring communication speed between the camera and PC application was build to both FW and application. The camera first informs the application of the amount of data to be sent. The application then starts timer and measures time until the expected amount of data is received. HS USB is tested by sending 512 MB of dummy data and FS USB by sending 51.2 MB of data. For both system testing was done on the same PC running Intel® Core i5-5200U and using operating systems:

- Ubuntu 18.04.1 64 bit
- Windows 10 v. 1903

	FS USB		HS USB	
	serialPort	libusb	serialPort	libusb
Linux	1100-1200 kB/s	1000-1100 kB/s	4.1-4.5 MB/s	30-39 MB/s
Windows	N/A	750-850 kB/s	N/A	29-31 MB/s

Table 5.3. Measured data rates for different USB implementations

Serial port implementation in Qt seems to yield better results for full speed USB. However, things drastically change with high speed USB, where libusb implementation reaches nearly ten times higher data rates. Reliably measuring communication speed using CDC drivers (serial port implementation) under Windows was not possible (described in

section 5.2.1). The best results were achieved using libusb under Linux OS. Similar data rates were also reached under Windows. But few timing tweaks were performed to reach these speeds (with Linux setup speed were closer to 10 MB/s). Also, it must be noted, that especially HS USB data rates are influenced by PC workload.

5.5.3 High speed USB real data rate comparison with theory

If we would like to get to actual maximal high speed USB data rate using bulk transfers, a few variables need to be accounted for. As already mentioned, the absolute maximal USB 2.0 throughput is 60 MB/s. If we count just USB protocol overhead according to [10], we get to roughly 53.25 MB/s. That is, of course, using the maximum size of bulk transfer - 512 bytes. Single microframe has 7500 bytes of useful data, and protocol overhead for bulk transfer is defined as 55 bytes. Number of bulk transfers bt in one microframe can be computed as:

$$bt = \frac{7500}{(512 + 55)} = 13.23 \tag{5.3}$$

A number of bulk transfers needs to be integer number and having 8000 microframes per second we get the data rate:

$$DataRate = 13 \cdot 512 \cdot 8000 = 53.248 \text{ MB/s} \tag{5.4}$$

However, a few other things are reducing the actual bandwidth. USB protocol is also using bit stuffing. If a sequence of six 1 is sent, it is followed by stuffed 0, which is discarded by the receiver. Host scheduling the transfers must account with the worst case scenario resulting in the increase of bulk packet transmitted size:

$$7 \cdot int\left(\frac{512}{6}\right) = 595 \tag{5.5}$$

That leads us to the new number of bulk transfers per microframe:

$$bt = \frac{7500}{(595 + 55)} = 11.54 \tag{5.6}$$

But not all the USB bandwidth is reserved for bulk transfers. The host is required to reserve about 10 % of the bandwidth for control transfers, thus further reducing the bulk transfer data rate to roughly $bt = 10.4$ bulk transfers per microframe. That translates to maximal data rate using bulk transfers of:

$$DataRate_{max} = 10 \cdot 512 \cdot 8000 = 40.96 \text{ MB/s} \tag{5.7}$$

From these computations we can see, that with high speed USB on Linux reaching up 40 MB/s actual limits are touched and high speed USB is used in its full potential.

Chapter 6
PC application

PC application was developed using Qt widget toolkit (v 5.11.2) with C++ programming language. The application supports Linux and Windows operating systems. All the functionalities are available on both platforms in full scale. However, especially performance of USB communication is significantly limited under the Windows operating system (details in chapter 5). For this reason, it is recommended to use Linux variant, if possible.

The application enables full CMOS sensor control, including registers access. Featured is also separate linear sensor mode with tools for basic analysis. User can also save currently shown image or save CMOS sensor setting for later usage. The camera can communicate with application using high speed USB (mini USB connector on interfacing board) or full speed UBS (micro USB connector on Nucleo board). Details about communication between the camera and PC application are described in chapter 5.

6.1 Application structure

The application can be divided into several modules by their functions. Also, it is convenient to move some parts of the application (e.g. communication) to separate thread. Thread-safe communication can be realized under Qt using it's SIGNAL and SLOT notations (it can also serve for communication between Qt objects). The goal of this section is not to go in detail about development under Qt, but rather to describe basic application structure. Simplified high-level application structure is shown in figure 6.1.

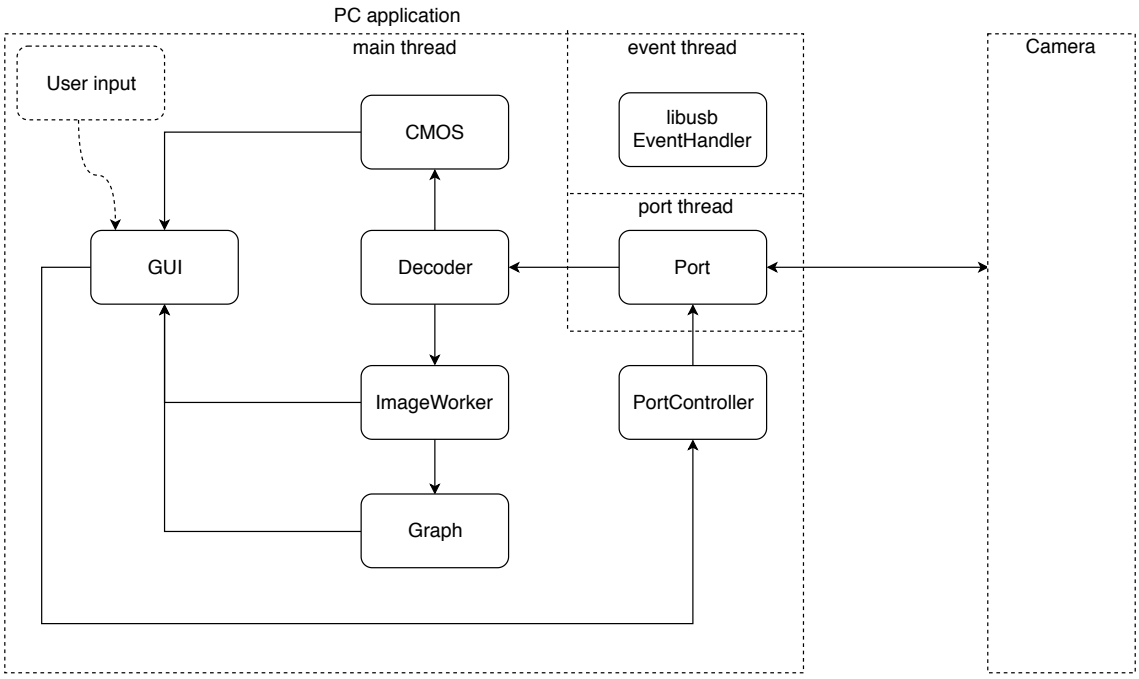


Figure 6.1. PC application structure diagram

- **GUI**
  Graphical user interface part of the application serves for informing the user about camera state and also for interaction between user and application. It visualize informations from `CMOS`, `ImageWorker` and `Graph` modules. Alongside visualization of the user interface are in the `GUI` module included submodules, taking care of correctly marking pending data or widgets pertinency depending on CMOS sensor settings.
- **CMOS**
  `CMOS` module is a representation of the sensor inside the application. It can take incoming parameters messages from `Decoder` and change its internal parameters while also informing `GUI` about parameters change. Saving and loading CMOS sensors setting feature is also provided by this module.
- **Decoder**
  The crucial module deciding what will happen with data received by the `Port`. It decodes the incoming data and after whole CMOS parameter or image is received, it forwards the data to the `CMOS` or `ImageWorker`.
- **ImageWorker**
  `ImageWorker` module takes care of all the work on the image data. Image scaling, line selection for linear mode, cursors or drawing pending selections over the image are all operations performed by `ImageWorker`. Interpolation of RGB images (details in section 6.1.1) is another big part of this module. After the image data are processed, it forwards the image to `GUI` for redraw.
- **Graph**
  Separate module for graph of linear mode data using `QCustomPlot` library[13]. After receiving data from the `ImageWorker` it performs all the work around the shown graph, including cursor control.
- **PortController**
  `PortController` acts for the rest of the application as the port itself. However, it is only a mean for communications between the main thread and `Port`. Inside this module, `Port` is created and then moved to its separate thread. It takes all commands expected from a port - connect, disconnect, or send data. Reading of data is done continuously by the `Port` and does not have to be started by the user.
- **Port**
  Actual `Port` module is running in a separate thread, as already mentioned. New read transfer is created as soon as the previous one is ended, so the port is always ready to receive data from the camera. All data are forwarded to the `Decoder` module to decide their purpose. Data are forwarded with 8 bytes alignment (the rest remain in the buffer and is forwarded with next data), so from `Decoder` point of few it reads with this 8-byte alignment. Write transfers are submitted as soon as data to send are received. More details about implementation is chapter 5. In place of the default libusb implementation can also be used older serial port implementation.
- **libusb EventHandler**
  This module is present only when libusb implementation of `Port` module is used. It is created by `Port` and serves only for servicing libusb asynchronous events. Handling of the events is done in `Port` itself.

6.1.1 Bayer filter interpolation

The only supported RGB CMOS sensor is using a Bayer mask to produce RGB images. Bayer mask is a pattern of filters that allows only red, green, or blue light to pass to a single photosensitive element of sensor representing a pixel. Different patterns are sometimes

Figure 6.2. Bayer filter structue

used as well, but Bayer mask is still the most used one. Its structure is shown in figure 6.2.

It contains 50% green, 25% blue, and 25% red pixels. By having twice more, green pixels function of the human eye is simulated (the human eye is most sensitive to green light). To obtain a full RGB image interpolation of color channels, data needs to be done. This interpolation is usually called demosaicing or debayering. Algorithms used for interpolation can be divided into two categories:

- **Non-adaptive**
  These type of algorithms are usually simple to implement, and most of the time using just linear operations. They often yield worse results producing some kind of artifacts. But as they are less complex, they tend to be less computationally intensive, thus being faster. From the most used ones can be named Nearest Neighbor, Bilinear interpolation or Cubic convolution.
- **Adaptive**
  Adaptive algorithms are called adaptive because they can, to some extent, adapt to features in the image (e.g. edges). They often use nonlinear operations and thus being more computationally intensive. But that is just a drawback for the better final result. From the many existing algorithms can be named, for example, Edge sensing interpolation or Patterned pixel grouping.

Two demosaicing algorithms are implemented in the PC application. The first one is used when continuous RGB image transfer is on (video). In this case, the main concern is a speed of given algorithms, as interpolation needs to be done in real-time. The least computationally intensive algorithm should be **Nearest Neighbor** (NN). Missing values are in this case set to value of the nearest value from a given color channel. If we have a look on pixels 22, 23, 32, 33 from figure 6.2 it would to translate to given equations:

$$R22 = R11; \quad G22 = G21; \quad B22 = B22 \tag{6.1}$$

$$R23 = R13; \quad G23 = G23; \quad B23 = B22 \tag{6.2}$$

$$R32 = R31; \quad G32 = G32; \quad B32 = B22 \tag{6.3}$$

$$R33 = R33; \quad B33 = B22; \quad G33 = G32 \tag{6.4}$$

The second algorithm is used for single images, where a bit longer processing does not an issue. To yield a bit better results, **Bilinear interpolation** is used. This algorithm interpolates missing values from the neighborhood of original values form given color channel. Equations for pixels 22, 23, 32, 33 looks following:

$$R22 = \frac{R11 + R13 + R31 + R33}{4}; \quad G22 = \frac{G12 + G21 + G23 + G32}{4}; \quad B22 = B22 \tag{6.5}$$

$$R23 = \frac{R13 + R33}{2}; \quad G23 = G23; \quad B23 = \frac{B22 + B24}{2} \tag{6.6}$$

$$R32 = \frac{R31 + R33}{2}; \quad G32 = G32; \quad B32 = \frac{B22 + B42}{2} \tag{6.7}$$

$$R33 = R33; \quad G33 = \frac{G23 + G32 + G34 + G43}{4}; \quad B33 = \frac{B22 + B24 + B42 + B44}{4} \tag{6.8}$$

6.2 GUI layout

This section of the thesis is focused on the description of the PC application GUI and its functions. It can also serve as a manual to the application. The GUI can be divided into three parts - Main control area, Area Mode, Linear Mode. Following subsections focus on individual parts and describes their functionalities.

Pending values are marked through the entire application by red color. By pending values are meant, change parameters, that have not been set yet. Examples of pending values can be seen for example in figure 6.5 - both `Exposure` and `Row skip` contains currently pending values. Same can be seen in figure 6.7 - `Selected line` value is also pending.

6.2.1 Main control area

The main control area is always visible to the user and contains basic application controls. Things like file saving, connection control, capture control, or image view control are included in this area. All present functanddd shown in figure 6.3. Follows their descriptions using numbering from figure:

1. **Action menus**

   The action menu is located in the left top corner of the application. The first menu is called `File` and includes options for image or data saving. Following options are present in `File` menu:

   - `Save image` - to save currently displayed image in png, jpg or bmp format
   - `Save line` - to save currently selected image line in png, jpg or bmp format
   - `Save line data` - to save currently selected image line data in csv format
   - `Save CMOS settings` - to save current CMOS settings to file
   - `Load CMOS settings` - to load CMOS settings from file

   The second menu `Test` for controlling test modes of connected CMOS sensor. Test mode can be turned on/off, and if more test patterns are available, they can be selected. `Help` menu contains just one item, that evokes window with information about the application.

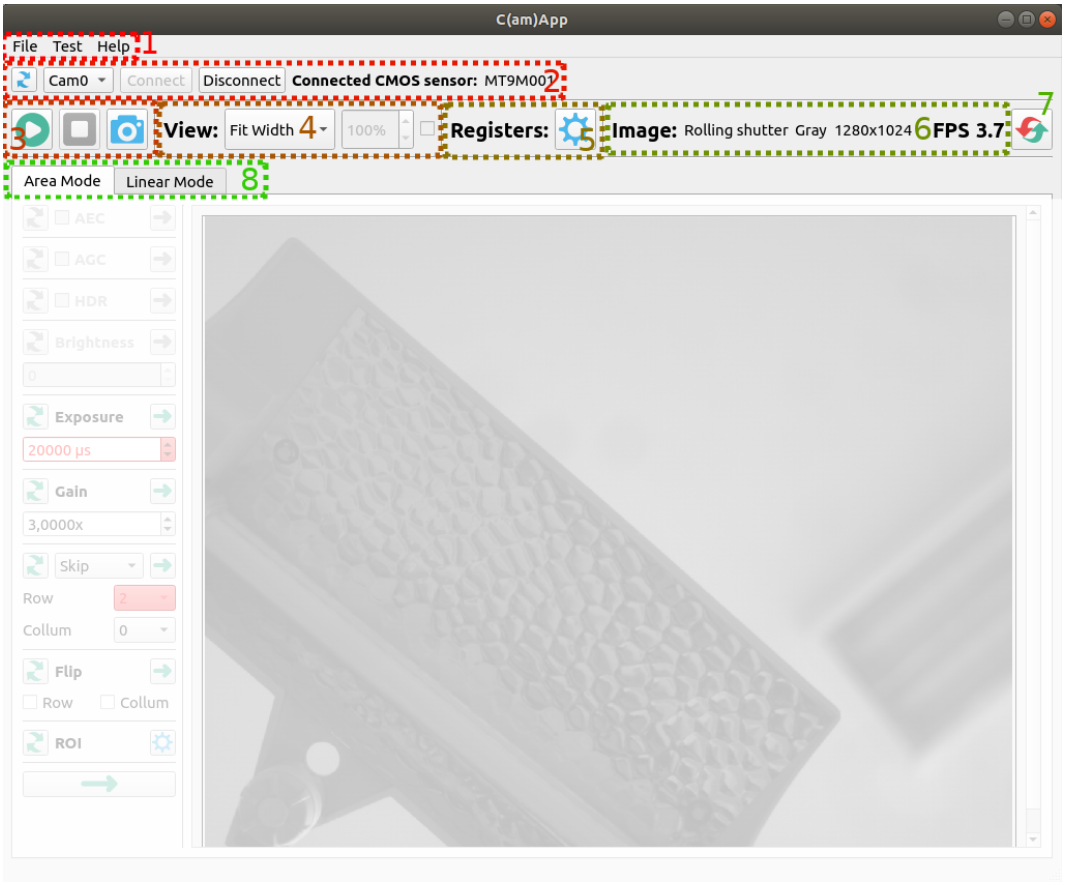


Figure 6.3. PC application main controls

2. **Camera connection control**
   Camera connection control area contains form left to right following items:

   - `Refresh devices button` - to refresh list of connected USB devices (filtered using PID and VID)
   - `List of connected device` - to select connected device to work with
   - `Connect button` - to establish connection with selected USB device
   - `Disconnect button` - to close connection with currently connected USB device
   - `Connected sensor` - shows type of connected CMOS sensor

3. **Capture control**
   Three buttons included in this section serves as image capture control. Buttons icons should be self-explanatory, but to makes things clear, it will not hurt to mention buttons functions. From left to right, the first button starts continuous image transfer, next to it is stop continuous transfer button. On the right side is then the snapshot button, that induces single image capture and its transfer to the application.
4. **Image display control**
   To change current image scale controls in this section can be used. The main select box contains present scale - 20%, 50%, 100%, 200%. There is also `Arbitrary` option that enables spin box next to the select box. Scale can be then changed continuously between 1-999%. One possibility for scale change is to use spin box directly or to move the mouse over the image and use `ctrl + mouse wheel`. Two options that change image scale dynamically with application window size are also present. `Fit Window` scales the image to fit the image area and `Fit Width` scales to fit just width of the

image to the image area. On the right side of this section is also checkbox, that turns on/off interpolation for RGB images (it is not pertinent for monochrome images).

5. **Registers control**

   An only single button is present in this area, but it evokes a dedicated window for registers control shown in figure 6.4. There are `Read` and `Write` buttons operating with respective fields above them. `Address` fields allow only 2 characters and the values are in hexadecimal number format. Fields named `Value` shows 4 characters with space between because sensors registers have 16 bits format. The values are again in hexadecimal format, and the first two characters represent the most significant byte.

Figure 6.4. PC application registers control window

6. **Current image status**

   User cannot directly interact with current image status as it only serves only for showing image parameters. It contains information about the used type of shutter, image format (monochrome/RGB) and actual transferred image size. The last item included in this area is fps counter, showing a number of frames per second if the continuous transfer is on.

7. **Default setting button**

   Button resetting CMOS sensor on connected camera to default settings.

8. **Mode selection**

   Selection for changing shown user interface between `Area Mode` (contains general sensor settings) and `Linear Mode` (simulates linear sensor and also contains basic tools for line data analysis).

6.2.2 Area mode

Area mode section of application includes controls for changing available CMOS sensor settings. It also includes an area for displaying the latest received image. Screenshot with highlighted parts is presented in figure 6.5. Following the description of individual parts uses the same order as on the presented screenshot. If the currently used sensor does not support one of the functionalities, it is not available (controls for given functionality are not pertinent).

Small buttons with two green arrows as glyph appears many times on in this area. All of them are located on the left side of the application. Their serve for loading currently set value to their respective parameter. After the user changes parameters values, it is marked as pending (red color). Using this button currently set value of the parameter can be reloaded, so the user does not have to search it manually.

Similar, we can notice a button with a single green arrow as glyph being more than once on the right side of the parameters panel. Its purpose is to a simple set respective parameter to the camera. After the camera tries to set the value, it sends back the real parameter value for GUI to show. If the value user just tried to change remain the same, the setting was successful.

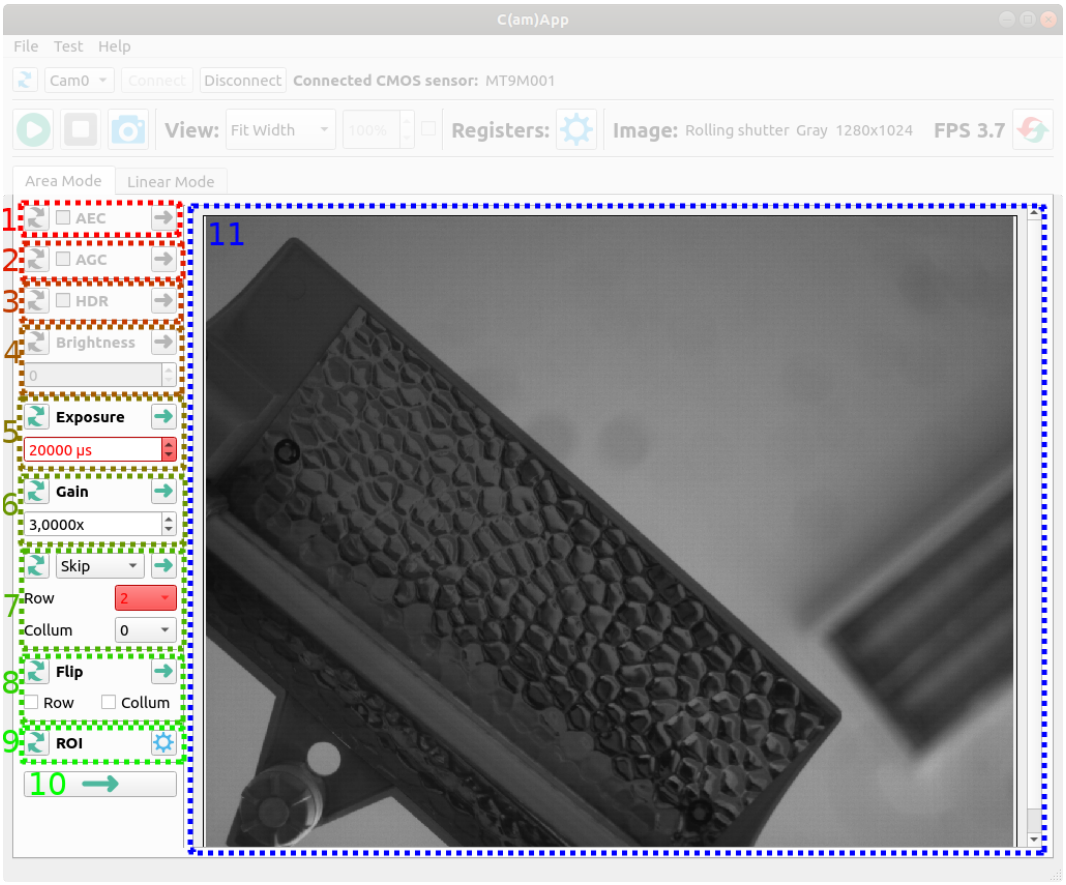


Figure 6.5. PC application area mode

1. **AEC control**
   Control elements for automatic exposure control that can be turned on or off. Includes refresh button, checkbox, and set button. AEC is functionality, which allows the sensor to change exposure time, depending on the image brightness dynamically.
2. **AGC control**
   Control elements for automatic gain control that can be turned on or off. Includes refresh button, checkbox, and set button. AGC is functionality, which allows the sensor to dynamically analog gain, depending on the image brightness.
3. **HDR control**
   Control elements for high dynamic range that can be turned on or off. Includes refresh button, checkbox, and set button. HDR can increase the dynamic range of the sensor by a significant margin.
4. **Brightness control**
   Control elements for image brightness. Includes refresh button, spin box, and set button. Brightness parameter is pertinent only when AEC or AGC is turned on. It can be set to a discrete value representing the overall brightness of the image, that is to be maintained by the sensor (using exposure and gain control).
5. **Exposure time control**
   Control elements for exposure time. Includes refresh button, spin box, and set button. The value presented in the spin box is in μs. Exposure time is the amount of time photosensitive elements of the sensor are exposed to light.

6. **Analog gain control**
   Control elements for analog gain. Includes refresh button, spin box, and set button. Analog gain parameter represents gain of sensors internal amplifiers.
7. **Pixel skip/binning control**
   Control elements for pixel skip and binning. Includes refresh button, select box for a skip or binning selection, set button, and also individual select boxes for row and column. Values for row and column says how many times will be the height and width of the image reduced. Different values can be set for row and column. Select box for a skip or binning selection contains only values supported by used CMOS sensor.
   Pixel skip just skips rows or columns in the image and so only reduced image is read. If the skip is 2, it means only every second row/column is read. For pixel binning things are a bit more complicated. The equivalent value of pixel skip and binning will produce an image of the same size, but the size is reduced by different means. Instead of just skipping the data, binning averages the data from that area resulting in reduction. So when both row and column binning are set to 2, every pixel is the average of signal from 4 photosensitive elements. Binning can reduce aliasing impact on the image in case of the image reduction, so if possible, it is the preferred option.
8. **Image flip control**
   Control elements for image flipping. Includes refresh button, set button, and check-boxes for row and column flip separately. By setting row/column flip order in which rows/columns are read is flipped. So row flip mirrors the image over the horizontal axis and column flip over the vertical axes.
9. **ROI control**
   ROI control elements are located in a dedicated window shown in figure 6.6

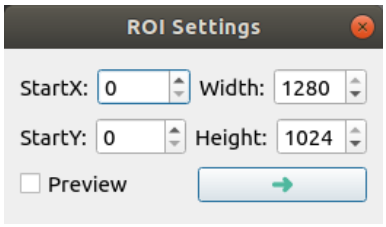


Figure 6.6. PC application ROI setting window

   To set ROI (or sometimes called windowing) there are 4 values to adjust - `StartX` is number of the first column , `StartY` is number of the first row, `Width` is image width (number of columns) and `Height` is image height (number of rows). It is also possible to show the pending ROI area to the current image by enabling `Preview` checkbox. The button with a short green arrow is the set button for ROI.
10. **Set all button**
    Button with longer green arrow serving for setting all sensor parameters above. Can be used instead of individual parameters set buttons.
11. **Image area**
    Area of the application displaying the latest received image. The image can be scaled, show ROI pending window, or be interpolated (only RGB). If the image is bigger the current size of the image area, scroll bars appears on the right side and bottom of the area. Using scroll bars user can view the whole image that will not fit the image area.

6.2.3 Linear mode

By switching to the `Linear Mode` tab, we get access to some basic tools for line analysis. This mode is to be mainly used as a simulation of a linear image sensor, that can be simulated on application or camera level. For basic line data analysis, there are implemented cursors. Highlighted parts of linear mode screen are shown in figure 6.7. Description of individual parts follows numbering from the figure.

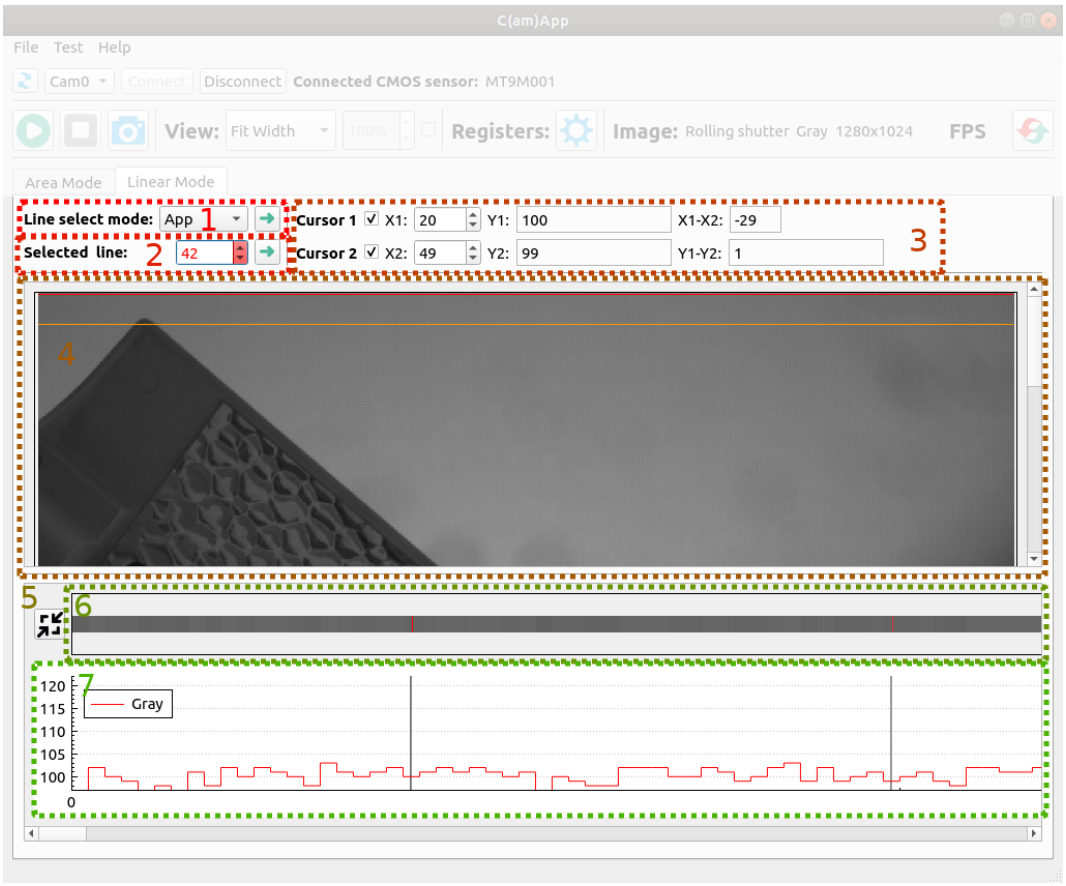


Figure 6.7. PC application linear mode

1. **Line select mode**

   By changing `Line select mode` level on witch linear mode is simulated can be changed. When `App` option is set, the whole image is transferred to the application and line is selected on the application level. By setting the `MCU` option, the line is selected on camera level, and only the selected line is captured and transferred to the application. Option are selected using select box and value is then set by pressing the set button (with short green arrow glyph). `MCU` option for line select mode is not available for RGB sensors, as it is not possible to select only a single line. However, the selection on the application level is fully functionally.
2. **Line select**

   Serves for selecting a line from image to work with. Set button (with short green arrow glyph) is present again at the right side of the marked area. If the `Line select mode` is set to `App` currently selected line and pending select are marked in the image shown in `Image area`. Currently selected line is marked by red color and pending line by orange color.

3. **Cursors control**

   Cursors serve as a tool for basic analysis of selected line data. There two cursors available, that can be turned on/off when needed. Adjustable is their X position that represents moving through individual pixels of line. The brightness of pixel on that position is then shown in `Y1` or `Y2` field. Differences between cursors `X1-X2` and `Y1-Y2` are also shown is individual fields. When enabled cursors are shown in both `Line view` and `Line graph`.

4. **Image area**

   Shows latest received image, if the `Line select mode` is set to `App`. In this case, both the currently selected line and pending line are highlighted. Using scroll bars user can view the whole image that will not fit the image area. `Image display` controls also applies to the image shown in this area.

5. **Home graph button**

   By pressing this button currently shown line graph Y axis is scaled, so all the pixel values are visible. Meaning minimal value on the Y axis is the minimum brightness value from the line and maximal value on the Y axis is maximal brightness value from the line.

6. **Line view**

   Shows magnified selected line. Scale cannot be changed by the user, but scroll bar common with a graph can be used to view the whole line. If the cursors are enabled, they are also drawn on displayed line in red colors (slightly different shades for each cursor).

7. **Line graph**

   Graph representation of selected line data is shown in this area. On Y axis is brightness of given pixel, hence range is 0-255. On X axis is the position of the pixel in image and range depends on image width. When working with RGB image, three plots are shown, each representing one color channel. Cursors are also visualized to the graph if they are enabled (black and gray colors).

Chapter 7
Conclusion

All the goals from the specification were satisfied within the thesis. The main goal was to design and realize measurement camera with CMOS sensor for educational laboratories. It would be possible to use a commercially available camera. However, custom made camera will allow full access to all CMOS sensor settings, which can help with explaining sensors properties. Alongside the camera realization programs for both camera and PC were created. The camera is complemented by PC application providing camera control and other functionalities.

In the design phase, it was decided to use modular construction for the camera. The camera is made from three parts. Control of CMOS sensor and communication with PC is carried out by the microcontroller STM32H743 on Nucleo development board. Used CMOS sensors have their own board developed in the previous thesis in the laboratory of Videometry (with standardized physical interface). For the connection between Nucleo and CMOS board interfacing board was designed. This modular solution allows simple replacement in the case of hardware failure. Only interfacing board needs to be assembled, which is much more straightforward than a single board solution containing all components on one board. The modular solution also enables support for multiple CMOS sensors, as they can be swapped.

The camera supports three different CMOS sensors. The number of supported sensors can be easily increased thanks to FW design. For communication with PC is used USB. To enable real-time image data transfer high speed USB support is added by external PHY placed on interfacing board. Real-time image data transfer enabled processing of images, that do not fit microcontrollers internal RAM. The functionality of internal full speed USB PHY is preserved and can be used through micro USB connector on Nucleo board.

PC application complementing the camera is developed under Qt widget tooling. Version for Linux and Windows operating systems was created. The application allows the user to control the camera on sensor level. The user even has direct access to sensors registers. Included is also a linear mode simulating linear image sensor. Basic tools for line analysis are present in linear mode. Naturally, features for image saving and CMOS sensor setting saving are present.

Possible improvements of the realized solution still come in mind. The number of supported CMOS sensors could be increased. Continuous image grab mode can also be improved, as sensors are generally running continuous mode and frames are dropped during sending. PC application could be more stable with nonstandard inputs. More analysis tools may be added to the application. It would also be desired to improve image redraw speed in the application, but it is maybe limited by Qt toolkit.

References

[1] Fischer, Jan. *Optoelektronické senzory a videometrie*. Prague: Skripta ČVUT FEL, 2002. ISBN 80-01-02525-1.

[2] STMicroelectronics. *AN5020, Application note - Digital camera interface (DCMI) for STM32 MCUs, Rev. 1*.
http://www.st.com.

[3] STMicroelectronics. *LFXX Datasheet, Rev. 31*.
http://www.st.com.

[4] STMicroelectronics. *UM1974, User manual - STM32 Nucleo-144 boards, Rev. 7*.
http://www.st.com.

[5] STMicroelectronics. *LD1117 Datasheet, Rev. 33*.
http://www.st.com.

[6] Semiconductor Components Industries, LLC. *MT9V034 Datasheet, Rev. G*.
https://www.onsemi.com.

[7] Micron Technology, Inc. *MT9M001 Datasheet, Rev. F*.
https://www.micron.com.

[8] Micron Technology, Inc. *MT9T001 Datasheet, Rev. D*.
https://www.micron.com.

[9] STMicroelectronics. *AN4891, Application note - STM32H74x and STM32H75x system architecture andperformance software expansion for STM32Cube, Rev. 2*.
http://www.st.com.

[10] USB Implementers Forum, Inc. *Universal Serial Bus Specification, Rev. 2.0*.
https://www.usb.org.

[11] libusb. *A cross-platform user library to access USB devices* .
https://libusb.info/. (cit. 2019-05-11).

[12] STMicroelectronics. *AN4891, Using the STM32F2, STM32F4 and STM32F7 Series DMA controller, Rev. 3*.
http://www.st.com.

[13] Eichhammer, Emanuel. *QCustomPlot - Qt C++ widget for plotting and data visualization*.
https://www.qcustomplot.com/. (cit. 2019-05-14).

[14] Vodseďálek, Jakub. *Camera Based Vibration Sensing System*. Prague: CTU FEE, Department of Measurement, 2017. Bachelor thesis.

[15] Řípa, Radek. *Synchronized Image Sensors*. Praha: CTU FEE, Department of Measurement, 2012. Master's thesis.

[16] Davies, E.R. *Computer and Machine Visions: Theory, Algorithms, Practicalities*. 4. ed. Oxford: Academic Press, 2012. ISBN 978-0-12-386908-1.

[17] Yiu, Joseph. *The Definitive Guide to ARM® Cortex®–M3 and Cortex®–M4 Processors*. 3. ed. Oxford: Newnes, 2013. ISBN 978-0-12-408082-9.

[18] STMicroelectronics. *AN4296, Application note - Overview and tips for using STM32F303/328/334/358xx CCM RAM with IAR EWARM, Keil MDK-ARM and GNU-based toolchains, Rev. 3*.
http://www.st.com.

[19] STMicroelectronics. *RM0433, Reference manual - STM32H743/753 and STM32H750 advanced ARM®-based 32-bit MCUs, Rev. 5*.
http://www.st.com.

[20] STMicroelectronics. *DS12110, STM32H743xI Datasheet, Rev. 5*.
http://www.st.com.

[21] Microchip Technology, Inc. *DS00001792E - USB3320 Datasheet, Rev. E*.
https://www.microchip.com.

[22] USB Implementers Forum, Inc. *Universal Serial Bus Class Definitions for Communications Devices, Rev. 1.2*.
https://www.usb.org.

[23] USB Implementers Forum, Inc. *Universal Serial Bus Communications Class Subclass Specification for PSTN Devices, Rev. 1.2*.
https://www.usb.org.

Appendix A
Abbreviations

AEC	Automatic Exposure Control. Image sensor feature to automatically adjust exposure time to reach desired image brightness.
AGC	Automatic Gain Control. Image sensor feature to automatically adjust gain to reach desired image brightness.
AHB	Advance High-performance Bus. Full duplex parallel bus used for connecting functional block (peripherals) in system-on-a-chip designs.
ARM®	Advanced RISC Machine. Type of computers architecture.
CCD	Charge-coupled device. In this thesis used as "CCD image sensor", which is a type of image sensor.
CMOS	Complementary Metal–Oxide–Semiconductor. The technology used in integrated circuits. In this thesis used as "CMOS image sensor", which is a type of image sensor.
CPHA	Clock PHAse. Type of setting of SPI SCKL signal.
CPOL	Clock POLarity. Type of setting of SPI SCKL signal.
CTE	Charge Tranfer Efficiency. Parameter of charge shift register in CCD image sensor specifying efficiency of charge transfer.
CTU	Czech Technical University. Meaning Czech Technical University in Prague.
DCMI	Digital CaMera Interface. Parallel interface for digital image sensors.
DMA	Direct Memory Access. Type of direct data transfer between memory and peripheral (or memory). Data are transferred using DMA peripheral, which saves processor time.
DTCM	Data Tightly Coupled Memory. Type RAM to witch core has fast access. Used time-critical data.
FEE	Faculty of ELectrical Engineering. Meaning faculty on Czech Technical University in Prague.
fps	frames per second
FW	FirmWare. Software for device specific hardware.
GCC	GNU Compiler Collection. Compiler system by the GNU project for multiple programming languages.
GND	Ground.
GPIO	General Purpose Input Output. Signal pin of integrated circuit that can work in general manner in input or output mode.
GUI	Graphical User Interface. Graphical interface that allows the user to interact with software.
HAL	Hardware Abstraction Layer. Layer of software abstracts hardware parameters for higher level layers.
HDR	High Dynamic Range. In context of CMOS sensor mode increasing sensors dynamic range.
HS	Horizontal Synchronization. Signal indicating star/end of line.
HW	HardWare. Physical part of device.

- I^2C — Inter-Integrated Circuit. Two-wire synchronous serial communication bus commonly used in embedded systems.
- IDE — Integrated Development Environment. A software application that provides tools and environment to programmers for software development.
- ITCM — Instruction Tightly Coupled Memory. Type RAM to witch core has fast access. Used time-critical instructions.
- JPEG — Joint Photographic Experts Group. Lossy compression used for digital images.
- LED — Light Emitting Diode. Semiconductor part emitting light when powered.
- LL — Low Level. Used for STM32 libraries for direct hardware access (no abstraction in difference to HAL).
- MISO — Master Input Slave Output. Name of one of the defined SPI signals. The signal used for transmitting data from slave to master.
- MOSI — Master Output Slave Input. Name of one of the defined SPI signals. The signal used for transmitting data from master to slave.
- OS — Operating System.
- PC — Personal Computer.
- PCB — Printed Circuit Board.
- PHY — PHYsical layer.
- PX — PiXel clock. Synchronization signal indicating individual pixel data valid.
- RAM — Random Access Memory. Type of memory with direct access allowing write and read operation.
- RGB — Red Green Blue. Additive color model using reg, green and blue colors.
- ROI — Region Of Interest. Chosen samples from data indentified for particular purpose.
- RTOS — Real-Time Operating System. Operating system specializated for real-time applications.
- SCL — Serial CLock. Name of one of the defined I^2C signals. The clock signal from master serving for serial synchronization.
- SCLK — Serial CLocK. Name of one of the defined SPI signals. The clock signal from master serving for serial synchronization.
- SDA — Serial DAta. Name of one of the defined I^2C signals. Signal for data transfer.
- SPI — Serial Peripheral Interface. Four-wire synchronous communication bus commonly used in embedded systems.
- SRAM — Synchronous Random Access Memory. Type of RAM with synchronous data transfer.
- SS — Slave Select. Name of one of the defined SPI signals. Signal use for slave selection in case of multiple slaves are present on bus.
- ULPI — UTMI Low Pin Interface. Standardized 12-pin interface for connection USB core logic with transceiver.
- USB — Universal Serial Bus. Universal bus used mostly for connection computer peripherals.
- USB OTG — USB On The Go. USB standard enabling to switch between device and host role depending on connection.
- UTMI — USB Transceiver Macrocell Interface. Standardized interface for connection USB core logic with transceiver.
- VCP — Virtual COM Port. Virtualized connection behaving as COM/Serial port.
- VS — Vertical Synchronization. Signal indicating star/end of frame.

Appendix B
Interfacing board documentation

Part	Reference	Package	Quantity
cap 100 nF ceramics	C2, C3, C4, C6, C7, C11, C12, C13, C14, C15, C16, C19	SMD0805	12
cap 10 μF tantal	C1, C5, C8, C9 C10, C17, C18	SMD1206	7
res 1 kΩ	R2, R4, R5, R6, R7	SMD0805	5
res 10 kΩ	R1	SMD0805	1
res 8.06 kΩ $\pm$1%	R3	SMD0805	1
USB3320	U1	QFN32	1
LF33CDT-TR	CR1	DPAK	1
LF18CDT-TR	CR2	DPAK	1
LED	PWR_LED1, LED1, LED2 LED2, LED4	SMD1206	5
diode SUF4007	D2	MELF	1
USB mini connector	J6	SMD	1
DC connector	J5	thru-hole	1
3x1 header	JP1	thru-hole 2.5 mm	1
8x2 header	J1	thru-hole 2.5 mm	1
10x2 header	J3	thru-hole 2.5 mm	1
15x2 header	J2, J7	thru-hole 2.5 mm	2
17x2 header	J4	thru-hole 2.5 mm	1

Table B.1. Bill of material for the final interfacing board

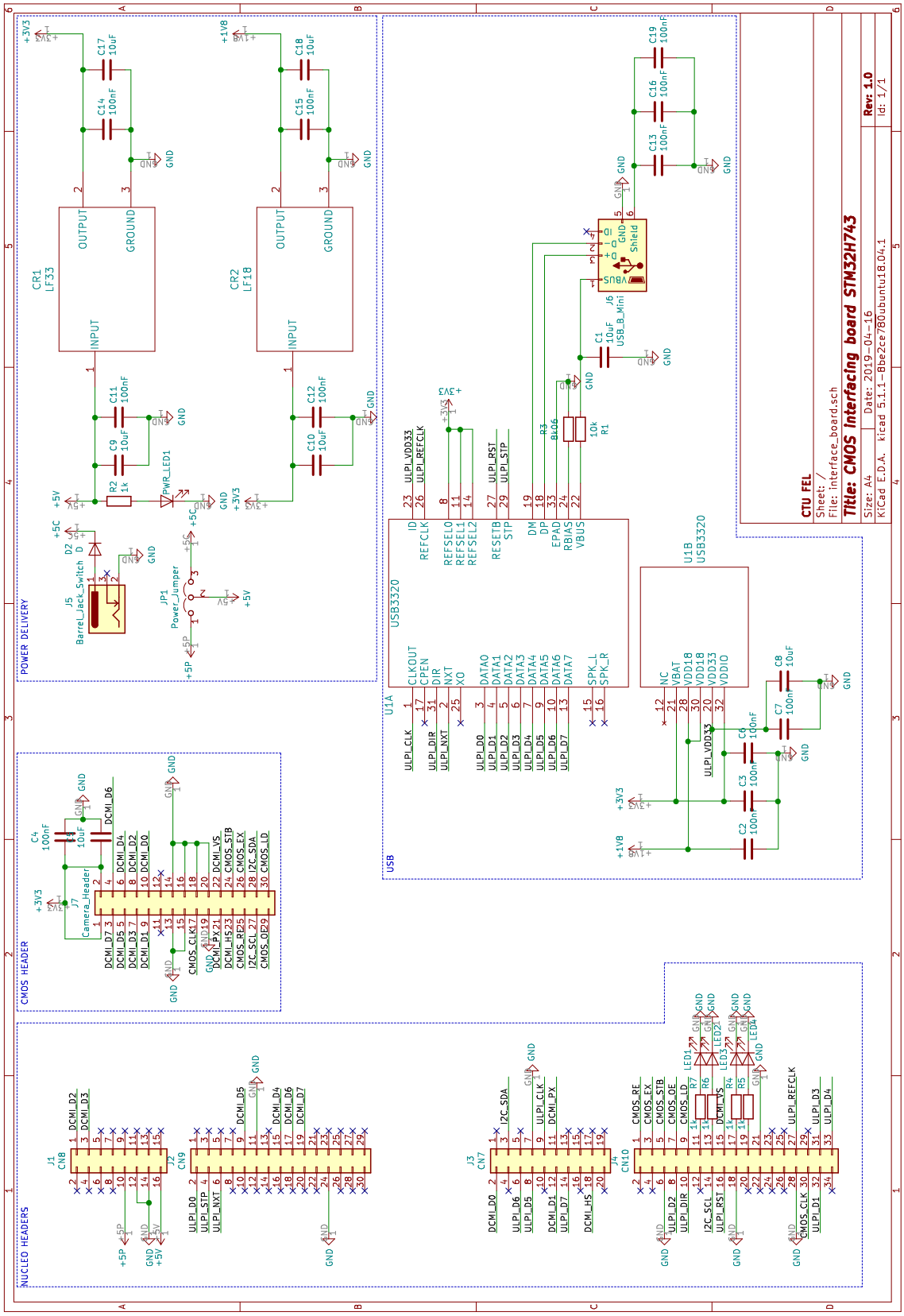


Figure B.1. Schematics of the final interfacing board

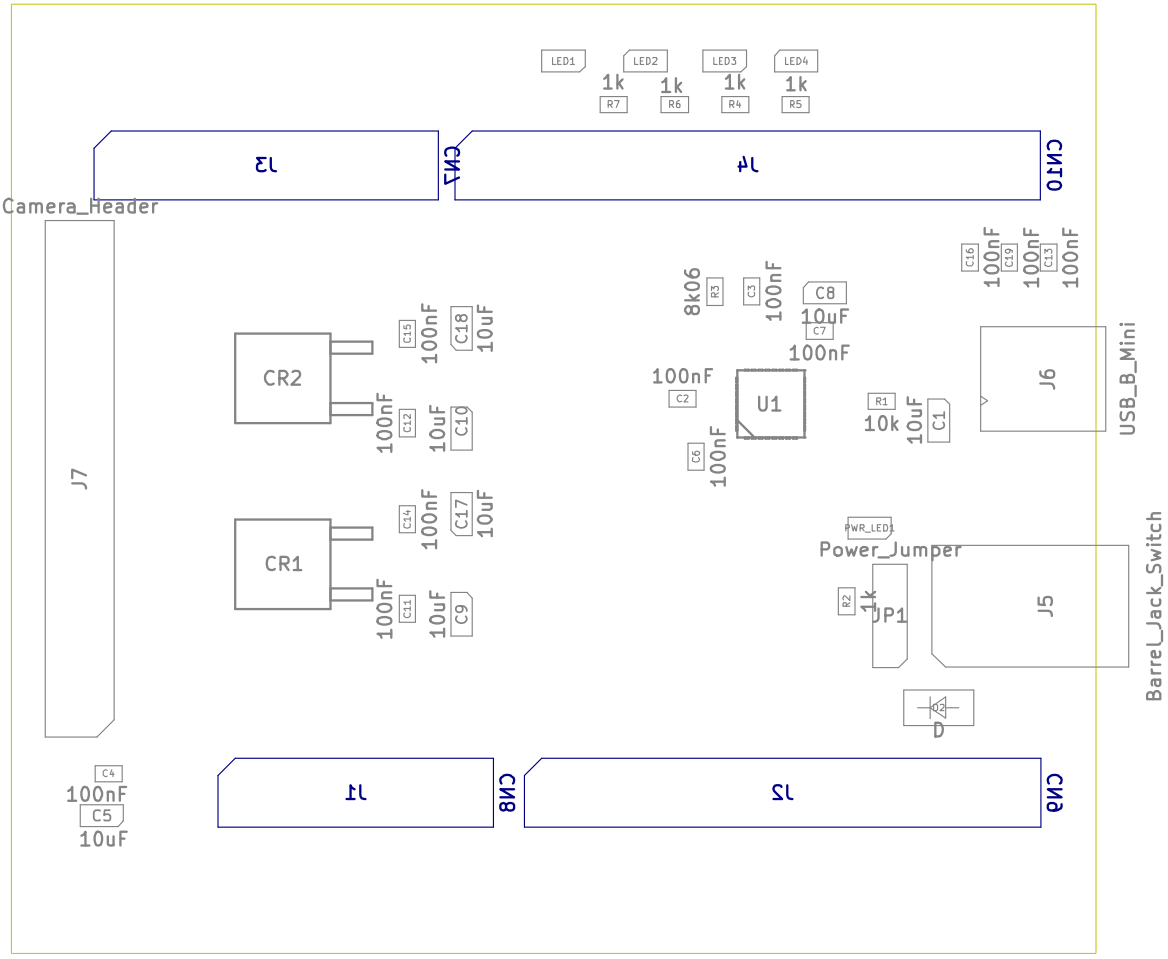


Figure B.2. Assembly drawing for the final interfacing board

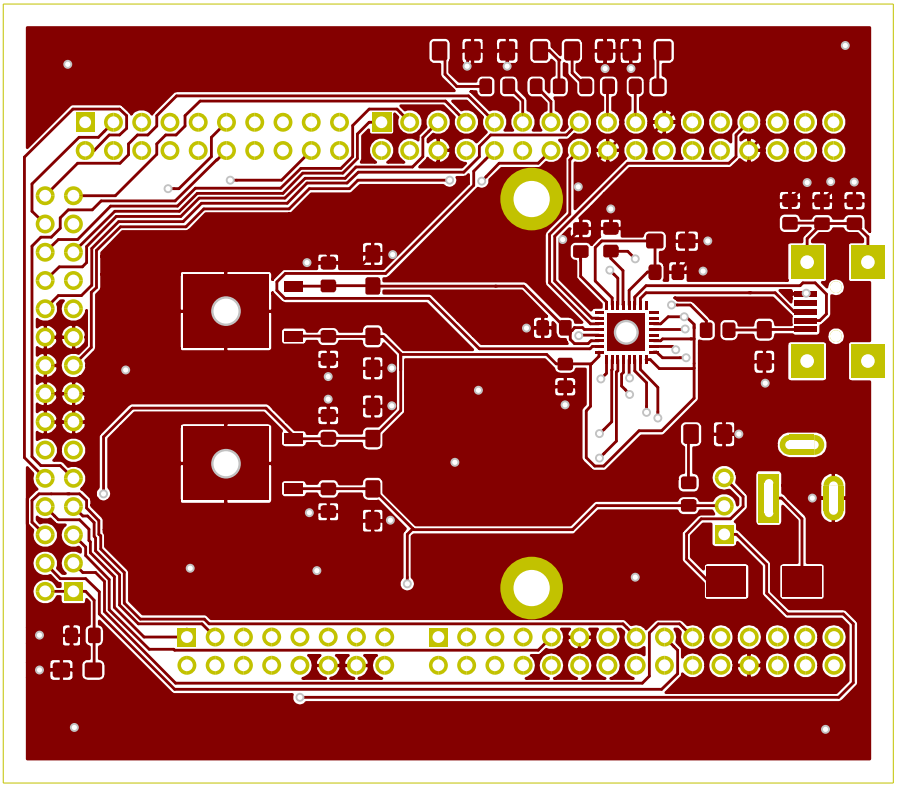


Figure B.3. Top copper drawing for the final interfacing board

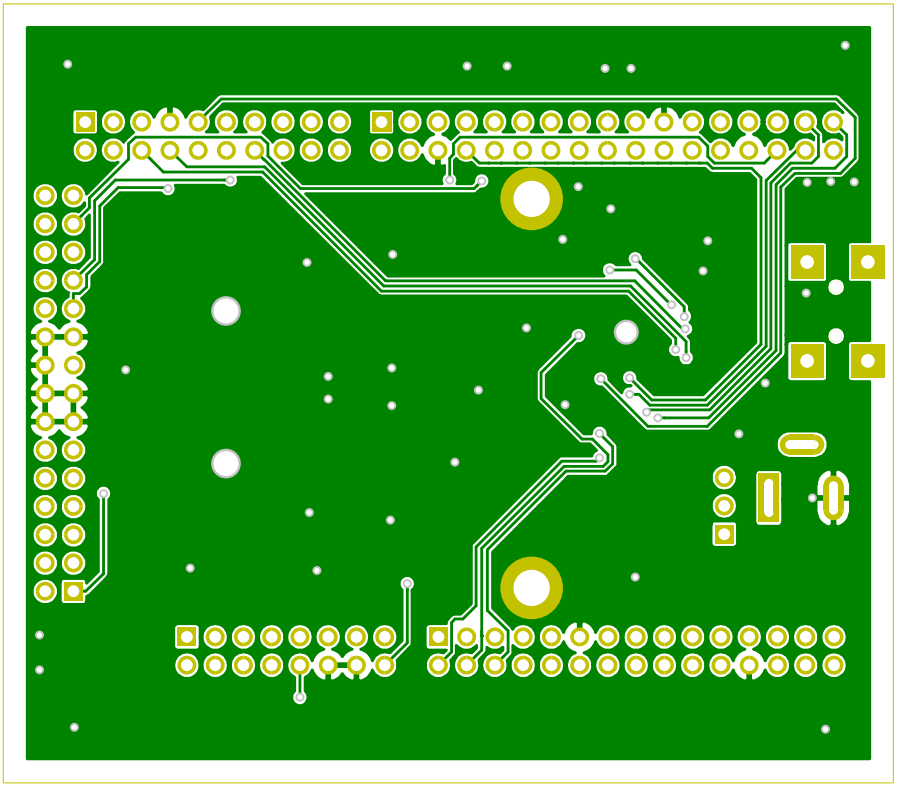


Figure B.4. Bottom copper drawing for the final interfacing board

Appendix C
Photo documentation

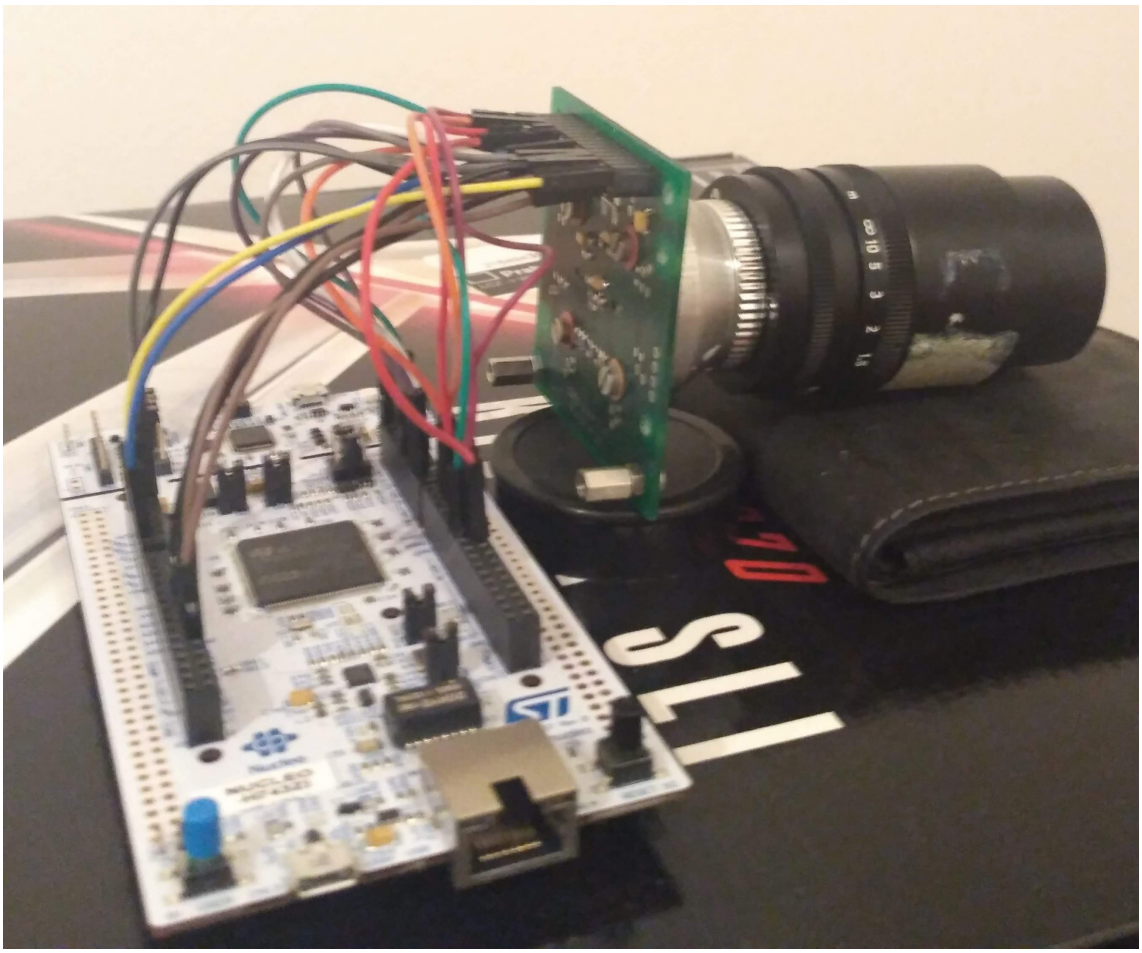


Figure C.5. Early stages of development - CMOS board connect to Nucleo board with wires

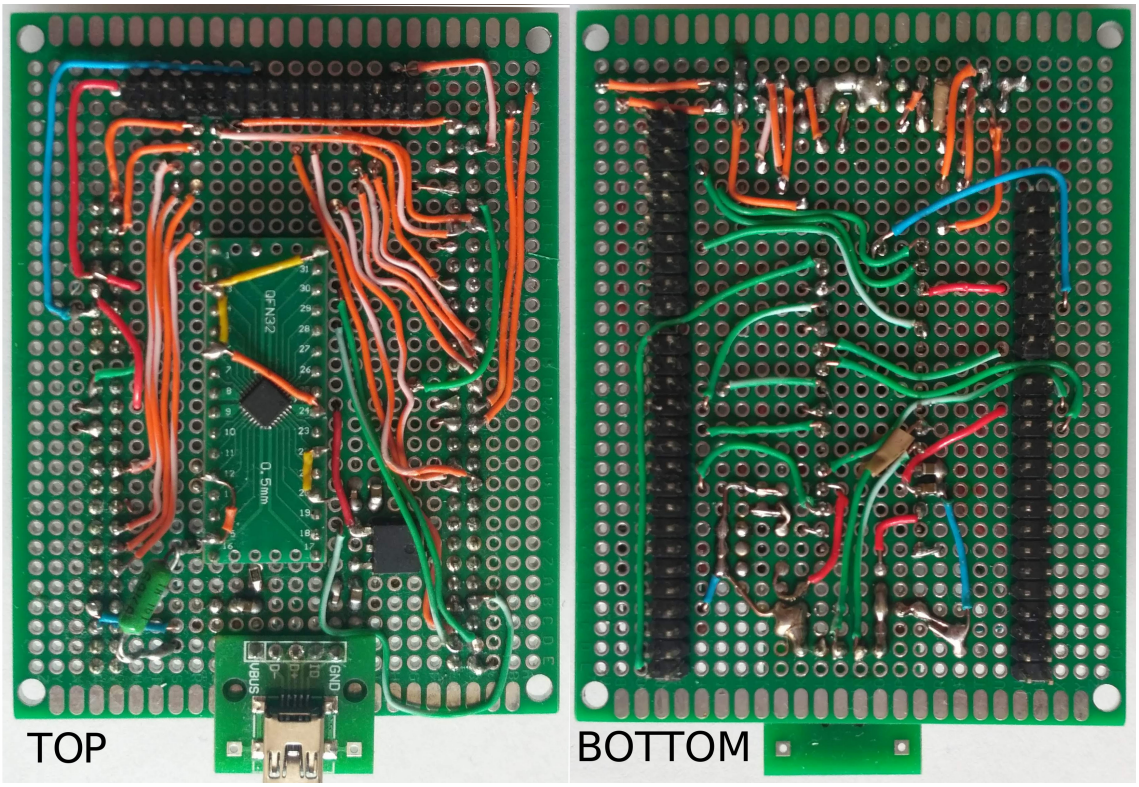


Figure C.6. Prototype of interfacing board

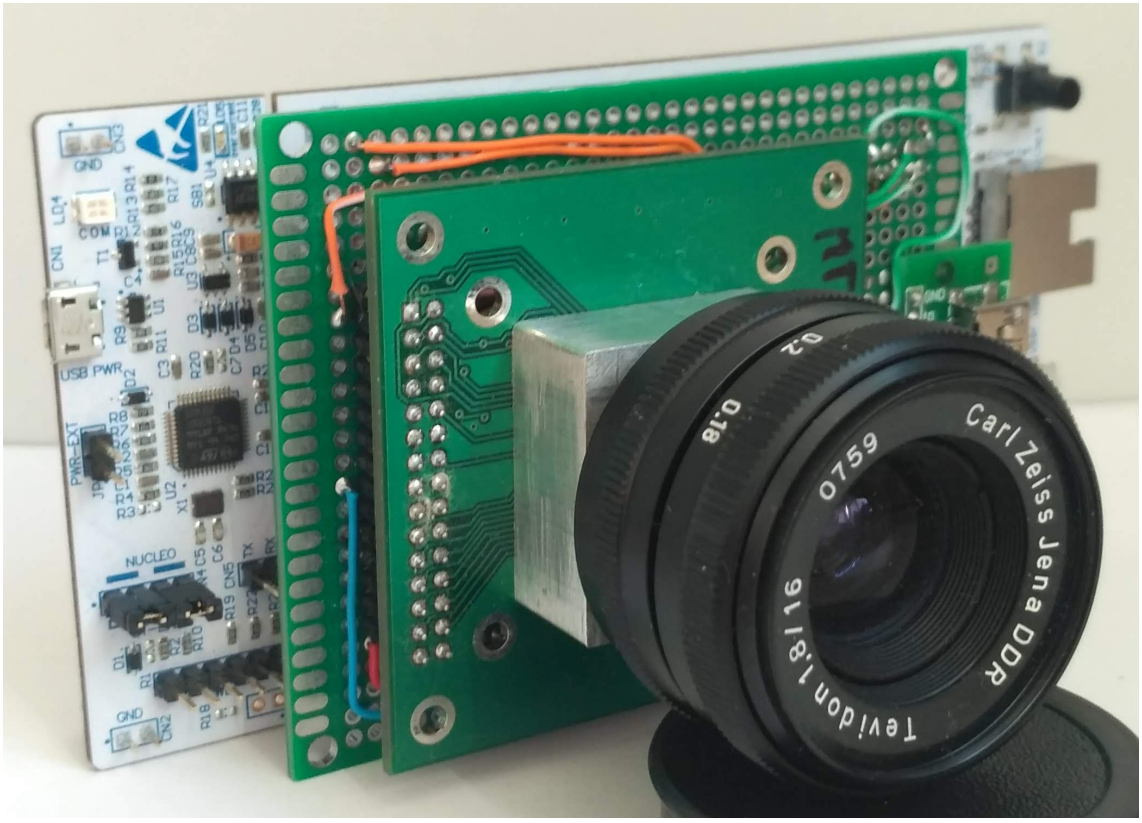


Figure C.7. Whole camera with prototype of interfacing board

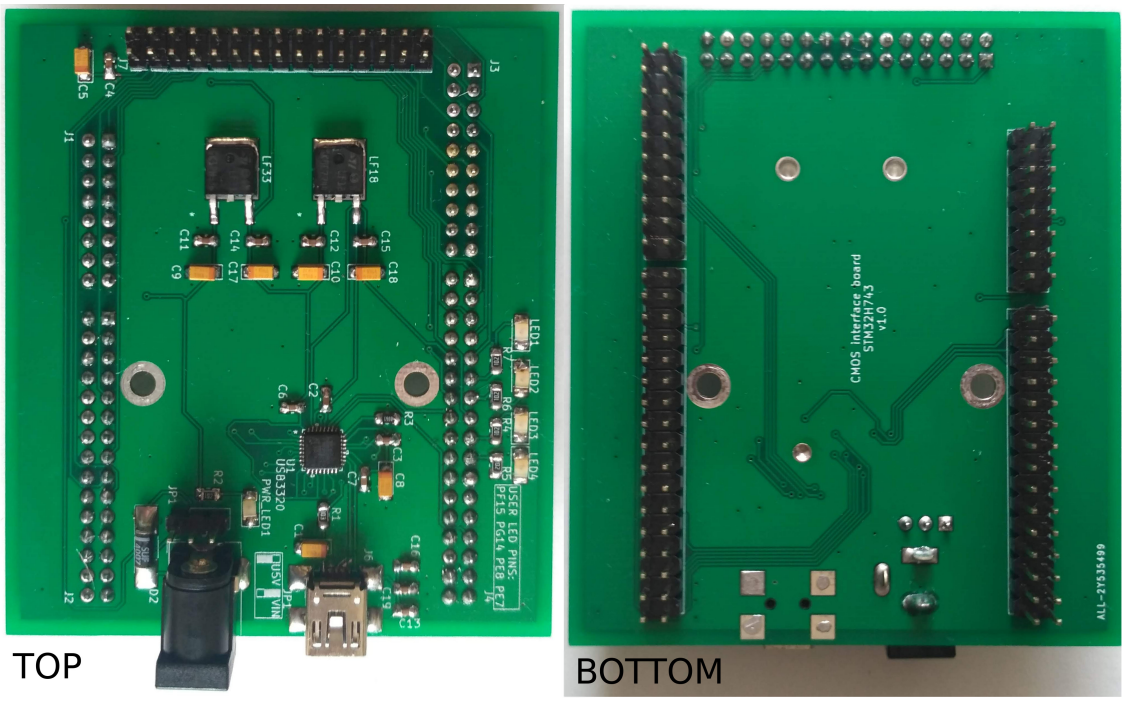


Figure C.8. Final interfacing board

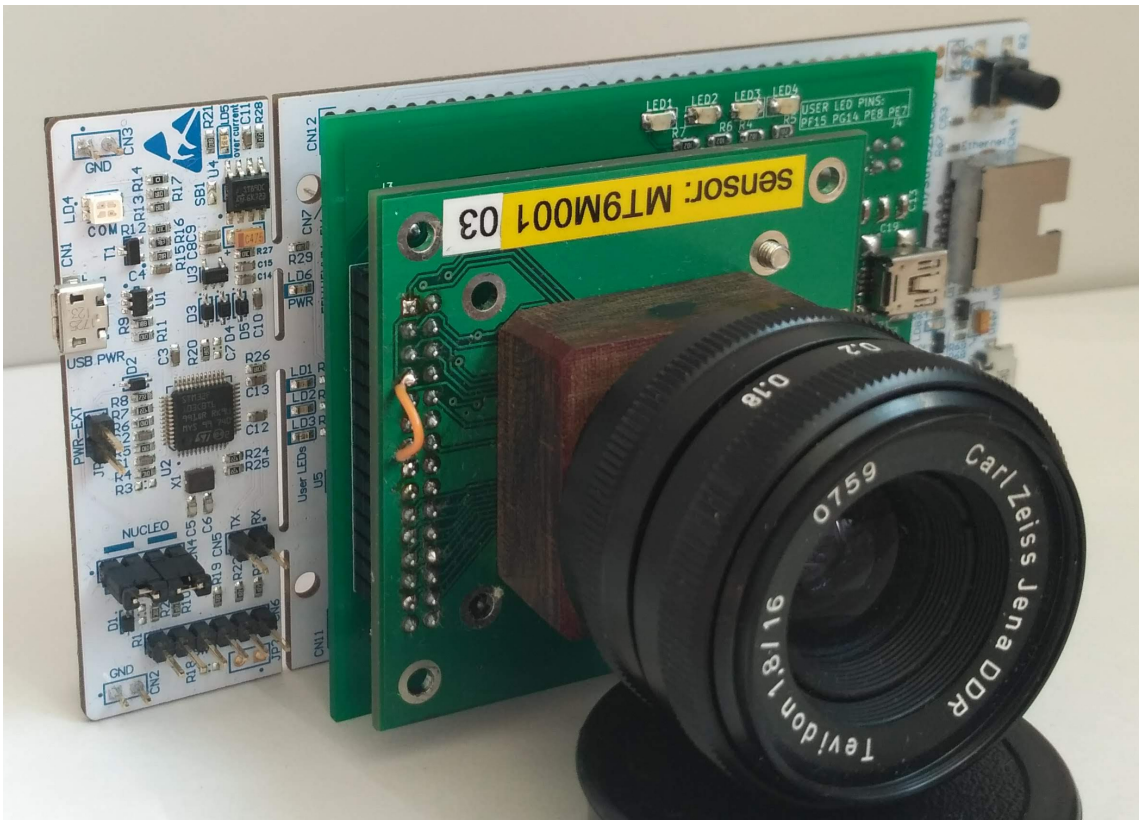


Figure C.9. Whole camera with final interfacing board

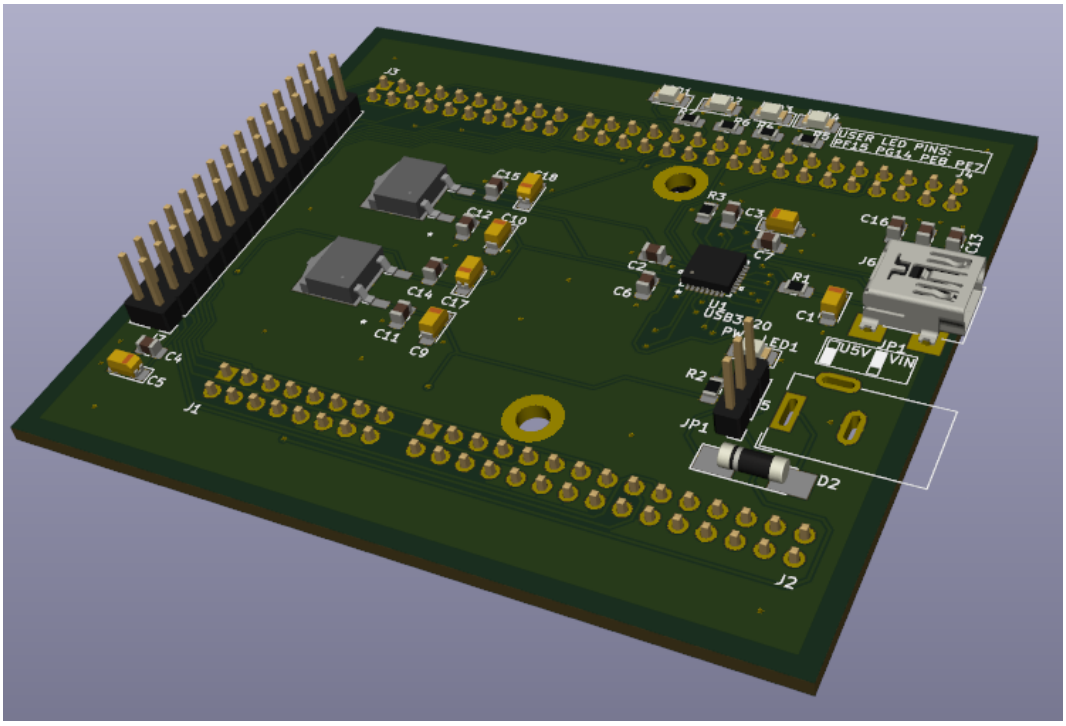


Figure C.10. 3D model of designed interfacing board PCB

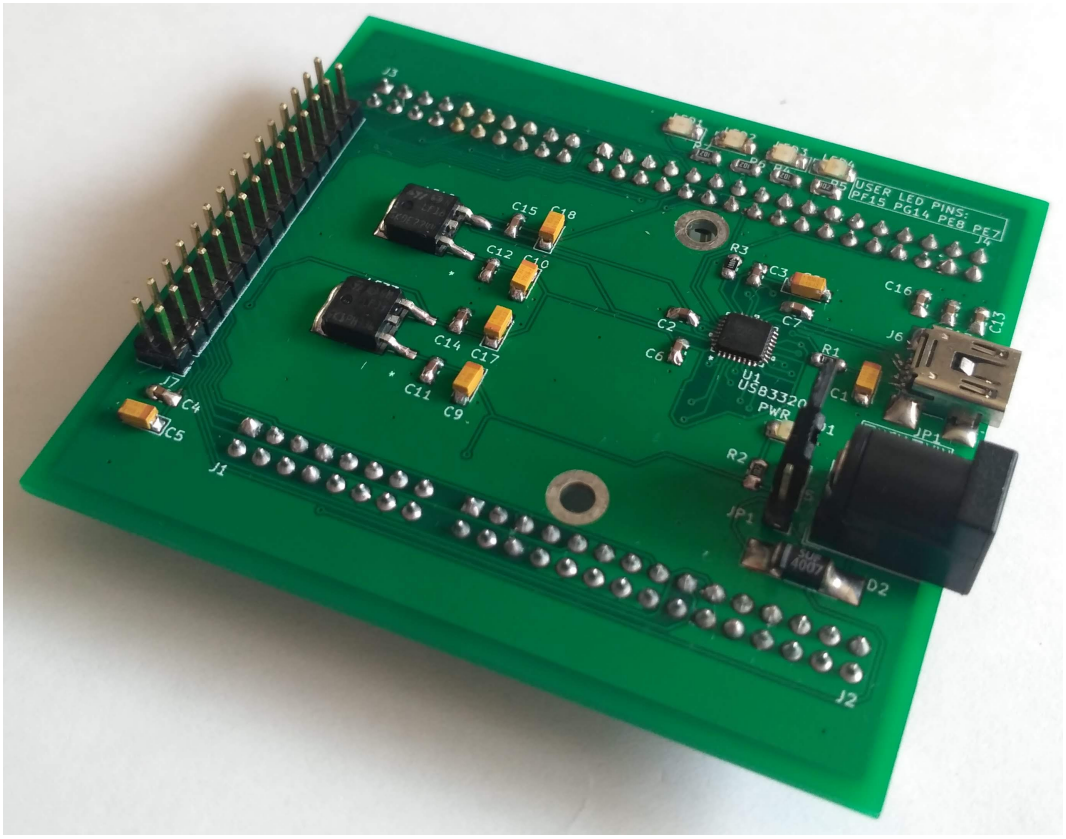


Figure C.11. Finished interfacing board PCB

Appendix D
Content of attached CD

- `Camera` — camera FW files
  - `sources` — source files of camera FW
  - `documentation` — FW code documentation
  - `deploy` — final compiled binaries of camera FW
- `PCapp` — PC application files
  - `sources` — source files of PC application
    - `windows` — for Windows version
    - `linux` — for Linux version
  - `deploy` — final compiled binaries of PC application
    - `windows` — for Windows version
    - `linux` — for Linux version
- `Thesis` — Thesis files
  - `sources` — source files of thesis in plain TEX format
    - `figs` — figures used in thesis
  - `DP_Vodsedalek_Jakub_2019.pdf` — text of thesis in PDF format
- `Docs` — used datasheets and other freely available documentation
- `InterfaceBoard` — KiCad documentation for the interfacing board
- `readme.txt` — description of CD content