CZECH TECHNICAL UNIVERSITY IN PRAGUE
Faculty of Electrical Engineering
Department of Measurement

Diploma Thesis

STM32F429 Based CMOS Camera for Teaching Purposes

Bc. Vojtěch Dokoupil

Cybernetics and Robotics
Sensors and Instrumentation

2015

Diploma Thesis Supervisor: doc. Ing. Jan Fischer, CSc.

CZECH TECHNICAL UNIVERSITY IN PRAGUE
Faculty of Electrical Engineering
Department of Measurement

DIPLOMA THESIS ASSINGMENT

Student:	**Bc. Vojtěch Dokoupil**
Study programme:	**Cybernetics and Robotics**
Specialization:	**Sensors and Instrumentation**
Title of Diploma Thesis:	**STM32F429 Based CMOS Camera for Teaching Purposes**

Guidelines:

Develop and implement CMOS camera for laboratory - measurement purposes, which is based on STM32F429I-Disco development kit. Develop interface boards for various CMOS image sensors. Create pattern program modules for image processing (thresholding, edge detection, horizon detection, bright object center finding). These modules can be used for further simple students´ tasks focusing on image sensors properties measurement. Develop demo applications with measurement and control topic using image information.

Bibliography/Sources:

[1] DS9405 - STM32F429 Datasheet, STMicroelectronics, 2014, www.st.com

[2] RM0090 reference Manual, STMicroelectronics, 2014, www.st.com

[3] Yiu, J.: Definitive Guide to ARM® Cortex®-M3 and Cortex®-M4 Processors

Diploma Thesis Supervisor:	doc. Ing. Jan Fischer, CSc.
Date of Assignment:	27th November 2014
Valid until:	31st August 2016

L.S.

doc. Ing. Jan Holub, Ph.D.
Head of Department

prof. Ing. Pavel Ripka, Ph.D.
Dean

Prague, November 27, 2014

ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
Fakulta elektrotechnická
Katedra měření

Akademický rok **2014-15**

ZADÁNÍ DIPLOMOVÉ PRÁCE

Student:	**Bc. Vojtěch Dokoupil**
Studijní program:	**Kybernetika a robotika**
Obor:	**Senzory a přístrojová technika**
Název tématu česky:	**Kamera CMOS s STM32F429 pro výuku**
Název tématu anglicky:	**STM32F429 Based CMOS Camera for Teaching Purposes**

Pokyny pro vypracování:

Navrhněte a s využitím modulu STM32F429 Disco realizujte měřicí kameru se senzorem CMOS pro účely laboratorní výuky. Pro kameru navrhněte desky s různými obrazovými senzory CMOS. Vytvořte vzorové programové moduly pro zpracování obrazu (hledání hran v obrazu, hledání středu jasných objektů, prahování, nalezení obrazu horizontu a další potřebné), z nichž bude možno vycházet při samostatných pracích studentů, řešících jednoduché úlohy měření vlastností obrazových senzorů a zpracování obrazu. Vytvořte demonstrační aplikace orientované na měření a řízení s využitím obrazové informace.

Seznam odborné literatury:

[1] DS9405 - STM32F429 Datasheet, STMicroelectronics, 2014, www.st.com

[2] RM0090 reference Manual, STMicroelectronics, 2014, www.st.com

[3] Yiu, J.: Definitive Guide to ARM® Cortex®-M3 and Cortex®-M4 Processors

Vedoucí diplomové práce:	doc. Ing. Jan Fischer, CSc.
Datum zadání diplomové práce:	27. listopadu 2014
Platnost zadání do[1]:	31. srpna 2016

ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE Fakulta elektrotechnická

L. S.

Doc. Ing. Jan Holub, Ph.D.
vedoucí katedry

Prof. Ing. Pavel Ripka, CSc.
děkan

V Praze dne 27. 11. 2014

[1] Platnost zadání je omezena na dobu tří následujících semestrů.

Statement

I declare that I wrote following thesis on my own. All the quoted and used resources are listed in the end of the document according to Methodical instruction about complying ethical principles during elaborating university theses.

This work was developed under the Laboratory of Videometry at the Department of Measurement of FEE CTU in Prague. Thesis was supervised by doc. Ing. Jan Fischer, CSc. and partly follows the outputs of MSM6840770015 - "Research of Methods and Systems for Measurement of Physical Quantities and Measured Data Processing" and other research tasks solved in Laboratory of Videometry at FEE CTU. The most useful was the knowledge base in the field of optoelectronic sensors application.

Prague, May 7, 2015 Signature:

Acknowledgment

I would like to thank to my family and friends for the support during my university studies and during creating my diploma thesis.

Last but not least I would like to thank to my diploma thesis supervisor doc. Ing. Jan Fischer, CSc. for his advice, support and help during writing my thesis.

Annotation (EN)

Diploma thesis is focused on embedded microcontroller image processing using ARM Cortex - M4 microcontroller. The aim of the thesis is to show how to connect easily various CMOS image sensors to a serially produced development kit, including both HW and SW part. The development kit is used as the main image processing unit. One chapter of the work deals with easy image processing which should be used as a base for simple student's image applications. Another part of the work is focused on low-level communication between CMOS sensors and STM32F429 MCU through standard DCMI and I^{2}C interfaces. Various readout modes of three CMOS sensor types are tested and implemented. Last but not least the demonstration PC application for access to sensor and microcontroller through USB VCP is presented. Example of practical use of the camera using triangulation is described. The developed hardware for connecting of already existing parts of sensor and microcontroller modules is shown as well. The output of the work should serve as a universal camera for teaching purposes in the Laboratory of Videometry. It is low-cost and easily reproducible. In the end an alternative mechanical solution for attaching lens to CMOS camera module using 3D printing is mentioned.

Anotace (CZ)

Diplomová práce se zabývá zpracováním obrazu na vestavném mikrořadiči s jádrem ARM Cortex - M4. Cílem je ukázat jednoduché možnosti připojení různých obrazových CMOS senzorů k sériově vyráběnému vývojovému kitu, a to jak po programové, tak po hardwarové stránce. Jedna z kapitol se zabývá implementací jednoduchého zpracování obrazu, které by mělo sloužit jako základ pro cvičení předmětů zaměřených na obrazové senzory. Další část práce je zaměřena na možnosti low-level komunikace mikroprocesoru a CMOS senzoru, postavené na paralelním rozhraní DCMI a sériovém rozhraní I^{2}C. V průběhu práce byly vyzkoušeny různé módy vyčítání obrazových dat ze tří typů CMOS senzorů, především různé módy průměrování a přeskakování obrazových řádků a sloupců. V neposlední řadě je popsána PC aplikace pro komunikaci se zvolenými senzory, umožňující i jejich konfiguraci. Součástí práce je i příkladová praktická úloha využívající triangulace. Dále je předveden propojovací modul mezi existující HW moduly senzoru a mikrořadičové části. Výsledkem práce by měla být univerzální kamera pro účely použití v Laboratoři videometrie. Řešení je nízkonákladové a jednoduše sestavitelné. Na závěr práce je zmíněn realizovaný návrh upevňovacího bloku pro objektiv. Tento blok byl vytištěn pomocí populárního 3D tisku.

Glossary

AEC	*Auto Exposure Control*
AGC	*Auto Gain Control*
ARM	*Advanced RISC Machine*
Bayer Filter	*Pattern used in CMOS sensors to gain RGB brightness values*
CG	*Centre of Gravity*
CMOS sensor	*Image sensor – Aptina MT9V032/34 or MT9M001 or MT9T031*
CTS	*Clear to Send signal, used in USART to HW flow control*
DCMI	*Digital Camera Interface – parallel data bus for camera*
Discovery kit	*STM32F4 Discovery development kit*
Disco kit	*STM32F429I-Disco development kit*
DMA	*Direct Memory Access*
Freeware	*Free Software*
GUI	*Graphical User Interface*
I^2C	*Inter Integrated Circuit*
LCD	*Liquid Crystal Display*
LSB	*Least Significant Byte*
LTDC	*LCD TFT Display Controller*
MCU	*Microcontroller Unit*
MSB	*Most Significant Byte*
PC	*Personal Computer*
PCB	*Printed Circuit Board*
ROI	*Region of Interest – part of image captured by camera*
RTS	*Request to Send, used in USART to HW flow control*
SDRAM	*Synchronous Dynamic Random Access Memory*
SPI	*Serial Peripheral Interface*
USART	*Universal Synchronous Asynchronous Receiver Transmitter*
USB	*Universal Serial Bus*
VCP	*Virtual COM Port*

Content

List of Figures

List of Tables

List of Equations

1. Introduction

The motivation for writing this thesis is usage of the microcontroller with ARM Core to prepare universal camera for teaching purposes. The camera should have two main purposes. The first one is to serve as a basic platform for student application development; the second one is to serve as a low-cost alternative to ViSor camera used for demonstrating image sensors properties in videometry classes. The camera is not planned to be used as a streaming camera, but more as a camera for snapshots and autonomous measurement based on captured image information.

The work aims to fluently build on the topic started in my bachelor thesis [12]. The microcontrollers we are interested in are often suitable for almost every embedded application purpose including image processing. There are various interface peripherals available for connecting CMOS image sensors to the embedded microcontrollers. They can be therefore compared to the signal processors in a lot of applications.

The image processing capabilities of embedded microcontrollers are supposed to be shown by development of easy demonstration applications, preparing a template for tasks being solved by students as semester projects in the future. These applications should take into account image processing operations as thresholding, edge detection or bright object detection. The triangulation sensor application with line-laser is one of the suggested options. The horizon detection application is another possible task. For these applications a stable HW solution is necessary.

The interface board between existing CMOS module boards and chosen development kit is planned to be developed. The reason for having such a board is to ensure the compactness of the solution and minimizing the possibility of HW error. Another requirement is to minimize the area of the board to reduce the cost. The power supply sufficient for empowering the CMOS sensor and possibly empowering more peripherals should be included.

In the end the audio codec available on the platform from bachelor thesis (STM32F4 Discovery) should be used to generate audio output for control applications. As a test application the audio output can be implemented for the existing demo-application performing counting dots on dices. This kit can be than used as an audio output kit for the camera with the STM32F429I-Disco kit and compensate the lack of audio codec on the newest kit in this way.

2. Input Conditions and Desired Outputs

In the introduction I outlined the main aims of my diploma thesis. Now I want to be more specific and describe them in detail. I decided to introduce current situation and expected development both in HW and SW areas. First I want to defend the choice of the development kit and compare it with the similar one used earlier in my projects. The expected benefits resulting from this choice are discussed. Further I want to describe the SW part of the project and the desired output.

2.1. Hardware and Mechanics – Current Situation and Expectations

The exact platform STM32F4 Discovery kit with STM32F407 microprocessor used in bachelor thesis was replaced by STM32F429I-Disco kit with STM32F429I microprocessor. The differences between these two microcontrollers are the possible upper limit of the system clock frequency (168 MHz vs. 180 MHz), the RAM size (192 KB vs. 256 KB) and the Flash memory size (1 MB vs. 2 MB). The main reason for changing the kit is not in the microcontrollers' parameters but in the whole development kits' parameters. The chosen kit has extra 8 MB external SDRAM and LCD display. The LCD display is unfortunately attached on the same wires as the camera interface, which makes its use quite uncomfortable. The external memory is very important because it allows also to connect sensors with higher resolutions (QVGA, SXGA) where the "on the run" processing with the bounded SRAM would be useless.

There is a standard for the CMOS sensors interface strictly defined in the Videometry Lab as well as the standard for the size of the CMOS board including pinout and the position of the header for the power and communication.

The output which I am supposed to come with is described in following lines. I have to prepare a new board to be placed between the defined boards performing the signal routing from the kit to the CMOS sensor board. This interface board should also contain power supply for creating the 3.3 V for the CMOS sensor. The 5 V need not to be stabilized but protecting diode has to be placed on the 5V board input. The additional peripherals and power signals should be placed on a defined header connector to possibly connect other devices.

There is also a voluntary task to create an alternative CS mount flange basis to attach a CS mount lens. The current version is made from the phenolic paper. The number of these blocks in the laboratory is not sufficient. I decided to try a 3D printing to solve this task.

2.2. Software – Basic Requirements

As regards the software the task is more complicated. The main task is to program the microcontroller to perform communication with various image sensors. This includes testing DMA capabilities in order of transferred data size, communicating with external data memory, etc. The codes from the standard C Libraries and STMicroelectronics® can be used. Typically this covers the lower layer of software as GPIO, Interrupt control, DCMI, DMA and USB VCP Libraries.

Image operations described in the assignment should then be implemented in the MCU. The implementation of the image processing of the captured image should than aim to manage readout and rewriting functions to access and change the data in external memory. It is supposed to implement basic functions as thresholding, edge detection and bright object localization. The necessity is to study and implement DMA transfers from various image sensors to the memory as well, which has to be configured carefully based on the resolution of the sensor and desired resolution of the upper level processing unit. The control flow on the I^2C interface should be studied carefully with the options of sensor readout. The readout modes of sensors should be tested and some of them presented.

For bitmap displaying and saving purposes any PC application should be used or implemented. The communication with the PC is not included explicitly in the task but it cannot be avoided in order to show successfully output of the inside MCU image operations.

The last thing is to prepare easy examples of applications based on adjustment of developed SW in an embedded fashion (without PC).

3. Hardware for the Camera

3.1. ST Microelectronics Development Kits

3.1.1. STM32F4 Discovery Kit

This is the first development kit that is used for minor part of this thesis. This kit is not used for image processing itself here. The main reason for presenting and using this hardware is presence of the built-in audio codec CS43L22, which can be used as an application feedback to human being; especially for applications with intelligent camera (the only limiting criterion is 1MB inside flash memory present on F407VG MCU). This is presented in great detail in chapter 6.2. The appearance of the kit can be seen in fig. 1. For more information see datasheet [4].

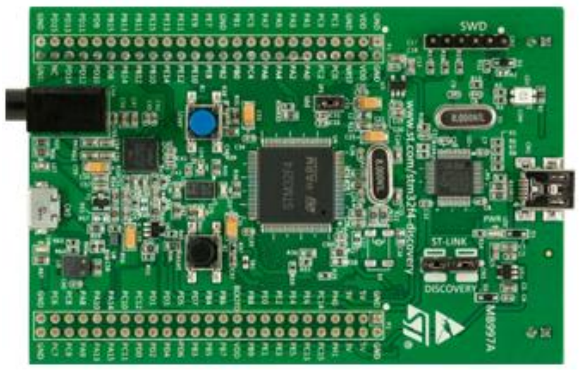


Fig. 1 - STM32F4 Discovery kit, picture from [4]

3.1.2. STM32F429I- Disco Kit

This is the main development platform of my thesis. The MCU on the kit contains DCMI and I^{2}C peripheral necessary for connecting CMOS sensors which were provided for this Diploma thesis. Moreover, there is also LCD display assembled. The only complication which blocks using CMOS sensor and kit as a real-time camera with display is the pin mapping collision between DCMI and LCD interface. The only possibility is then the time multiplexing of these two interfaces which makes the operation slow.

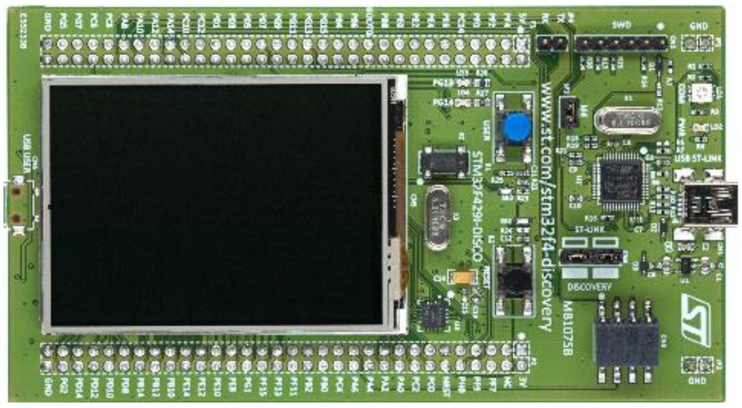


Fig. 2 - STM32F429I-Disco kit, picture from [5]

The kit has an external 64Mbit (8MB) SDRAM apart from the 256KB SRAM inside the F429I microprocessor. This allows especially in connection with higher resolution CMOS sensors to save the whole image without any need to process the image part by part. The flash memory size is 2 MB.

3.2. CMOS Sensor Module/Board

The idea was to use various CMOS sensors assembled on a standard platform (PCB developed in Videometry Lab on CTU). The header pinout of this CMOS PCB is as similar as possible for all of the sensors and thus the interface board mentioned further can be used for all of them with no need of hardware change. The only thing to be changed is the software configuration. All the grayscale sensors provide up to 10-bit data resolution. In my application I use 8-bit data resolution, but the hardware I have developed for interfacing between CMOS boards and STM32F429I-Disco kit can be easily changed to 10-bit depth data version. The RGB sensor MT9T031 has the same resolution possibilities as well.

3.2.1. Aptina MT9V034 / 032

This sensor is widely used one in the Videometry Lab. It has the resolution of 752 x 480 pixels (also known as WVGA). This sensor type is a global shutter (all photo elements are exposed at a time). Communication from/to MCU is based on DCMI (data bus) and I^{2}C (or two-wire, as the control bus). I used this sensor in my Bachelor thesis [12] as well, where additional information on this topic can be found.

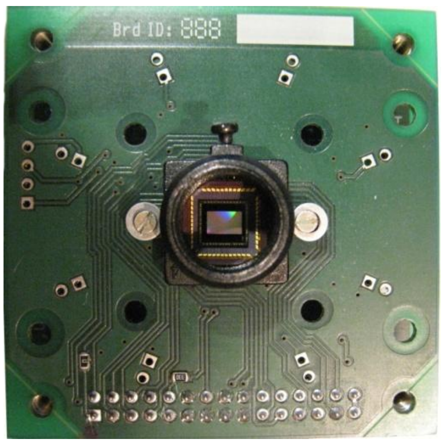


Fig. 3 - CMOS sensor MT9V034 board without lens

Regarding the lens it is difficult to find any information about. It is a small lens used for low-cost applications with no specific technical data.

The full resolution image in 8-bit resolution takes 360.96 KB which is about 4.5 % of the external SDRAM volume present on STM32F429I-Disco kit.

3.2.2. Aptina MT9M001

This is also CMOS sensor used in applications in Videometry Lab. It has the resolution of 1280 x 1024 pixels (also known as SXGA). This sensor type is a rolling shutter (photo elements are exposed in "wave" and not simultaneously). I have never worked with this sensor on Discovery kit before and so it was little bit tricky to make it work. Communication interface is except for some signals almost the same as on MT9V034, but the register addresses inside the sensor are in some cases different. Data bus is DCMI and control bus is I^{2}C.

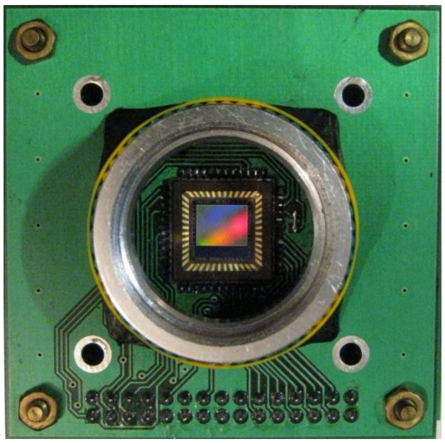


Fig. 4 - CMOS sensor MT9M001 board without lens

For the sensor MTM9001 I had to prepare a flange basis for attaching CS-mount lens as well in order to fix lens in the right position (flange focal distance is 12.5 mm - sensor has to be 12.5 mm from the flange). For this purpose I was inspired by the flange basis created from the phenolic paper block, which was fabricated before. I wanted to use 3D printer to try to prepare this block. More information on this method you can find in Annex 2 – CS Mount Flange Basis Design.

The lens used with this sensor is **Carl Zeiss Jena DDR Tevidon 1.8/16 7600**. Its focal length is 16mm. Its minimal aperture is f(focal length) divided by 1.8. The lens is attached to the camera body by the CS-mount system.

The full resolution image in 8-bit resolution takes 1.31 MB which is about 16.4 % of external SDRAM volume present on STM32F429I-Disco kit.

3.2.3. Aptina MT9T031

This is the first RGB sensor I worked with. This sensor can be assembled on the same camera board as MT9M001. However, there is a small difference in the position of the photoactive area in relation to mechanical package of the sensor. The active imaging area center is 0.23 mm shifted from the position it has on MT9M001 sensor. You can see it in fig. 5. For testing purposes I decided to ignore this difference (it is possible that CMOS sensor inaccuracy originated from soldering can be higher than this difference).

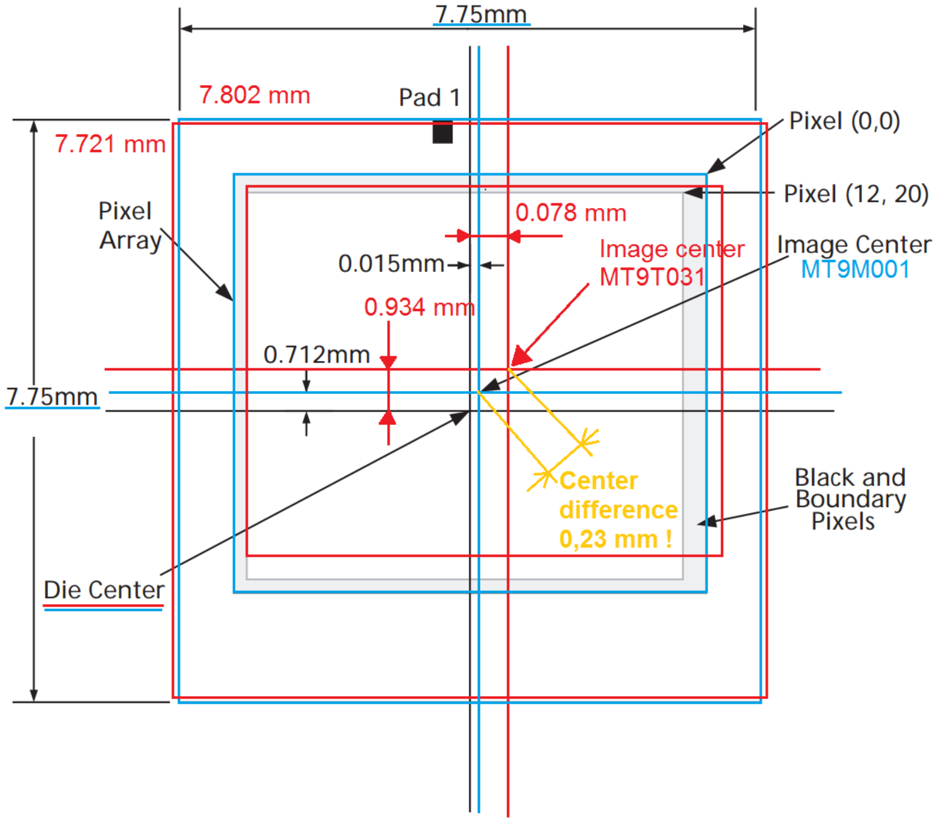


Fig. 5 - MT9T031 and MT9M001 image centre difference

The resolution of this RGB sensor is 2048 x 1536 (also known as QXGA) pixels. But there is one important difference contrary to grayscale sensors. The resolution is defined as the active image pixel count in horizontal and vertical directions. These sensors use Bayer filters and thus one final colour image point constitutes from 4 active pixels as can be seen in fig. 24. The final RGB image resolution is then 1024 x 768 image points (in case of using my easy debayerization method implemented in chapter 4.2.4)!

For attaching lens to this sensor I used the same flange basis as described in 3.2.2. The lens is also the same.

Communication between the sensor and MCU is again very similar to the one on MT9M001. Several differences are described in further sections.

The full resolution data of raw image in 8-bit resolution takes 3.15 MB which is about 39 % of external SDRAM volume present on STM32F429I-Disco kit.

For displaying data in PC (for sending to PC) or on LCD display it is necessary to transform raw data to bmp RGB888 format. The final image is then 3/4 size of the raw data size (in case of using my easy debayerization method implemented in chapter 4.2.4). For the transformation I need to have original raw data and converted RGB888 image data at a time and thus approximately 69 % of external memory is blocked while MCU is performing this application. The transform method will be described in section 4.2.3.

3.2.4. CMOS Camera Sizes Comparison

Here I decided to mention few words in order to compare sensors and their module boards in connection to lens size. The visual comparison you can see in fig. 6. The bigger lens is used in connection with MT9M001 and MT9T031 sensors. The smaller lens is used for MT9V032/34 sensors.

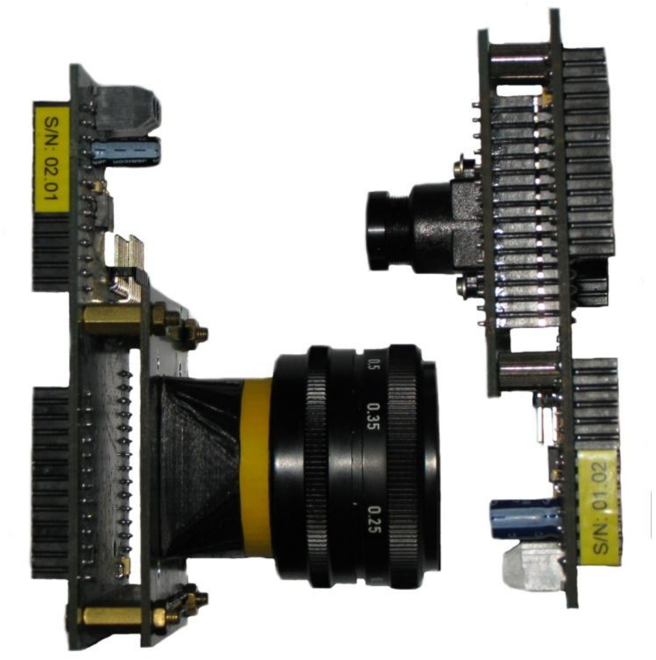


Fig. 6 - Comparison: CMOS camera module boards with lenses on CONN.BOARDs F429

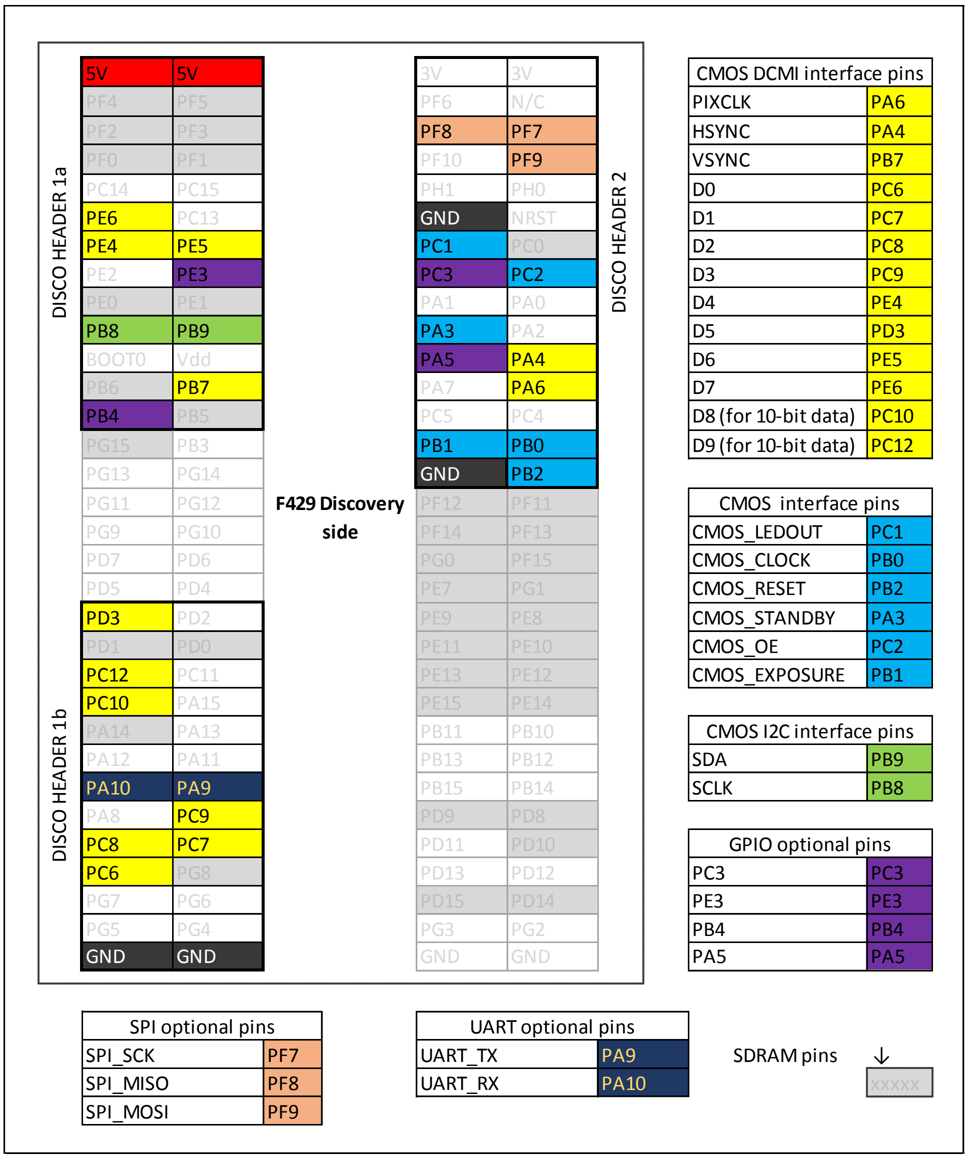
3.3. CONN.BOARD F429 between STM32F429I-Disco Kit and CMOS Board

As soon as new kit (STM32F429I-Disco) is used I need to fabricate new version of interface board similar to the one for the F4 Discovery kit (developed in my bachelor thesis). The next section gives overview on the boards, for schematics and blueprints look into additional documents (Annex 1 – Assembly Instruction for the CONN.BOARD F429 Version1/2).

3.3.1. CONN.BOARD F429 Pinout

Based on the previous work concerning connecting this kit with a sensor module, I reused Bc. Daniel Toms pinout presented in his bachelor thesis [9] to draw the board. The pinout with some changes is described in tab. 1 .

Tab. 1 - CONN.BOARD F429 pinout



F429 Discovery side

DISCO HEADER 1a / 1b			DISCO HEADER 2	
DISCO HEADER 1a	5V	5V	3V	3V
	PF4	PF5	PF6	N/C
	PF2	PF3	PF8	PF7
	PF0	PF1	PF10	PF9
	PC14	PC15	PH1	PH0
	PE6	PC13	GND	NRST
	PE4	PE5	PC1	PC0
	PE2	PE3	PC3	PC2
	PE0	PE1	PA1	PA0
	PB8	PB9	PA3	PA2
	BOOT0	Vdd	PA5	PA4
	PB6	PB7	PA7	PA6
	PB4	PB5	PC5	PC4
	PG15	PB3	PB1	PB0
	PG13	PG14	GND	PB2
	PG11	PG12	PF12	PF11
	PG9	PG10	PF14	PF13
	PD7	PD6	PG0	PF15
	PD5	PD4	PE7	PG1
DISCO HEADER 1b	PD3	PD2	PE9	PE8
	PD1	PD0	PE11	PE10
	PC12	PC11	PE13	PE12
	PC10	PA15	PE15	PE14
	PA14	PA13	PB11	PB10
	PA12	PA11	PB13	PB12
	PA10	PA9	PB15	PB14
	PA8	PC9	PD9	PD8
	PC8	PC7	PD11	PD10
	PC6	PG8	PD13	PD12
	PG7	PG6	PD15	PD14
	PG5	PG4	PG3	PG2
	GND	GND	GND	GND

CMOS DCMI interface pins	
PIXCLK	PA6
HSYNC	PA4
VSYNC	PB7
D0	PC6
D1	PC7
D2	PC8
D3	PC9
D4	PE4
D5	PD3
D6	PE5
D7	PE6
D8 (for 10-bit data)	PC10
D9 (for 10-bit data)	PC12

CMOS interface pins	
CMOS_LEDOUT	PC1
CMOS_CLOCK	PB0
CMOS_RESET	PB2
CMOS_STANDBY	PA3
CMOS_OE	PC2
CMOS_EXPOSURE	PB1

CMOS I2C interface pins	
SDA	PB9
SCLK	PB8

GPIO optional pins	
PC3	PC3
PE3	PE3
PB4	PB4
PA5	PA5

SPI optional pins	
SPI_SCK	PF7
SPI_MISO	PF8
SPI_MOSI	PF9

UART optional pins	
UART_TX	PA9
UART_RX	PA10

SDRAM pins ↓ xxxxx

There was also a requirement on possibility to change the data width from eight to ten bits easily. This is realized through zero ohm resistors, which can be assembled in case of 10-bit version of data bus. This change moves all data signals about two bits higher to reserve place for two new LSB bits which provide higher data resolution. This change can be seen in tab. 2.

Tab. 2 - CMOS header description in view of 8/10 bit data bus operation

29	27	25	23	21	19	17	15	13	11	9	7	5	3	1
3V	D6 (D8)	D4(D6)	D2(D4)	D0(D2)	NC (D0)	G	G	G	G	VS	STB	EX	SDA	LD
3V	D7(D9)	D5(D7)	D3(D5)	D1(D3)	NC (D1)	G	G	CLK	G	PX	HS	RE	SCL	OE
30	28	26	24	22	20	18	16	14	12	10	8	6	4	2

names in brackets are for 10-bit data version
pin 1 has square PAD

CMOS HEADER from CMOS side

The hardware change takes place near the CMOS header (connector), which position depends on the version of CONN.BOARD F429. The placement of the zero ohm resistors (for the data bus width change) on the first version differs from the placement on the second version of CONN.BOARD F429. Fig. 7 shows how to change the DCMI bus width in the first version of the CONN.BOARD F429.

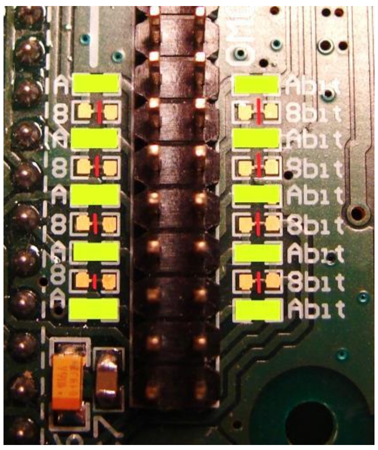


Fig. 7 - Hardware change of DCMI bus width

3.3.2. CONN.BOARD F429 Version 1

In this version the CMOS sensor has the same orientation as the LCD display on STM32F429I-Disco board. This is convenient in case the user wants to use time multiplexing between reading from CMOS and sending image to display. The orientation of the picture is then "camera wise". It means that the displayed picture has the same orientation as the captured one. However it can also be used in another application with no necessity of displaying CMOS data. Fig. 8 shows the 3D model from Altium® software used for development of this board. The board in the final setup can be seen in fig. 9.

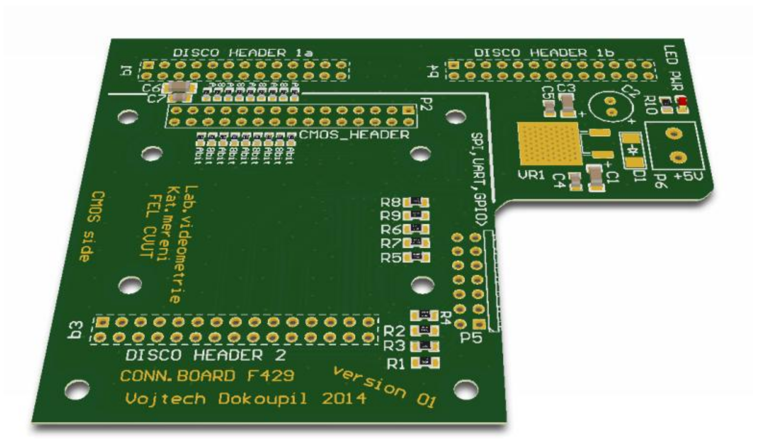


Fig. 8 - Render of the first version of CONN.BOARD F429

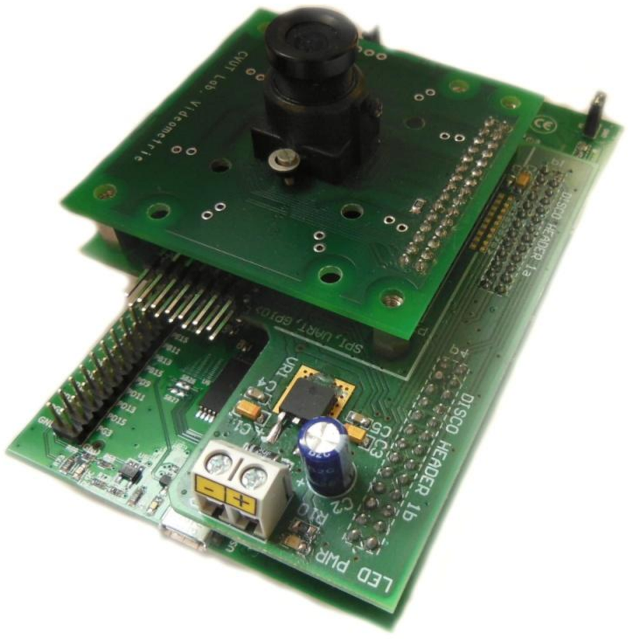


Fig. 9 - CONN.BOARD F429 for CMOS module and STM32F429I-Disco kit connection

3.3.3. CONN.BOARD F429 Version 2 (90° rotated)

The second CONN.BOARD F429 version is focused on applications with no necessity to display the data in the orientation of CMOS sensor; there is probably no need to display them at all. The orientation of the sensor is then only a question of mechanical design of the system. The sensor is placed similarly to the placement of the sensor on the interface board developed in my bachelor thesis [12]. Fig. 10 shows the render of the second version of CONN.BOARD F429.

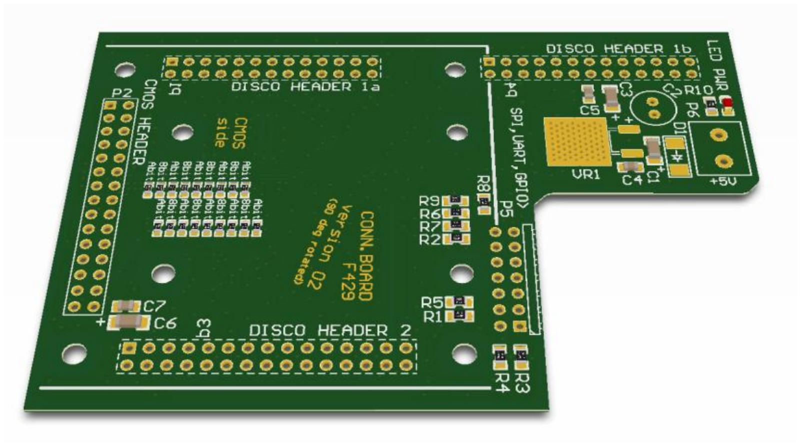


Fig. 10 - Render of the second version of CONN.BOARD F429

4. Software for the Camera

Keil® Microvision IDE is used for all the MCU programs. The code is written in C programming language.

4.1. Low-level MCU Software – Interfacing CMOS Image Sensors

This chapter depicts a bunch of information connected to interfacing between F429I microprocessor and few CMOS sensors which can be used in my application. Those sensors are all from Aptina® Company (nowadays owned by OnSemi®). That signalizes better interchangeability of various sensors, but still there are differences in the interface. The other topic covered in this chapter concerns MCU capabilities of DMA direct data transfer from DCMI Peripheral to SDRAM external memory.

4.1.1. Master Clock Frequency for the CMOS Sensor

The possible master clock frequency range for the CMOS sensor depends on the selected type. The sensor MT9V034/32 (described in chapter 3.2.1) clock frequency should be between up to 27 MHz. On the other hand the sensor MT9M001 can be clocked up to 48 MHz. With current hardware the optimal clock frequency for the both sensors was set on 21 MHz. We don't need to challenge the data rate, because due to the image processing the camera is not to be real-time anyway. The clock signal (currently connected MT9M001) is shown on oscilloscopic screen in fig. 11. There is a special case when the clock frequency has to be decreased in order to successfully process image data actually received.

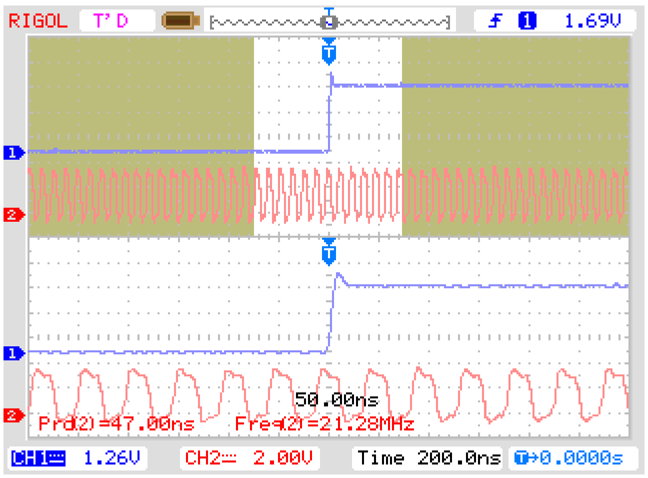


Fig. 11 - Master Clock Frequency to CMOS - 21 MHz

I added some more information to describe signals in fig. 11. The clock signal is the red one and the main time base of the oscilloscope is 200 ns, the detail time base is then 50 ns. The blue signal used as a trigger source is frame valid signal. The trigger is set on the transition from zero to one. Frame valid signal is set to be active through the whole image readout.

4.1.2. Standard DMA Transfer from DCMI Peripheral to SDRAM

In my application there is one DMA channel set to transfer data from DCMI to SDRAM. There are few restrictions on the transaction, for instance the buffer size of DMA transfer etc. The standard image sizes to be transferred are shown in tab. 4 and in tab. 5. The maximal value counter of DMA is bounded by 16-bit counter maximum – 0xFFFF. This value is maximal no matter what is the data-width. Due to this fact the transfer can contain most data possible in case of **WORD** data width (4 bytes). The maximal buffer size can be seen in tab. 3.

Tab. 3 - DMA buffer capabilities

DMA mode	Counter max [hex]	Counter max [dec]	Max. data width[B]	Max. data in one "cycle" [B]
Single buffer	FFFF	65535	4	**262140**
Double buffer	FFFF	65535	4	**524280**

The single buffer would be sufficient for one transfer cycle of the data up to 262 140 bytes, but in case the data length is bigger (higher resolutions), we have to find another way, how to transfer the data to SDRAM and not overwrite the first part of transferred data by the next DMA buffer. It is necessary to mention that I need whole image data in the memory after one “snapshot” on DCMI interface.

In order to increase possible size of image data transferred in one “cycle” - meaning DMA cycle not buffer cycle - I can use double buffer mode. This means that DMA controller has two pointers to memory. In the first buffer cycle it writes the data to the first address and if there are still input data (from DCMI peripheral) it transfers another buffer (the timer is cleared) to the second memory location. This can effectively double the transferred data size. This means that each image data part is saved at distinct addresses. The whole image can be displayed from the memory at a time. For the sensor MT9V032/34 this mode is sufficient even for the highest resolution (WVGA) in 8-bit data width. The final settings for the MT9V032/34 in view of buffer modes and sizes are described in tab. 4.

Tab. 4 - DMA buffer settings for basic resolutions on MT9V032/34

Resolution	Width[px]	Height[px]	Depth[B]	Size[B]	Used DMA mode	buffers [B]
standard	752	480	1	360960	double buffer	2 x 180480
1/2 binned	376	240	1	90240	double buffer	2 x 163840
1/4 binned	188	120	1	22560	double buffer	2 x 40960

The other situation occurs with the MT9M001 CMOS sensor. The lower resolutions are processed in the same way as described for the previous sensor. The problem appears for data size bigger than 524 280 Bytes. Then the same problem occurs as in the case with rewriting single buffer, only with the difference that we are rewriting it also in a double buffer mode. It concerns the image in the highest resolution and the ROIs bigger than that value. Tab. 5 shows basic resolutions of the CMOS sensor MT9M001.

Tab. 5 - DMA buffer settings for basic resolutions on MT9M001

Resolution	Width[px]	Height[px]	Depth[B]	Size[B]	Used DMA mode	buffers [B]
standard	1280	1024	1	1310720	double buffer	**described in section 4.1.3**
2 skipped	640	512	1	327680	double buffer	2 x 163840
4 skipped	320	256	1	81920	double buffer	2 x 40960
8 skipped	160	128	1	20480	double buffer	2 x 10240

For better understanding I have provided few oscilloscopic screens for the standard double-buffer mode readout (the double buffer is written only once). The proposed solution of the problematic full resolution readout will be described in the next section 4.1.3.

Fig. 12 shows the 160 x 128 resolution image readout (8 skipped rows/columns mode on CMOS sensor). The blue signal shows FRAME VALID signal which signalizes the readout of the image. This FRAME VALID signal is being changed periodically and it is derived from the master clock signal (chapter 4.1.1). Master clock signal is for disposal all the time as well as this signal. **The snapshot mode (one frame capture) is then set on DCMI level**. If there is a signal to capture image, the DCMI triggers the closer FRAME VALID signal and starts DMA transfer, otherwise the DMA is deactivated. The red signal shows to which of the two memory locations of double - buffer mode it is currently written. The signal makes sense only in the interval where FRAME VALID is high. Low signal level means saving to *location0* and high to *location1* (starting half image size later). The operation of DMA readout for this resolution takes **14.4 milliseconds**.

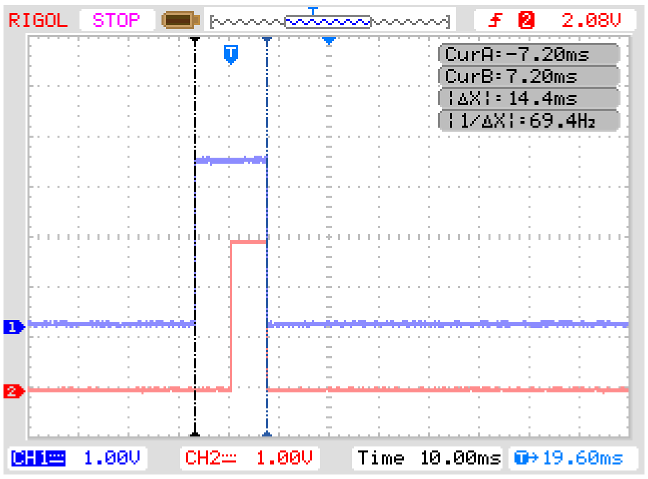


Fig. 12 - MT9M001 DMA 160 x 128 readout (8 rows/columns skip mode)

Fig. 13 shows the 320 x 256 resolution image readout (4 skipped rows/columns mode on CMOS sensor). The signal notation is still the same as in the previous figure. The operation of DMA readout for this resolution takes **31.2 milliseconds**.

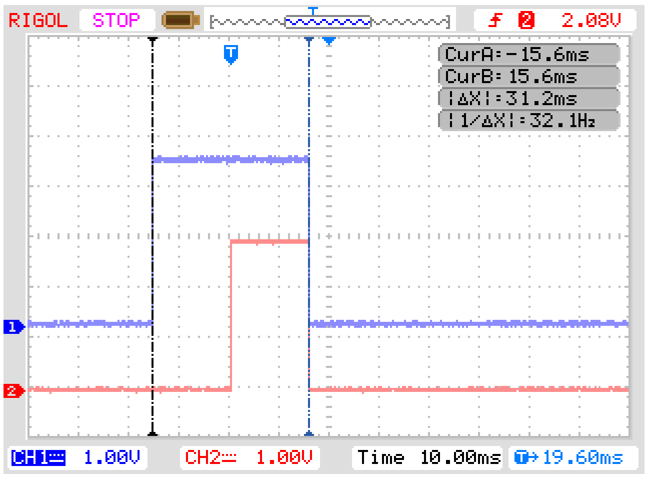


Fig. 13 - MT9M001 DMA 320 x 256 readout (4 rows/columns skip mode)

Fig. 14 shows the 640 x 512 resolution image readout (2 skipped rows/columns mode on CMOS sensor). The signal notation is still the same as in the previous figure. The operation of DMA readout for this resolution takes **70 milliseconds**.

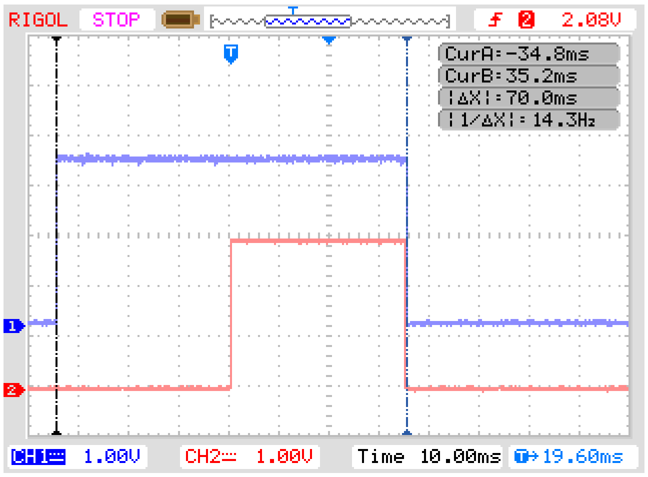


Fig. 14 - MT9M001 DMA 640 x 512 readout (2 rows/columns skip mode)

When I connected the third sensor to my universal camera unit, I have already known about the DMA buffer size problems and so the solution was programmed in a very similar way as the solution for the second grayscale sensor (M9TM001). If the image size grows over certain level (described in tab. 3) the DMA mode is changed to the one described in 4.1.3.2. The example of various image sizes for some of resolution modes on MT9T031 CMOS sensor are described in tab. 6.

Tab. 6 - DMA buffer settings for basic resolutions on MT9T031

Resolution	Width[px]	Height[px]	Depth[B]	Size[B]	Used DMA mode	buffers [B]
standard	2048	1536	1	3145728	double buffer	**described in section 4.1.3.2**
2 binned	1024	768	1	786432	double buffer	**described in section 6.1.3.2**
4 skipped	512	384	1	81920	double buffer	2 x 98304
8 skipped	256	192	1	49152	double buffer	2 x 24576

4.1.3. Bigger Data-length DMA Transfer from DCMI Peripheral to SDRAM

As I presented before data longer than 524 280 Bytes has to be proceeded differently in the scope of DMA transfer. Chronologically I tried two solutions (the second one is probably much better but this is step-by-step documentation of my development) to the problem of insufficient buffer size even if I am using double-buffer DMA mode.

4.1.3.1. Received Buffers Moving by MCU Software

The first straight-forward solution which I describe is based on received data buffer transfer in SDRAM while the second buffer in double-buffer mode is transferred via DMA. In order to capture the whole image I have to transfer the data before the particular DMA buffer destination is used again.

If I copy the data using pure software (pointer incrementing) solution, it is not possible to transfer the data in time at the nominal (21 MHz) master clock frequency to CMOS sensor. The influence on the time limit for data transfer has also the length of the transferred buffer and CMOS horizontal blanking.

Horizontal blanking is set in the beginning of each CMOS read configuration and says how many ticks of pixel clock will be extra added to the line length. These “dummy” ticks are prolonging readout time of each line. The DMA buffer size is thus the same as with shorter horizontal blanking but it takes longer time to transfer it (due to waiting on blanking pixel ticks). I have then more time to process the other buffer in the second location.

Experimentally I found that I need to decrease Master clock frequency to 8.4 MHz. That can be seen in fig. 15. The clock frequency is approximately three times lower.

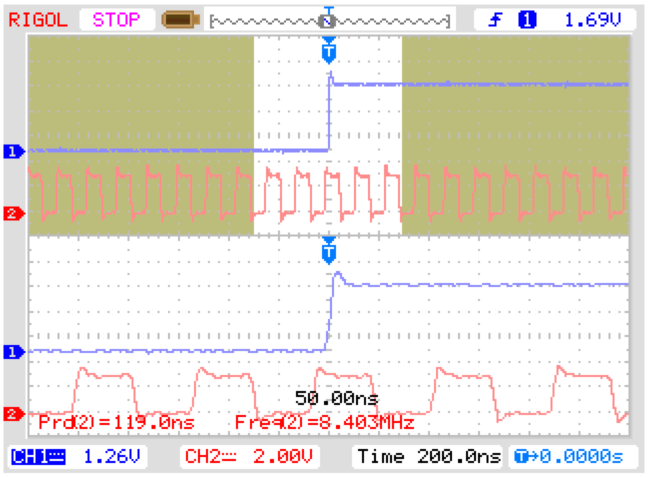


Fig. 15 - Master Clock Frequency to CMOS (MT9M001) - 8 MHz

Now I want to show the influence of faster master clock (21 MHz) signal than possible (not enough time to move the data in the memory before new DMA buffer reception). Fig. 16 shows signals connected to the image readout, which is corrupted by wrong timing (and possibly also wrong horizontal blanking setting). The blue signal shows distorted movement of transferred buffers in SDRAM. The red signal shows the location where the current buffer is being saved. Low signal level means saving to *location0* whilst the high signal level saving to *location1*.The readout is faster than movement of data in the memory. Moreover, horizontal blanking is minimal and thus there is not enough time to move buffers from the DMA destinations; it can be observed on the blue signal. The signal is not set low before new buffer readout (to the same location) is started. This results in saving corrupted image to the memory, as can be seen in fig. 17

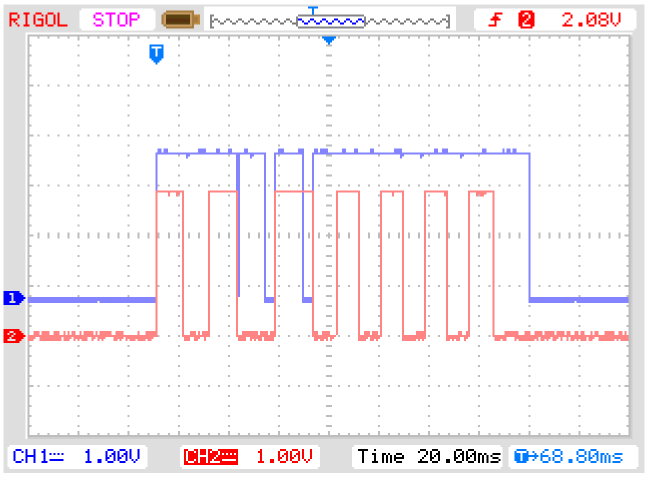


Fig. 16 - MT9M001 full resolution image – signals of corrupted readout

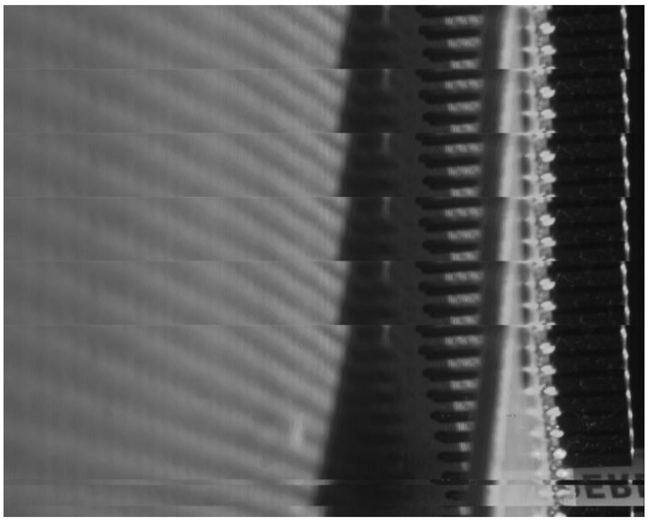


Fig. 17 - MT9M001 full resolution image - displaying corrupted data

As I tried, only the change of horizontal blanking is not sufficient to get enough time to move received data in SDRAM by MCU software means. The master clock has to be lower as I suggested in the beginning of this chapter. Horizontal blanking can be then used for adjustment of the line readout time.

Fig. 18 shows correct readout timing. The image in full resolution was divided into 16 parts which are alternately saved to *location0* and *location1.* The readout of the whole image takes **430 ms** and is bounded by the cursors in the oscilloscopic screen. The red signal within this interval shows location where it is currently being saved. High level of the blue signal represents moving previously captured part while the new part is loaded to the second memory location. In the moment the transaction is pointed to the first location again, the previous data is further correctly transferred into SDRAM.

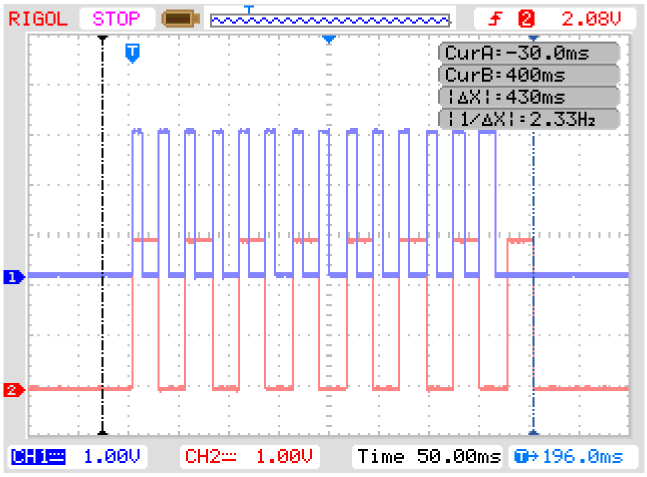


Fig. 18 - MT9M001 full resolution image - correct data readout

4.1.3.2. Received Buffer Destination Changing while DMA Running

The second solution comes right after the first one. After deeper study of DMA on F429I microcontroller I found out that it is possible to change the destination of DMA double buffer memory locations. As I mentioned before, double-buffer DMA mode uses *location0* and *location1* pointers to the memory, which are periodically alternated. The point is in the fact that when the DMA is writing to one of the locations possible, the other pointer can be changed. It can happen dynamically - without a need of reconfiguring DMA controller. The solution is much easier than the previous one and it does not request master clock adjustment. The sensor can run on the 21 MHz for the whole time being as presented in chapter 4.1.1.

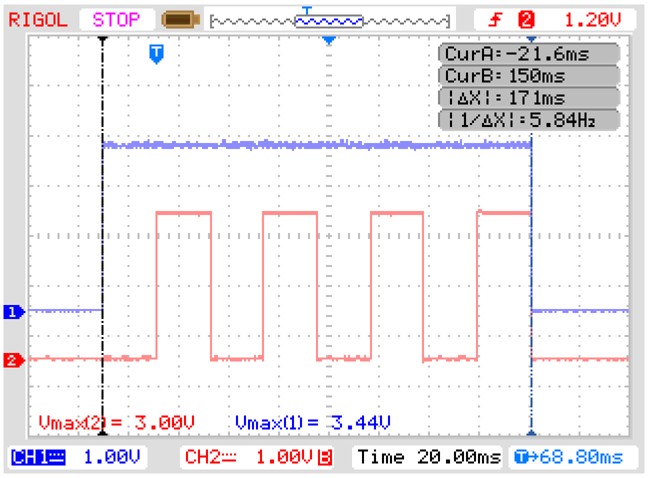


Fig. 19 - MT9M001 full resolution image - correct data readout - second method

Fig. 19 shows the timing of described method. Blue signal is in this case FRAME VALID signal and the red signal symbolizes changing the DMA memory location (the pointer where the data is written). The reading of the whole image data to SDRAM takes **171 ms**.

Fig. 20 shows the exact addresses of DMA pointers in the memory. It can be also seen that now I used division of image data only to 8 parts not to 16 as in 4.1.3.1. The reason is that the pointer change is very short operation and decreasing buffer length has no impact on performance of this method. This is the lowest possible number of parts if the image data in the maximal resolution are divided to equal parts.

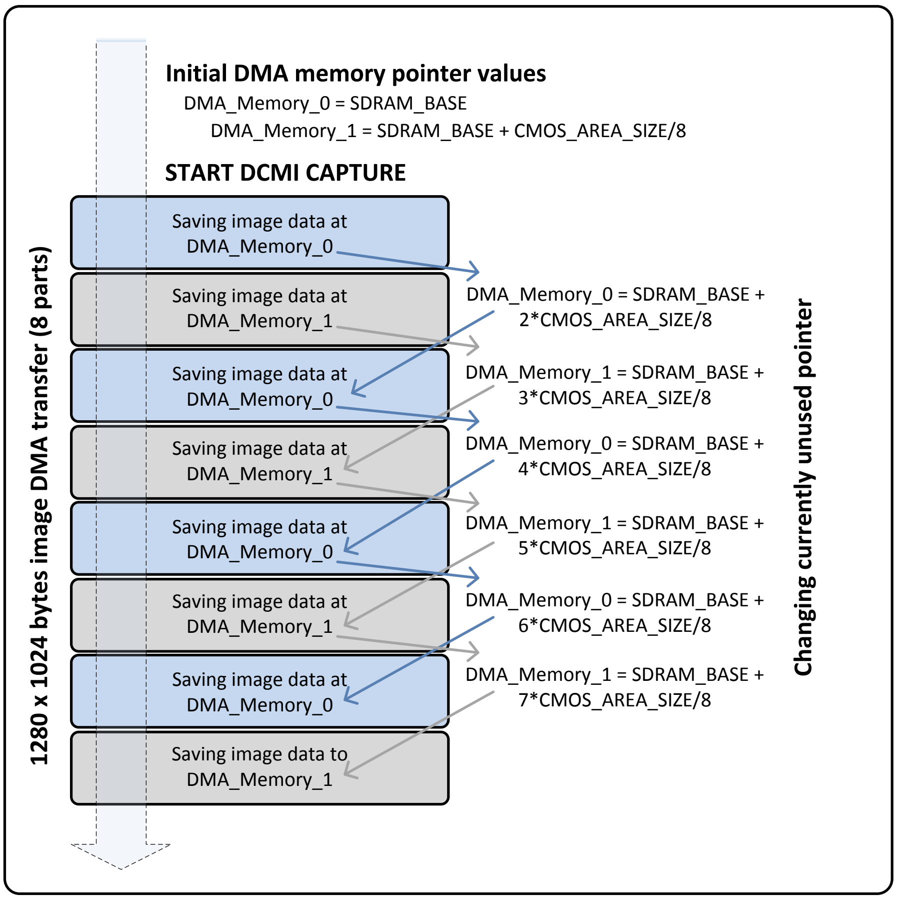

Initial DMA memory pointer values
DMA_Memory_0 = SDRAM_BASE
DMA_Memory_1 = SDRAM_BASE + CMOS_AREA_SIZE/8
START DCMI CAPTURE
Saving image data at DMA_Memory_0
Saving image data at DMA_Memory_1
Saving image data at DMA_Memory_0
Saving image data at DMA_Memory_1
Saving image data at DMA_Memory_0
Saving image data at DMA_Memory_1
Saving image data at DMA_Memory_0
Saving image data to DMA_Memory_1
DMA_Memory_0 = SDRAM_BASE + 2*CMOS_AREA_SIZE/8
DMA_Memory_1 = SDRAM_BASE + 3*CMOS_AREA_SIZE/8
DMA_Memory_0 = SDRAM_BASE + 4*CMOS_AREA_SIZE/8
DMA_Memory_1 = SDRAM_BASE + 5*CMOS_AREA_SIZE/8
DMA_Memory_0 = SDRAM_BASE + 6*CMOS_AREA_SIZE/8
DMA_Memory_1 = SDRAM_BASE + 7*CMOS_AREA_SIZE/8
1280 x 1024 bytes image DMA transfer (8 parts)
Changing currently unused pointer

Fig. 20 - MT9M001 full resolution image - memory addressing diagram using DMA

This mode was also used for reading the data out of the third sensor - MT9T031. Due to the RGB raw data size which is 2048 x 1536 pixels, the division of the transfer to four double-buffer cycles is not sufficient. The double-buffer count has to be bigger in order to get the data into one transfer size which is bounded by condition described in section 4.1.2 .

4.2. Image Processing MCU Functions

This chapter is focused on low-level image processing based on local image information. The simpler filters are just pixel-wise ones and the other filters work with 3 x 3 pixel neighbourhood. The chapter also includes specific image processing methods necessary for obtaining desired RGB data format from RAW data. For an easy lookup into the code attached on CD, this chapter summarizes the library **STM32F429_image_process.c.**

4.2.1. Image Inversion and Thresholding Filter

These two functions are just pixel-wise. The necessary operations for the desired functionality are: reading certain amount of data from external SDRAM, pixel operation, writing data back to external SDRAM into the same location to save memory volume. For both of these functions I have selected SDRAM "read and write" buffer size of one sensor line. The principle can be easily followed in fig. 21 .

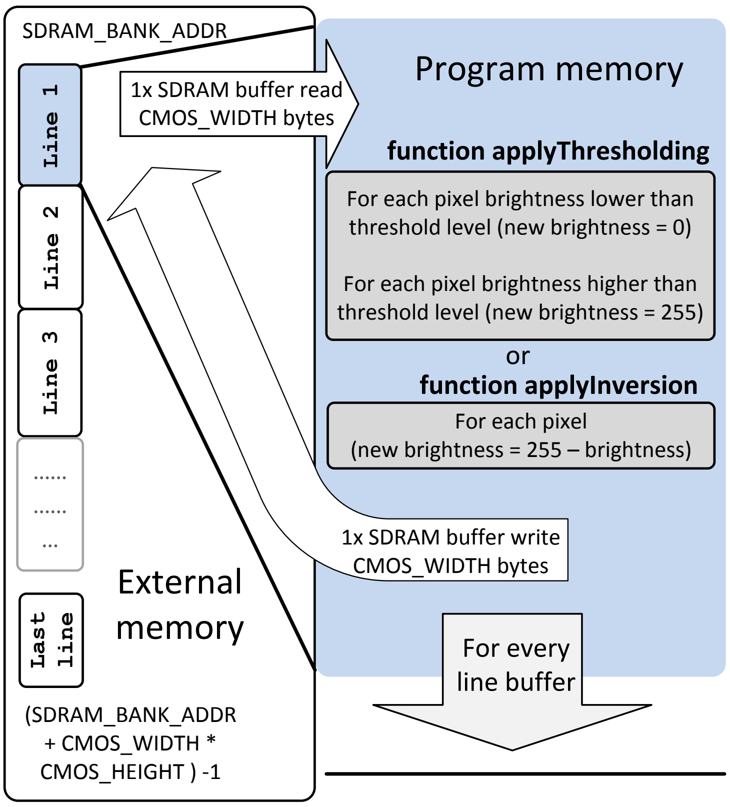

SDRAM_BANK_ADDR
Line 1
Line 2
Line 3
......
......
...
Last line
External memory
(SDRAM_BANK_ADDR + CMOS_WIDTH * CMOS_HEIGHT) -1
1x SDRAM buffer read CMOS_WIDTH bytes
Program memory
function applyThresholding
For each pixel brightness lower than threshold level (new brightness = 0)
For each pixel brightness higher than threshold level (new brightness = 255)
or
function applyInversion
For each pixel (new brightness = 255 – brightness)
1x SDRAM buffer write CMOS_WIDTH bytes
For every line buffer

Fig. 21 - Memory flow for pixel - wise image functions

These functions were originally written for grayscale image. In case of **applyInversion** function this is not an issue. The pixel brightness inversion will be successfully proceeded with no code change also on RGB data. The function **applyThresholding** currently changes pixel brightness according to one threshold level value. This results in the fact that all the red, green and blue values are threshold with the same value. It will be better to set three thresholds for the colours. This can be easily changed in the code, but at this time I am not focusing on this.

4.2.2. Image Prewitt Edge Filter – 4-directional

This filter is the first more complicated one. It demands knowledge of more than one pixel brightness. This fact in connection with the fact that I need the processed data at the same external memory location as the original one leads to the necessity of storing more data (then actual line data) in the program memory for processing purposes. The principle can be seen in fig. 22.

The Prewitt filter in my configuration demands the 3 x 3 pixel neighbourhood. The filter can consist of one directional filter up to 4-directional filter. This is possible due to the combination of four Prewitt masks which can be seen in equ. 1. The binary flags A, B, C and D, which are loaded from PC application and set through **applyPrewittFilter** function parameter, influence the mask sensitivity on particular edge directions. If the flag is zero, the sensitivity on the particular direction is not set and the Prewitt "sub-mask" is cleared.

$$A \cdot \begin{pmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{pmatrix} \cdot \frac{1}{3} + B \cdot \begin{pmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{pmatrix} \cdot \frac{1}{3} +$$

$$+ C \cdot \begin{pmatrix} 0 & -1 & -1 \\ 1 & 0 & -1 \\ 1 & 1 & 0 \end{pmatrix} \cdot \frac{1}{3} + D \cdot \begin{pmatrix} 0 & 1 & 1 \\ -1 & 0 & 1 \\ -1 & -1 & 0 \end{pmatrix} \cdot \frac{1}{3}$$

Equ. 1 - Synthesis of the complete Prewitt mask from direction sensitive masks

Now I look deeper inside the method in view of the memory operations. I need to get original image data for the previous and following image line if I want to successfully compute the filtered value for pixels from the actual line. Thus I have to store the filtered line data to the end of the next "line-filter cycle" and copy them to the original memory location just in the moment the original data on this line is no more needed for the next "line-filter cycle". This can be followed again via the diagram in fig. 22. In fact I need two line-buffers. One for storing filtered values of previous line and one for saving actually being filtered pixel values.

These two buffers are periodically switched. For reading from external SDRAM only three 3-byte buffers are used. This is data for one pixel 3 x 3 neighbourhood.

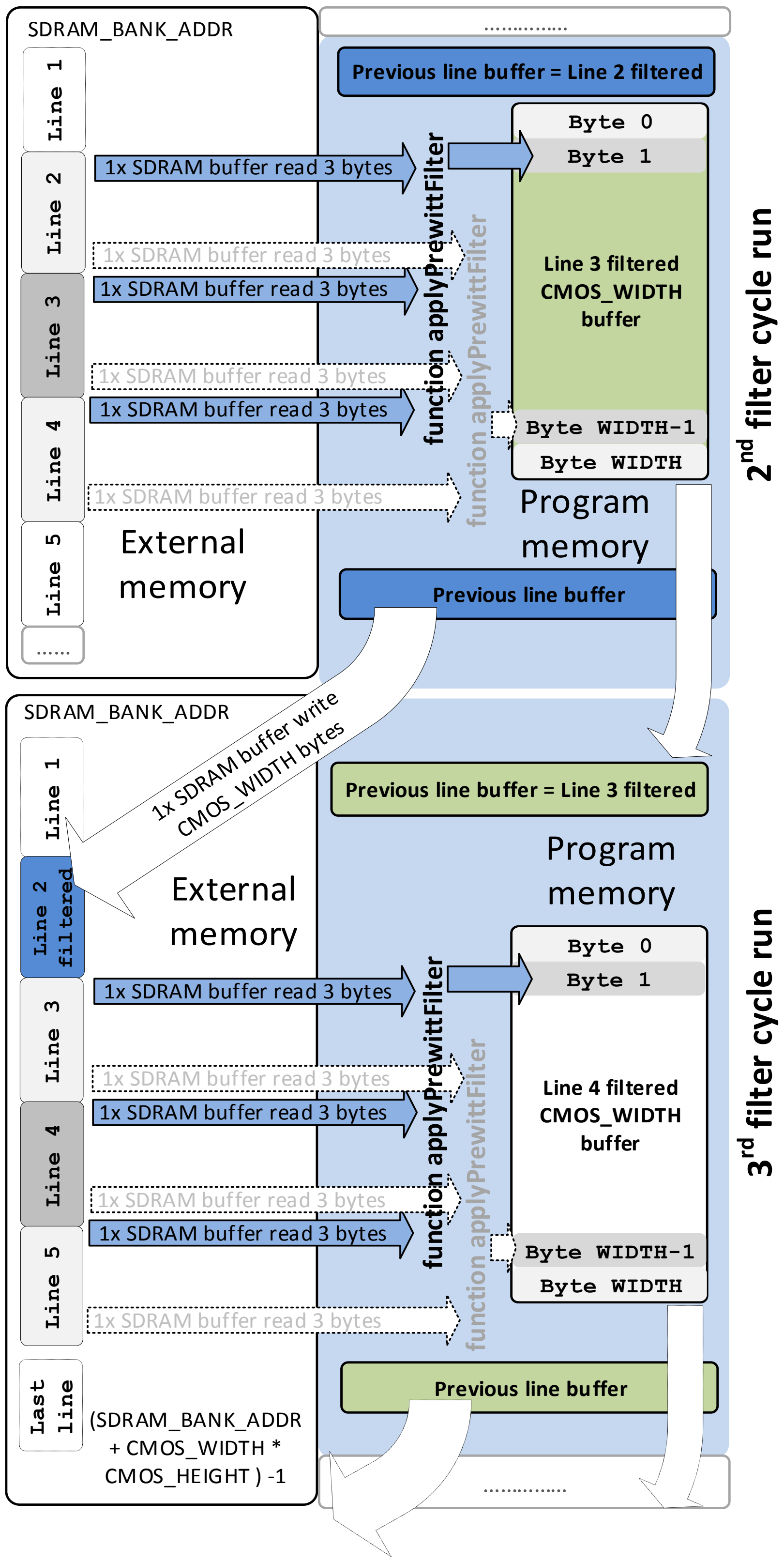

SDRAM_BANK_ADDR
Line 1
Line 2
Line 3
Line 4
Line 5
......
External memory
1x SDRAM buffer read 3 bytes
function applyPrewittFilter
Previous line buffer = Line 2 filtered
Byte 0
Byte 1
Line 3 filtered CMOS_WIDTH buffer
Byte WIDTH-1
Byte WIDTH
Program memory
Previous line buffer
2nd filter cycle run
1x SDRAM buffer write CMOS_WIDTH bytes
Line 2 filtered
Previous line buffer = Line 3 filtered
Line 4 filtered CMOS_WIDTH buffer
3rd filter cycle run
Last line
(SDRAM_BANK_ADDR + CMOS_WIDTH * CMOS_HEIGHT) -1

Fig. 22 - Memory flow for 3 x 3 pixel neighbourhood-wise functions

4.2.3. Integral Image Filter for Finding Bright Object in Image

The function **applyIntegratingImage** proceeds computation of the image centre of gravity. The algorithm computes sum of brightness's of image rows to one column vector (size is the height of the CMOS area) and analogically brightness's of image columns to one row vector (size is the width of the CMOS area). Then just the coordinates of maximums of these vectors are computed and the position of the image GC is clear. Equ. 2 shows the principle of the computation.

$$y_{GC} = index\ of\left(\max_{y}\left(\sum_{x=0}^{image\_width} brightness_{xy}\right)_{y=0}^{image\_height}\right)$$

$$x_{GC} = index\ of\left(\max_{x}\left(\sum_{y=0}^{image\_height} brightness_{xy}\right)_{x=0}^{image\_width}\right)$$

Equ. 2 - CG coordinates computation

The data in SDRAM is saved one row after another, thus computation of y CG coordinate is not a problem for any resolution. Computation of x CG coordinate in the same run is trickier, because it is not possible to access the data blocks via the same buffers as y CG uses and I need big static array in program SRAM to store the sub-sums. This is the reason I implemented this function only for lower resolutions. The possibility how to utilize this function also for higher resolutions is to use two runs (meaning accessing image data twice) - one for the computation of y CG and one for the computation of x CG.

This function is planned to be used for regulation purposes and thus I am not counting with the need of higher resolutions processing. The demonstration output of this function in the user application can be seen in fig. 23. The resolution of the image is 320 x 256 pixels and this image is received from the camera with attached MT9M001 sensor. Due to the principle of the method it can be used not only on grayscale but also on the binary image (threshold output). In the first case we can get better sensitivity but we can have problems with the other bright objects. If we use correctly the threshold binary image this problem with the other less bright objects is eliminated but the sensitivity on the detected object is lower because it is “saturated”.

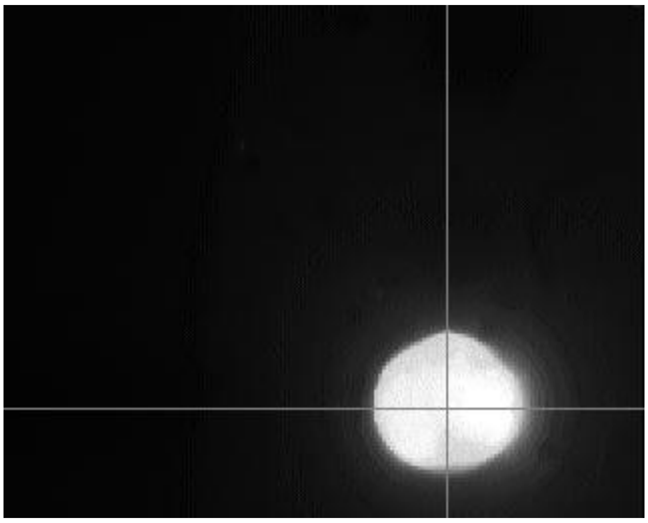


Fig. 23 - Image output of the bright object centre finding using CG of the image

4.2.4. RGB Transformation from Raw Data to RGB888 Format

Not only filtering functions but also image format transition function is included in the image processing library. With the decision to connect also the RGB sensor I have to deal with the file format which I am going to use for sending image to PC or possible display on LCD on-board. The transfer speed parameter is not an issue, because I am not building a streaming camera but rather a snapshot one. In order not to lose the quality I decided to use RGB888 format. This means that each colour has 8-bit brightness. Each image point (I am not using pixel) has then 3-times more memory than grayscale image point.

The function just takes data from one position in external SDRAM, where it has been transferred from CMOS Image sensor by DMA, and saves them right after the original data. As I said the data is saved in another location. The image filtering functions adjust the original image data in some way and I want them to be in the same memory location after the filtering. On the other hand, this transition of RGB data is based on necessity to create bitmap in a standardized format. The image operations can be still performed on raw RGB data. The whole process of transition from sensor to RGB888 format is shown in fig. 24. For the simplicity any additional filtering is not included in this memory diagram.

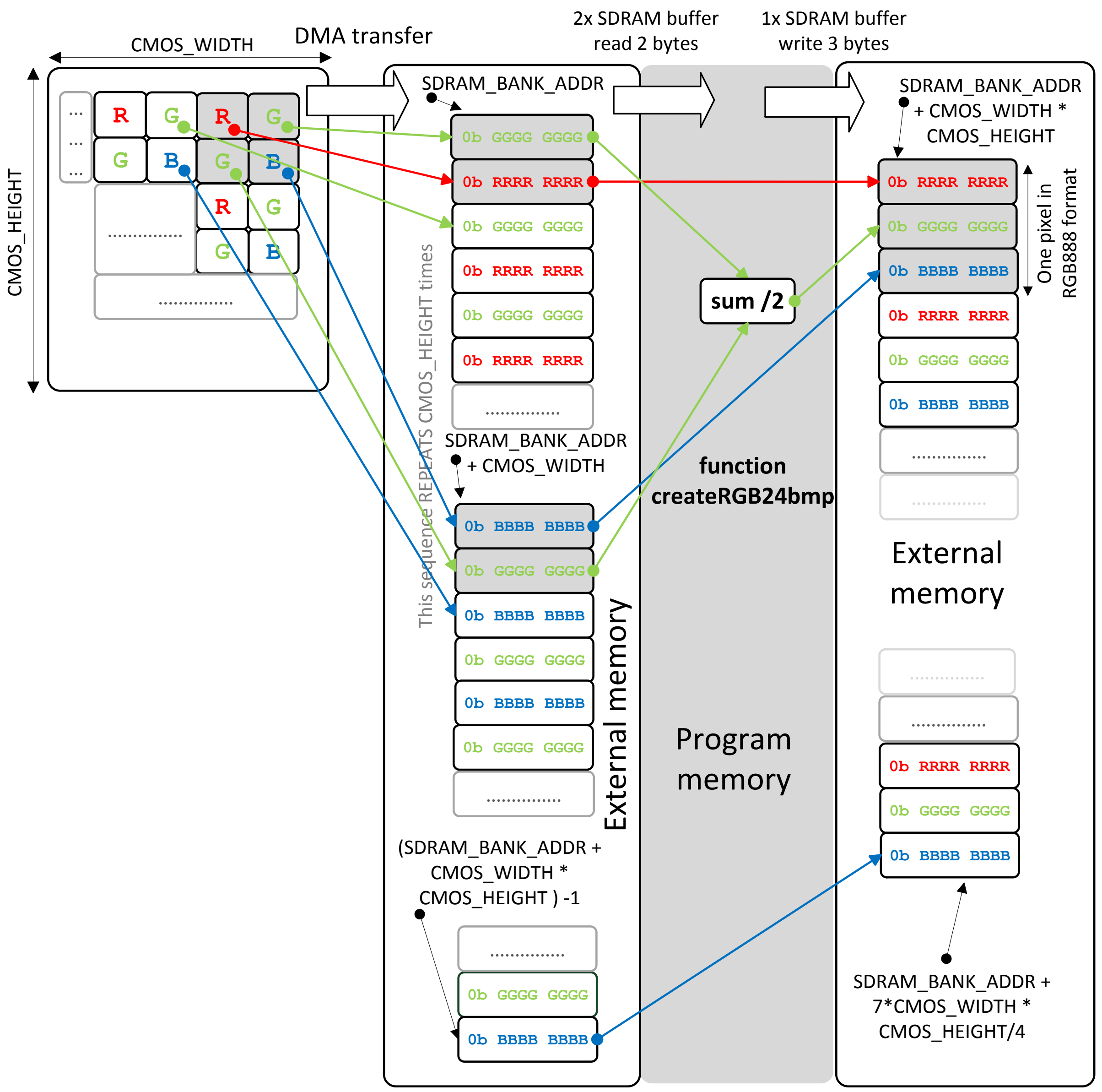

CMOS_WIDTH
DMA transfer
CMOS_HEIGHT
2x SDRAM buffer read 2 bytes
1x SDRAM buffer write 3 bytes
SDRAM_BANK_ADDR
SDRAM_BANK_ADDR + CMOS_WIDTH * CMOS_HEIGHT
R
G
B
0b GGGG GGGG
0b RRRR RRRR
0b BBBB BBBB
This sequence REPEATS CMOS_HEIGHT times
SDRAM_BANK_ADDR + CMOS_WIDTH
(SDRAM_BANK_ADDR + CMOS_WIDTH * CMOS_HEIGHT) -1
sum /2
function createRGB24bmp
Program memory
External memory
One pixel in RGB888 format
SDRAM_BANK_ADDR + 7*CMOS_WIDTH * CMOS_HEIGHT/4

Fig. 24 - Raw RGB image to RGB888 bitmap format transition

4.3. Communication and PC Application - Upper Level Software

4.3.1. USB VCP Communication Protocol between MCU and C# Application

The communication is performed over a known interface with known USB VCP Library for MCU available on STMicroelectronics® website. On the PC side there is used standard **serialPort** class in C# Application. The reason for using the VCP instead of USB is based on the knowledge from previous works in Videometry Lab on this platform. There is no necessity to use special drivers for the USB device and the common driver for VCP by ST is already present. For people with no knowledge of USB should be the VCP implementation easier readable than that of USB. For me it was also easier to implement this because I worked with this interface in my bachelor thesis.

Tab. 7 - VCP Communication messages in direction PC -> CMOS camera (MCU)

	mode selected in PC application	**streaming command handling**	**snapshot command**	**Sensor identification**	**image filters selected in PC application**	**request CMOS register readout**	**write value to CMOS register**
Buffer[0]	'm'	's'	'n'	'i'	'f'	'q'	'u'
Buffer[1]	Selected sensor = 0..1..2	Start Stop flag = 0..1			Inversion flag = 0..1	Register number to read = 0x00-0xFF	Register number to write = 0x00..0xFF
Buffer[2]	Desired Resolution = 0..1..2..(3)				Prewitt flag = 0..1		Value to write MSB = 0x00-0xFF
Buffer[3]	Desired Readout Orientation = 0..1..2..3				Prewitt type = 0-15		Value to write LSB = 0x00-0xFF
Buffer[4]	Desired Averaging = 0..2..4..8..16				Threshold flag = 0..1		
Buffer[5]	Reset registers to default = 0..1				Threshold level = 0-255		
Buffer[6]	RAW data readout flag = 0..1				Integral image flag = 0..1		

Tab. 7 shows used messages for communication from PC application to CMOS camera. There are messages connected mainly to get readout parameters which are transferred via MCU and I^{2}C peripheral to the CMOS chip itself. There are missing 'b' and 'c' commands in the table which were used to trigger communication of one image data part and whole image transfer trigger. I have a problem to communicate in the direction from PC while the big image data buffers are transferred into the other direction. That is why in the end only HW flow control on VCP was used (CTS and RTS signals).

Tab. 8 shows messages (buffers) sent in the direction from the camera to the PC application. There are few parameter messages, including identification message and CMOS register access answer messages. The main part of the communication in this direction is image data transfer. The image data is transferred one part after another in buffers of 2000 (data) + 6 (header) bytes.

Tab. 8 - VCP Communication messages in direction CMOS camera (MCU) -> PC

	Image part message block with header	**answer to sensor identification**	**answer to request CMOS register readout**	**answer to write value to CMOS register**
Buffer[0]	**Image frame buffer ID LSB = 0x00-0xFF**	'i'	'q'	'u'
Buffer[1]	**Image frame buffer ID MSB = 0x00-0xFF**	Register 0x00 value LSB = 0x24..0x13.. ..0x21..0x31	Read value from reg MSB = 0x00-0xFF	Successful writing to reg flag = 0..1
Buffer[2]	**Whole image width LSB = 0x00-0xFF**	Register 0x00 value MSB = 0x13..0x16.. ..0x84	Read value from reg LSB = 0x00-0xFF	
Buffer[3]	**Whole image width MSB = 0x00-0xFF**			
Buffer[4]	**Whole image height LSB = 0x00-0xFF**			
Buffer[5]	**Whole image height MSB = 0x00-0xFF**			
Buffer[6]	Image data			
...	Image data			
Buffer[2005] (maximal)	Image data			

The header includes 16-bit ID of frame (the range of this ID numbers for particular image can be computed by dividing image data size in bytes by transfer buffer size - 2000 bytes). The first buffer has ID 0x0000 then all the buffers have the same size (2006 bytes). The last buffer can be shorter and its ID is CMOS_AREA_SIZE/2000.

The other parameters in the header are whole image size - 16-bit height and 16-bit width. This data is sent in each frame although this is not necessary (the size of the bitmap is set right when the first image data buffer is captured).

Based on the IDs the image parts are formed into a bitmap in PC application and are fixed to the position. In case of any data lost the rest of the image can be still successfully built.

4.3.2. Connecting Various Sensors - Automatic identification

I have the task to prepare the camera enabling to connect more types of image sensors. In the end I worked with three types of image sensors. In order to make my example - PC application for the sensors data access - more user-friendly I implemented automatic identification of the sensor type. In the moment of connecting camera to VCP, the identification according to fig. 25 is performed and the appropriate tab in the application is activated and selected.

For this purposes the messages with 'i' identification (first byte in buffer) from tab. 8 and tab. 7 are used. The value of the register 0x00 of the sensor is inside the camera read through I^2C and from camera sent to PC to select the right sensor tab in the application.

It is not possible to set all the sensors on the same address by HW means. That is why we cannot use only one I^2C address for the identification process and so we have to use at least two addresses. In the identification phase the address 0xB8 is set and if no answer is received in certain time then it is switched to 0xBA as is clear from the diagram (fig. 25).

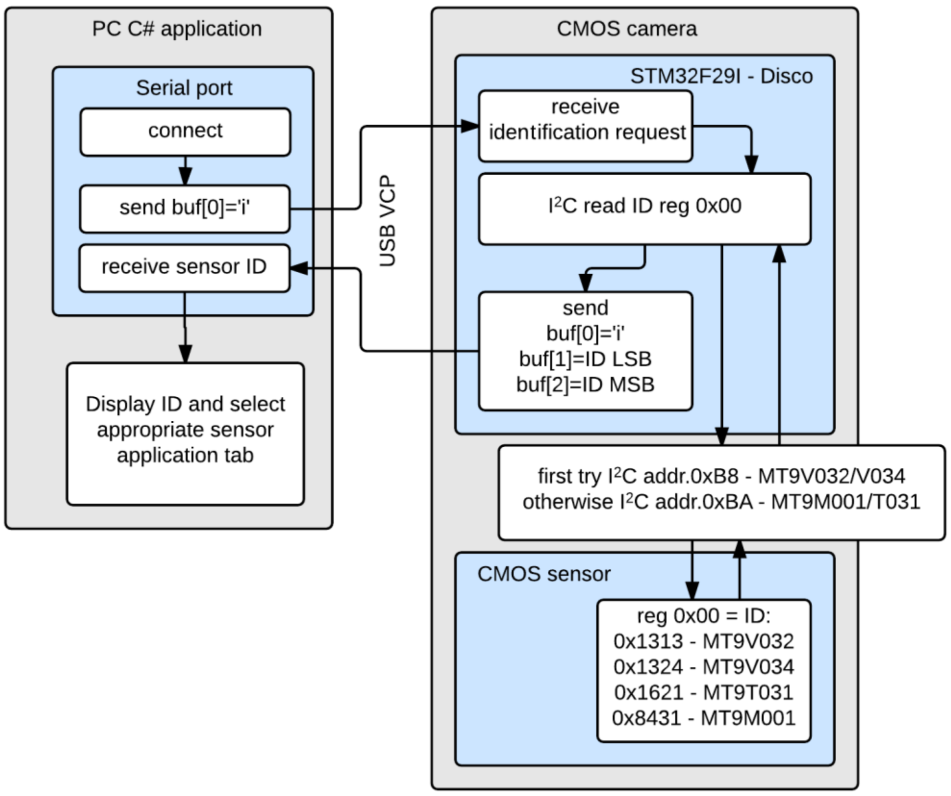

PC C# application
Serial port
connect
send buf[0]='i'
receive sensor ID
Display ID and select appropriate sensor application tab
USB VCP
CMOS camera
STM32F29I - Disco
receive identification request
I²C read ID reg 0x00
send buf[0]='i' buf[1]=ID LSB buf[2]=ID MSB
first try I²C addr.0xB8 - MT9V032/V034 otherwise I²C addr.0xBA - MT9M001/T031
CMOS sensor
reg 0x00 = ID: 0x1313 - MT9V032 0x1324 - MT9V034 0x1621 - MT9T031 0x8431 - MT9M001

Fig. 25 - Automatic CMOS sensor identification diagram

4.3.3. Main MCU Loop

The concept of MCU application is following. After the identification (includes CMOS I^{2}C initialization on MCU side) described in fig. 25 the CMOS basic DCMI and DMA initialization is proceeded and enabled. From this moment on the sensor master clock generated by MCU and distributed to the CMOS sensor is continuously running (to the moment user requests register restart which causes CMOS sensor reinitialization).

Fig. 26 describes main loop in a simplified way. The first data parameters are coming included in the message identified by **'m' character** (see tab. 7). This data is parsed and parsed output is processed through I^{2}C to sensor registers. Then four possible commands can be received. User can request register access (**'u'** writing or **'q'** - reading). Following this command the answer is generated ("register access" branch in fig. 26) and sent to application through VCP. The other possibly requested commands are used for triggering image capture and image send operations. It can be either the **'s'** identifier for starting or ending stream sequence or **'n'** one for snapshot data request.

There is one additional issue not clear enough from the diagram. The DMA has to be reconfigured in each image capture run in case the resolution (transferred data length) is changed. The CMOS sensor is running continuously and the triggering mechanism for image capture is controlled on DCMI level not on the sensor level (which would be also possible). When the image capture is requested the data is captured to SDRAM and after that the filtering operations described in section 4.2. are carried out. Details about the filtering program module are shown in fig. 27. Last operation is sending data to PC (see section 4.3.1).

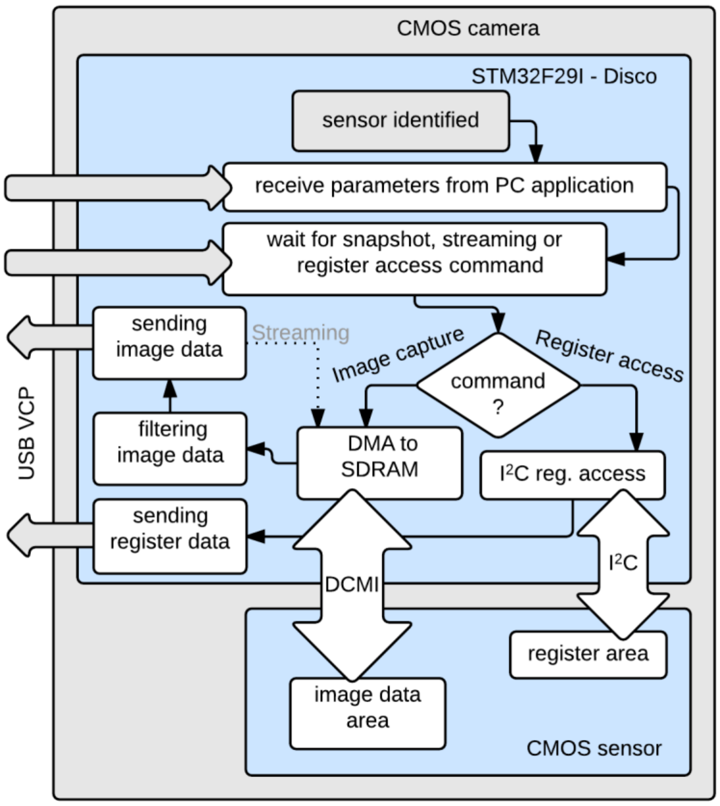


Fig. 26 - MCU main loop logic

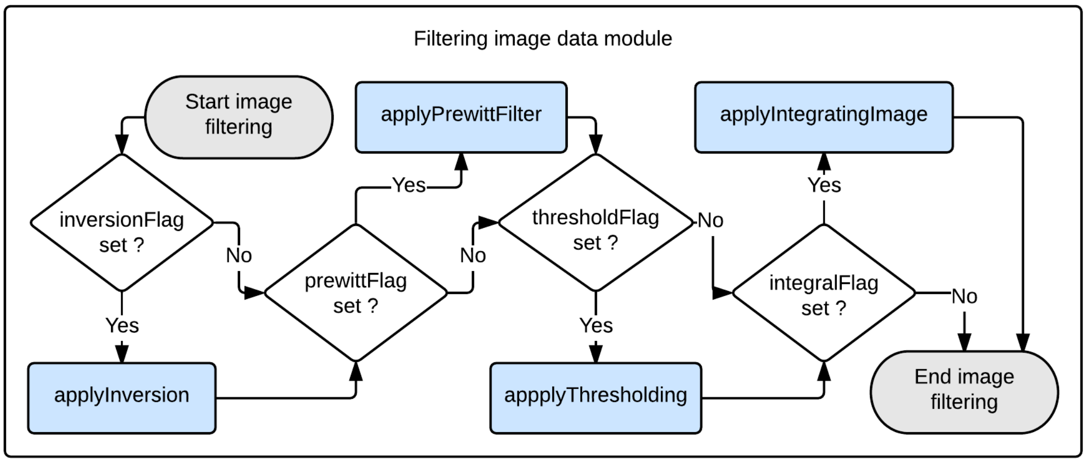
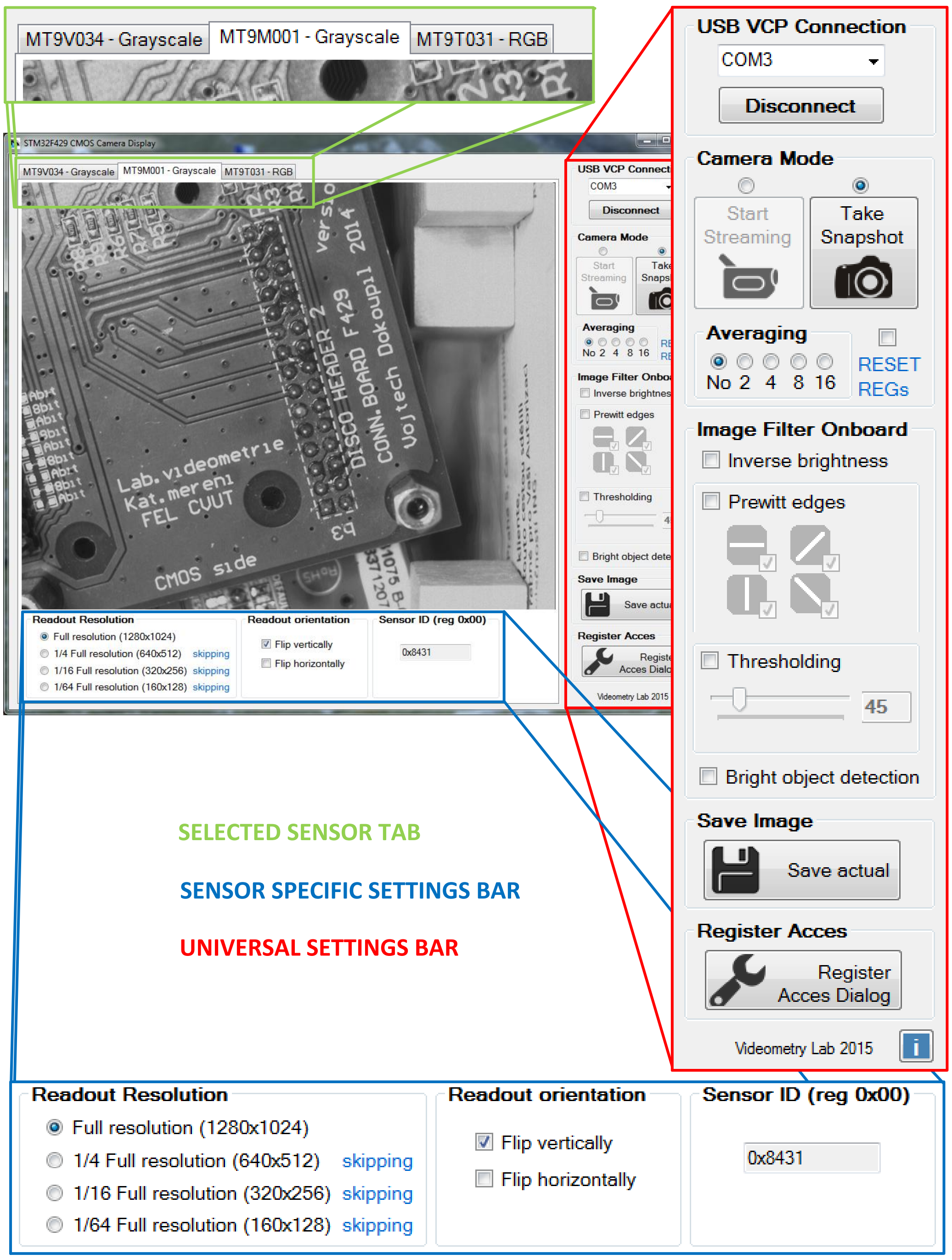


Fig. 27 - Filtering image - operations order

4.3.4. PC Application - GUI in C#

Fig. 28 shows the appearance of the application. I divided the menus to sensor specific settings (in blue) and main bar (in red). The specific readout resolution and also readout orientation is chosen in the sensor specific settings menu. These settings are available on the tab for the specific sensor. The main bar (in red) is visible no matter which sensor is connected and contains control means for connecting sensor, capturing and filtering image. On the top (in green) the actually selected sensor is shown.



USB VCP Connection: COM3

Camera Mode: [✓ Take Snapshot]

Averaging: [✓ No] 2 4 8 16

Image Filter Onboard: ☐ Inverse brightness; ☐ Prewitt edges; ☐ Thresholding 45; ☐ Bright object detection

Readout Resolution:
- [✓ Full resolution (1280x1024)]
- 1/4 Full resolution (640x512) skipping
- 1/16 Full resolution (320x256) skipping
- 1/64 Full resolution (160x128) skipping

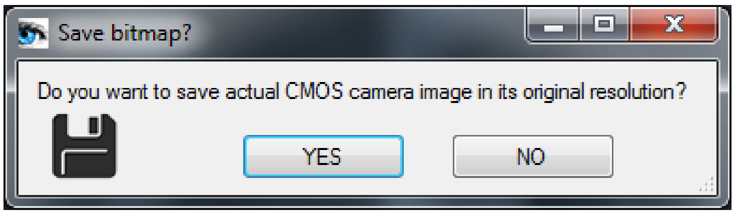
Readout orientation: ☑ Flip vertically; ☐ Flip horizontally

Sensor ID (reg 0x00): 0x8431

Fig. 28 - STM32F429 CMOS Camera display application GUI

4.3.4.1. *Saving Image as a Bitmap*

After capturing an image using buttons for snapshot or streaming the actually shown image can be saved as a bitmap (in original resolution and bit depth). This can be done directly by clicking on the image itself or using "Save actual" button from the main bar (universal settings bar). After completing either of these operations the dialog "Save bitmap?" (fig. 29) is started and user can proceed with naming the file and choosing the destination directory.


Save bitmap?

Do you want to save actual CMOS camera image in its original resolution?

Fig. 29 - Save bitmap dialog

Fig. 30, fig. 31 and fig. 32 show possible output images I saved as a bitmap. All these images were captured from MT9M001 in full resolution (1280 x 1024 pixels). First picture shows application of inversion filter in the CMOS camera. The second image shows Prewitt edge detection output in all four directions possible. The third image shows inverted and then threshold image. As a result we can see easily legible characters. At the top of the image there can be seen the difficulty when using one level thresholding. The unwanted maps from the lower brightness area around resistors can be seen.

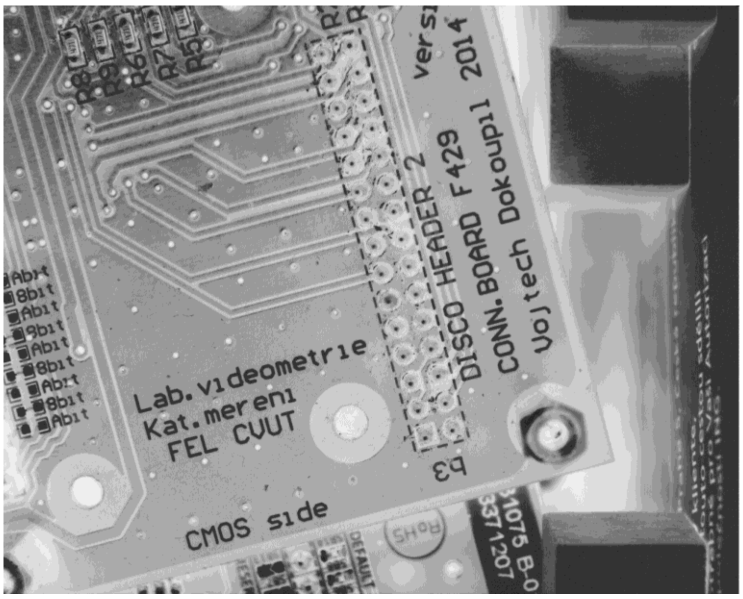


Fig. 30 - Inverted image bitmap from MT9M001

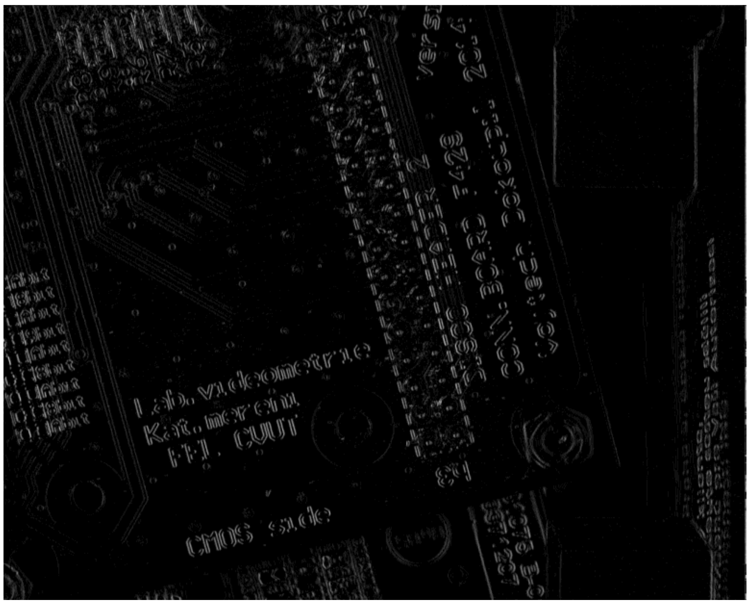


Fig. 31 - Prewitt edge detected image from MT9M001

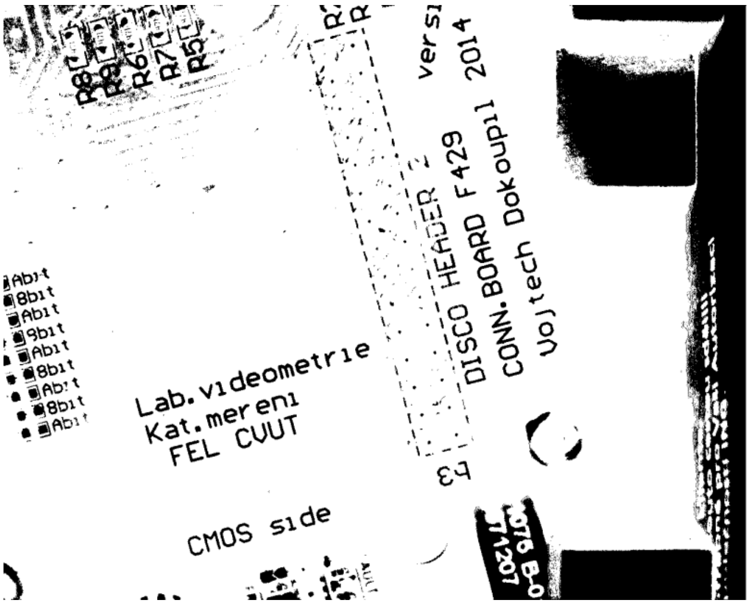
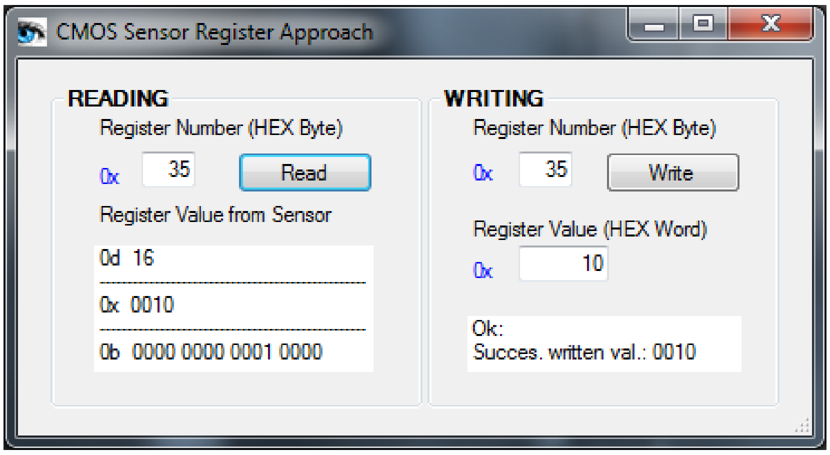


Fig. 32 - Inverted and threshold image from MT9M001

4.3.4.2. Reading and Writing Registers

The supervisor demands the possibility of changing the registers from the PC application. The reason for enabling this possibility is clear. Once I read out an image from the sensor and I want to adjust the gain, brightness or any other parameter (in the application boundaries) I need to change the register settings. The desired values are communicated via VCP on the camera-PC level and via I^{2}C on the MCU-CMOS sensor level as described in the section 4.3.3. If I want the registers to be reset to the initial values the check box "RESET REGs" has to be checked. Otherwise the data written to registers (writable registers) remains at the values set by the user. After clicking on "Register Access Dialog" button the dialog (fig. 33) is opened (I reused my dialog for this purpose from my bachelor thesis).



Register Number (HEX Byte)
0x 35
Register Value from Sensor
0d 16
0x 0010
0b 0000 0000 0001 0000

Register Number (HEX Byte)
0x 35
Register Value (HEX Word)
0x 10
Ok:
Succes. written val.: 0010

Fig. 33 - Register access dialog

The dialog contains reading and writing tables. The value read from the register address specified in the textbox is displayed decimal, hexadecimal and binary in the "Register Value from Sensor" textbox. The register address has to have hexadecimal value between 0x00 and 0xFF (16-bit register addresses). This restriction is applied on the writing address as well. The value to be written to the register has to be hexadecimal. As there can be various sensors connected, there is no write protection. I recommend writing strictly solely into permitted locations. User should check sensors datasheet (or register reference) [13], [14], [15], [16] in order to write only to the locations that are allowed to.

The **serialPort** object is defined in the main form of the application and this dialog approaches it via the constructor. In case more than those two forms access the **serialPort** object it would be better to define extra class of the **serialPort** handling. In this case it is clear to use it this way.

4.3.4.3. RGB Sensor Specific Settings - reading RAW Image without Conversion in the MCU

Since the method used in the MCU for debayerization (see 4.2.4) is quite easy and so the data can be displayed in a worse way than it could be using more sophisticated methods, I added the possibility to read RAW data; this data is then not displayed as RGB as default, but as grayscale in original sensor pixel resolution. They can then be converted to RGB for instance in MATLAB®. This can be done via ticking the check box on the MT9T031 tab page before readout.

5. Example Application - Triangulation Based Object Height Determination

This chapter describes how to create easily a useful optometric application using the previously developed camera software (firmware) and hardware. The application will be capable to measure the object height (respectively cylindrical objects diameter) based on the triangulation principle.

5.1. Triangulation Principle

The principle is based on easy projection geometry. There are two necessary devices for creating this kind of application. We need a camera for image data evaluation and a line laser for creating the laser marks on the projection plane.

The camera has to be fixed in certain position with a specific angle to the line laser position. For instance the laser can shine from the "side" at the angle of 45° from projection surface and has to be rotated in a way to create the line mark perpendicular to the laser direction (see fig. 35 of final application). Camera can be than placed directly above the projection surface (the axis of the camera lens is placed perpendicular to the projection plane).

If we then place the object with suitable reflection parameters on its surface the line mark is created. If we know the geometry of the laser and of the camera we can easily compute the height of the object from the distance between the base laser mark (primary line created by laser on the projection plane) and the mark on the object (I named it "delta" in my application).

The transfer characteristic between the "delta" and the object height should be linear. This is only theoretic issue because the lens has its inaccuracy and the effect of geometric distortion (barrel) can change the projected lines in the side of the image area into arcs.

On the one hand this is the reason why the main laser mark should be projected close to the image area centre. For many objects of small size this is the longest line in the image area and thus it can be captured with the biggest inaccuracy. On the other hand if we place this line into the centre of the image area we effectively use only half of the possible capability of the measured height range.

For my easy demo-application the line is projected little bit outside the image area centre. The range of the possible height that can be measured is in my case also influenced by the laser position and by the holder positioning capabilities. The range of the measured heights is approximately from 5 mm to 50 mm. The barrel distortion is not corrected, but if the object which is measured is placed in the area of the vertical laser axis, the measurement is logically as precise as possible. The application is described in the next sections more in detail.

5.2. Mechanics and HW for Demo Application

In order to prepare easy application for the triangulation height measurement I just used low-cost laser spirit-level. This laser source projects the line to the projection plane. It is attached on a "tripod" and is reclined in a way to project base laser mark under the sensor.

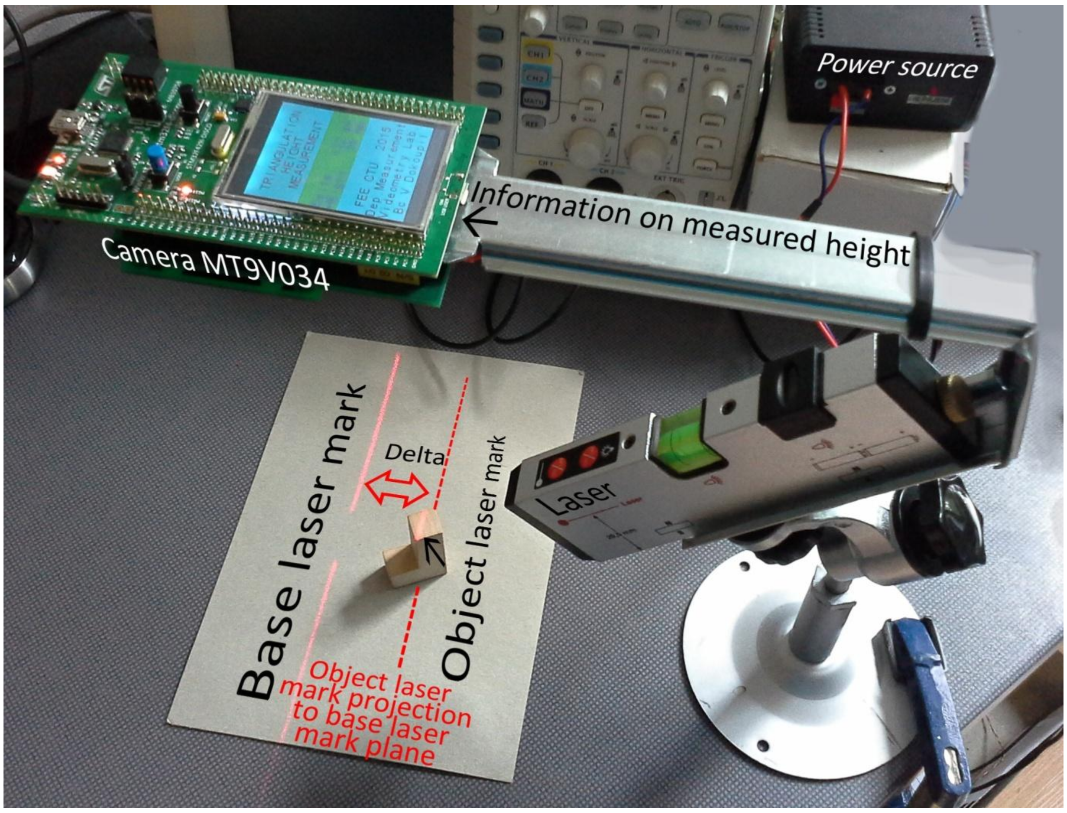

Power source
Information on measured height
Camera MT9V034
TRIANGULATION HEIGHT MEASUREMENT
Base laser mark
Delta
Object laser mark
Laser
Object laser mark projection to base laser mark plane

Fig. 34 - Laser height measurement application

The camera is attached on a pre-prepared holder from DIN35 rail. The fabrication of this holder requires electric drill, hacksaw and approximately 50 centimetres of DIN35 rail. It takes about 20 minutes to prepare it. For other applications similar holders can be prepared without any need to fabricate special mechanics. The holder is attached to the "tripod" under the laser spirit-level holding screw. Fig. 34 shows the whole application with the description of all necessary parts.

The camera is fixed on the holder by the M3 screws. These are attached to the mounting holes (see fig. 55 or fig. 56) of the CONN.BOARD F429 (3.3). The version 2 of the CONN.BOARD F429 is used in this application.

The laser is battery powered (2 x AAA cells) but in case of necessity it can be powered through the power source of 3.3 V on the CONN.BOARD F429. The whole camera is powered from 5 V through the CONN.BOARD F429 to create also lower voltage (3.3 V) for the sensor.

The used sensor type for the camera is MT9V034 (see 3.2.1).

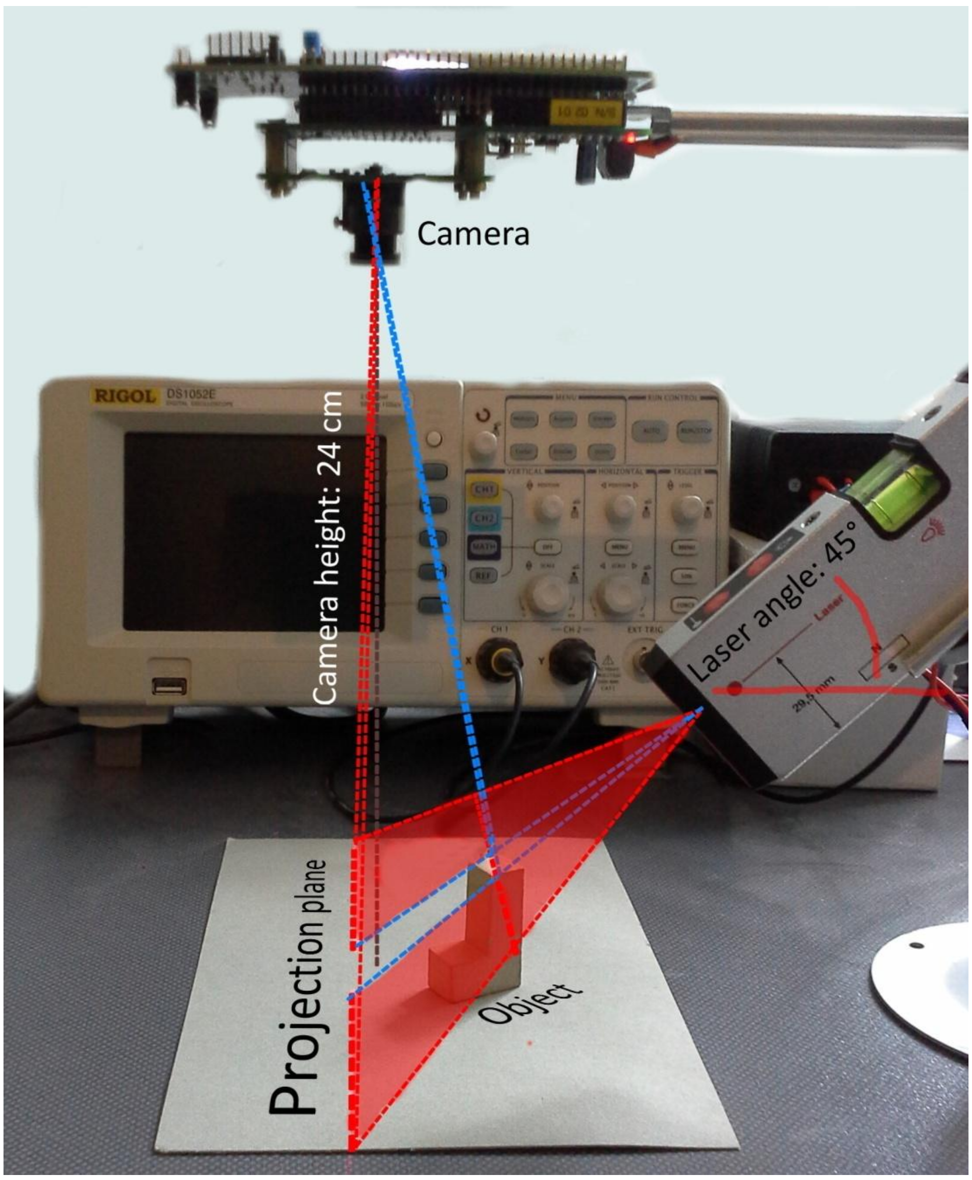


Fig. 35 - Side view on application with basic geometrics description

5.3. Software for the Laser Height Measuring Camera

5.3.1. Sensor Settings

I reused the universal software for MCU developed in previous parts of this thesis. In the development phase I used the image output to the universal application from 4.3.4 to find optimal sensor settings for as stable functionality as possible.

Finally I ended up with following settings. The used resolution for the application is the full one (752 x 480 pixels). The Global gain of the sensor (**register 0x35**) is set to 20 (possible maximum is 64). The Exposure (**register 0x0B**) is set quite low (10 from 32765) to eliminate the effect of the unwanted global lighting conditions. This is possible due to the strong light from the laser which is dominating and thus the exposure time can be very short.

In the final application this is set inside the camera and not through the PC application. The PC communication was used for optimizing the settings.

5.3.2. Image Processing Method Adjustment

The method for integrating the image (see 4.2.3) was used as a principle but it was adjusted to meet the specific requirements of the application. The original methods work with the CG of both x- and y-axes but in this application we need only CG of the y-axis direction. This makes things simpler because the image is saved row-wise in the memory which is more convenient when reading the line data buffer to compute its sum. The adjusted function is called **applyIntegratingImageForLaserDemo** and can be found in this demo-application source code in the image processing library.

The other side of the problem is that we need to find two lines in an image, which lead to finding of the two biggest CGs in y-axis.

Of course the laser line is not projected as an one-pixel bright row in the image. I had to set "forbidden belt of rows" which is skipped when finding the other laser mark (second CG) not to detect the same laser mark multiple times.

In the first run the brightest line in the image is found being either the **object laser mark** or the **base laser mark**. In the second run the part of the image above this line with certain y-offset is being searched and the brightest line of this image part is saved (y-coordinate and the sum of the row). The same procedure follows for the image part located bellow the brightest line in the image (found in the first run). Then the two brightest rows from the upper and lower image fragments are compared and the brighter value together with its y-coordinate is interpreted as a second laser mark. This is hidden in the function called **applyHeightMeasuringPx** in the laser height measuring demo-application software project in the image processing library.

In the last part of the **applyHeightMeasuringPx** function the pixel difference of the two laser marks found in y-direction is computed. It is not important which one of the marks is the base and which is the object one. The absolute value called simply "delta" is computed.

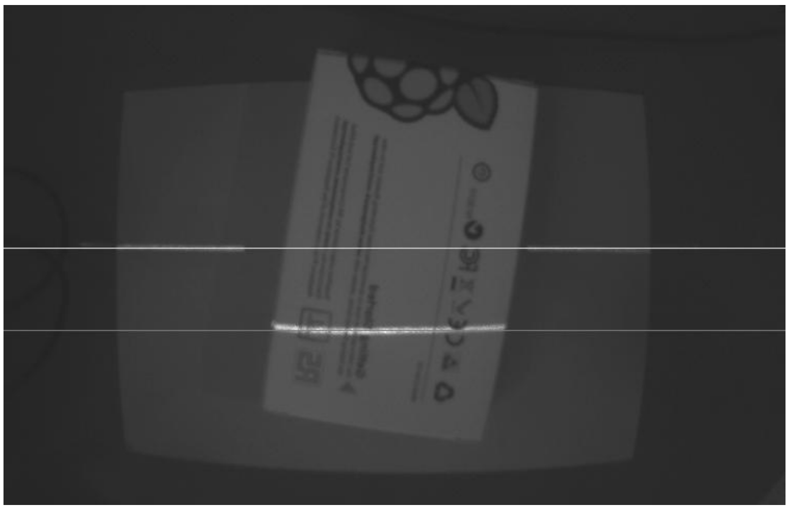


Fig. 36 - Detecting laser marks using image integration (original image)

It has to be mentioned that both situations of the laser mark detection can occur. If the object is wide and has good reflectance (white and glossy) than the brightest row can be easily found on this object (object laser mark). The second brightest line is then the base laser mark. On the other hand if the object has small width and the average reflectance it is obvious that the base laser mark is detected in the first run. This is the reason why I have to find the second brightest line in both upper and lower parts of the image.

I tested two possible image "formats" as an input for the described method. Firstly I integrated the original image (can be seen in fig. 36) and secondly I integrated the thresholded image (fig. 37). As I predicted the first method was more robust so I skipped the thresholding in the final application and used directly the captured image to be integrated in y-direction.

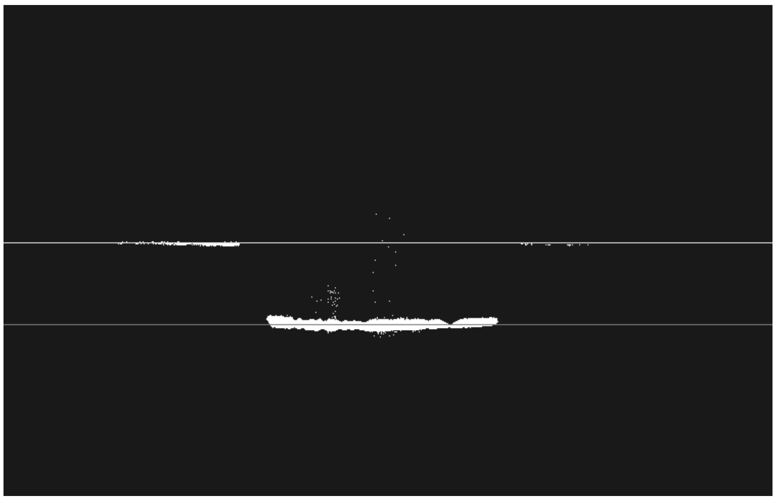


Fig. 37 - Detecting laser marks using image integration (thresholded image)

5.3.3. Transfer of the Image Output to the Height Value

The application mechanics is not precise and the aim of this demo is more than micrometre accuracy of the optical height-measurement procedure showing the method and its creation using the overall camera software depicted in chapter 4. I used the most straight-forward calibration method which is possible.

I picked few objects with known height with suitable reflectance. These were few boxes, wooden dices, PET bottle cap, white board marker (diameter) and spray (diameter). For these objects I measured the "delta" of the laser marks in pixels. Along with the height values in millimetres I created a graph of real transfer characteristics. This characteristic was linearized (least-squares method) and implemented as a transfer equation to the microcontroller. Both of these characteristics along with the equation can be seen in fig. 38.

It has to be mentioned that this computation is specific for the mechanics settings and has to be proceeded again in case the camera height etc. is changed.

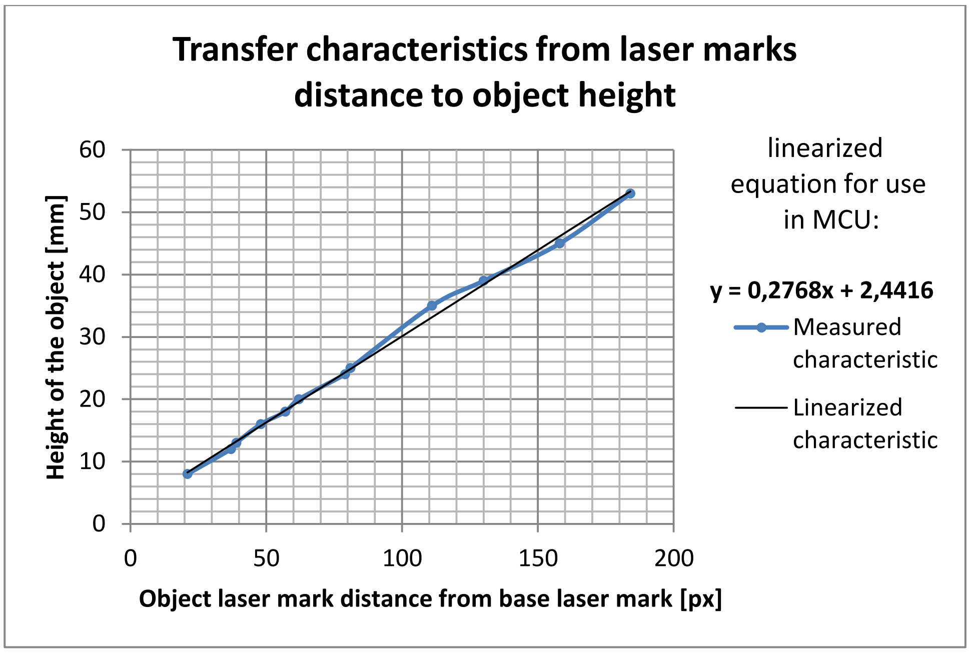


Fig. 38 - Transfer characteristic from distance "delta" of the laser marks to object height

The tested capability of the height measurement in terms of the range of possibly measured heights can lie between 5 - 55mm. For the lowest values the problems with the "forbidden belt of rows" mentioned in 5.3.2. influence the height measurement capability. On the contrary the higher values evince the problem with the position of the laser and the view of the camera.

Due to the reflectance behaviour also the **cylindrical object diameter** (as the white board marker or spray) can be measured. The point placed in the laser vertical axis plane on top of the cylindrical surface reflects the laser light most effectively towards the camera direction and this part of the laser mark is then recognized as the laser mark of the object.

5.3.4. User Input and Application Output on the Display

Standard STMicroelectronics® LTDC (LCD controller - lower level) and LCD (LCD Library) Libraries were used to show the output of the height measurement on the display. Fig. 39 shows that both the "delta" value in pixels and height value in mm can be seen.

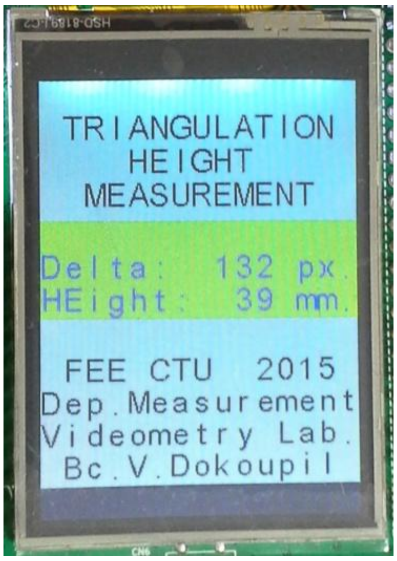

TRIANGULATION
HEIGHT
MEASUREMENT
Delta: 132 px.
HEight: 39 mm.
FEE CTU 2015
Dep.Measurement
Videometry Lab.
Bc.V.Dokoupil

Fig. 39 - STM32F429I-Disco display for the laser height measurement application

New measurement is triggered when user button (BLUE on the Disco kit) is pressed. In the application the CMOS sensor has to be configured at first and then the LCD has to be configured (the problem with the same wires as mentioned in chapter 6.2); this has to be repeated in each run of the application.

6. Additional Applications Connected to Integrated Camera Solution

6.1. Horizon Detection - Quick Standalone Application

This topic was mainly covered by my individual project. Based on the specificity of this application I decided not to cover it by the universal firmware for the developed camera, but let it work as one purpose application with extra firmware. The used image sensor is MT9V034. The development kit is STM32F429I-Disco.

6.1.1. Testing Possible Methods of Horizon Detection in MATLAB

6.1.1.1. Image Data Pre-processing

For operations in MATLAB® I used common functions, which will probably have to be implemented on MCU either. For white noise removal I applied 3 x 3 pixel neighbourhood Gaussian filter. Next I tried to apply opening and closing the image (according to [6]) which consisted of MATLAB® functions **imerode** and **imdilate.** The aim of this operation was removing differences in local brightness, which can be misinterpreted as the horizon line points after edge detection. The supposed effect of this operation was not reached. The quality of finding of one line horizon is almost not affected by this image operation.

The main influence on the correct horizon line detection has again the selection of suitable threshold (for distinguishing the sky and earth surfaces) for particular "situation" in the field of the view of the camera. Right after Gaussian averaging I applied thresholding to divide image into two parts - sky and the earth surface (binary image). Following operation is then the horizontal mask of the Prewitt type to detect the edge between these two areas. Output of these operations can be seen in fig. 40.

I used the methods which are based on local information, but for greater success it will be better to take also global information into account as written in [7].

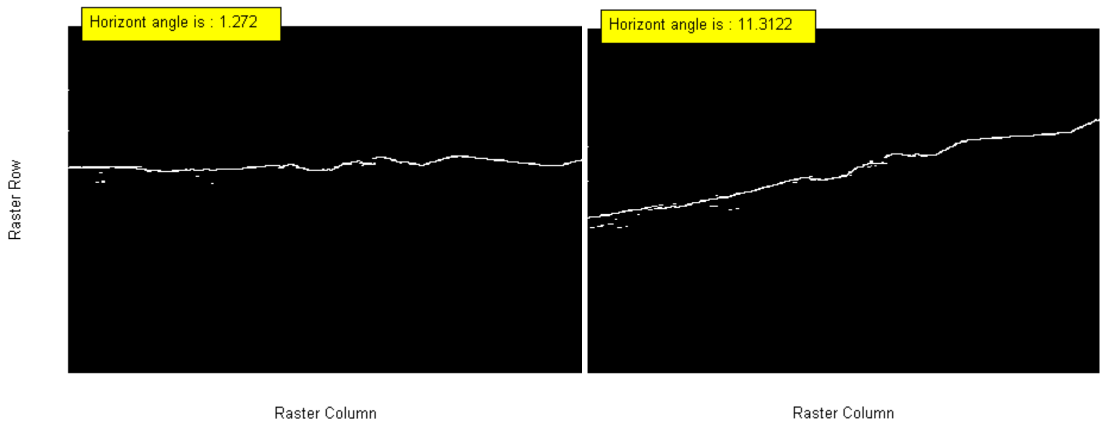


Fig. 40 - Pre-processed image - horizon detection

6.1.1.2. *Obtaining Horizon Points*

For each image column the average y-coordinate of all found horizon points (unfortunately possibly also false positively detected horizon points) is computed. If no horizon point is present in the current column then I set its y-coordinate on the value of the y-coordinate of the horizon point from its left adjacent column. The success of the horizon reconstruction is mainly based on the quality of the image pre-processing. This data can be seen in fig. 42 and fig. 43 as a blue line.

6.1.1.3. *Computing Roll Angle*

The desired output of this application is the information on the roll angle of the horizon relative to the CMOS camera frame. In order to compute an approximate value of this angle we have to proceed with the linearization on the horizon points (then we had "global horizon direction"). The simplicity for MCU implementation is desired, thus I only use the least squares method to obtain the tangent of the desired angle. Moreover, I am not interested in the absolute coefficient of the found line equation which might be used for the pitch angle but it is not necessary for the roll one.

Equ. 3 shows the simplified way how to compute desired tangent alpha. The notation of the angle polarity is shown in fig. 41.

$$\tan\alpha = \frac{(n \cdot \sum_{i=1}^{n} y_i \cdot i) - (\sum_{i=1}^{n} y_i \cdot \sum_{i=1}^{n} i)}{(n \cdot \sum_{i=1}^{n} i^2) - (\sum_{i=1}^{n} i)^2}$$

$$n = number\ of\ image\ data\ columns\ for\ linearization\ computation$$

$$y = horizon\ point\ y\ coordinates\ relative\ to\ the\ lower\ left\ corner\ of\ the\ image$$

Equ. 3 - Roll angle (alpha) tangent computation - least squares method

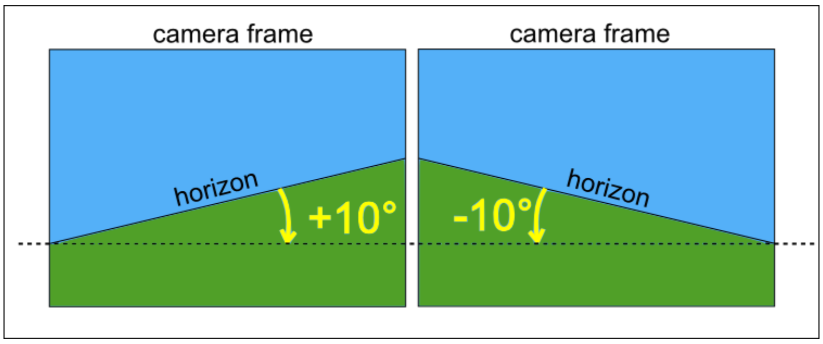


Fig. 41 - Notation used for distinguish between left and right roll

Following pictures fig. 42 and fig. 43 show the detected horizon (red line) which is the line for computing a roll angle.

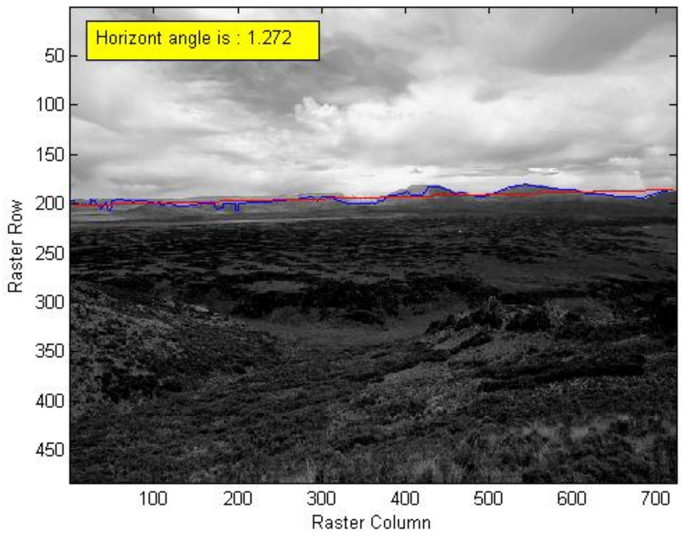


Fig. 42 - Linearization of horizon line using least squares method (original image in background)

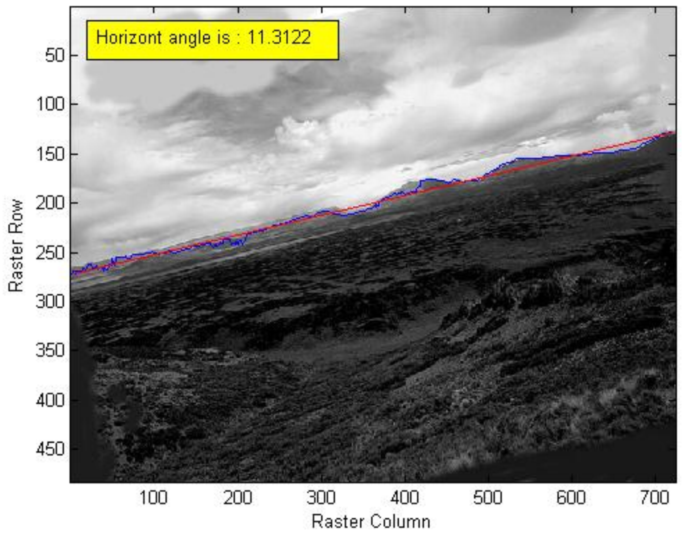


Fig. 43 - Linearization of horizon line using least squares method (rotated image in background)

6.1.2. Implementation Horizon Detection on STM32F29 - Disco kit

6.1.2.1. Setup of the LCD Display

The current software for the snapshot camera from [9] uses only one displayed Layer to show the data captured by the Image sensor. For providing user with some simple information on the roll angle detected on the kit I decided to configure the second displayed Layer which is available due to the capabilities of the on-board display driver. Thanks to the blending settings (see alpha parameter in [8]) both image and angle information can be then shown on the display. Each layer can be configured in its own way. In this case one layer is configured only for getting image information from the memory in the grayscale format (**LTDC_Pixelformat_L8**). The other layer uses display character settings (only one line) pre-programmed in STMicroelectronics® support libraries. The settings are not optimal, because the characters are shown on the shorter side of the display (90 degree rotated relative to the image). At least we have larger area of the display for image displaying

6.1.2.2. Roll Angle Detection on STM32F429

Fig. 44 describes simplified roll angle computing process (the angle relative to the longer side of CMOS sensor or display if using HW described in 3.3.2). The blocks in the light blue are focused on image filtration (as described in 6.1.1.2) and then the linear approximation of the horizon line points according to equ. 3.

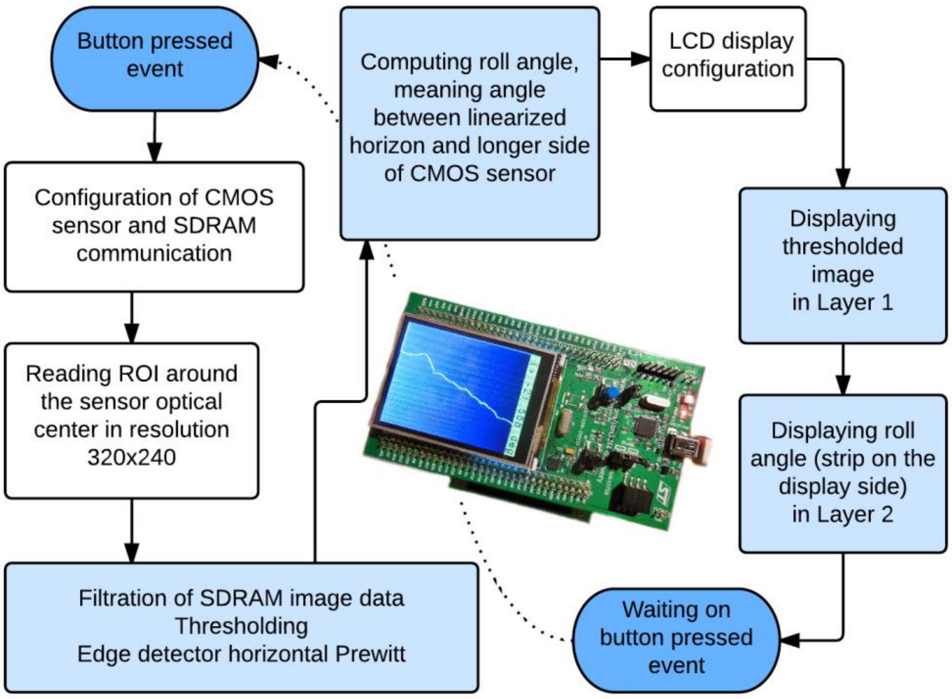

Button pressed event
Computing roll angle, meaning angle between linearized horizon and longer side of CMOS sensor
LCD display configuration
Configuration of CMOS sensor and SDRAM communication
Displaying thresholded image in Layer 1
Reading ROI around the sensor optical center in resolution 320x240
Displaying roll angle (strip on the display side) in Layer 2
Filtration of SDRAM image data
Thresholding
Edge detector horizontal Prewitt
Waiting on button pressed event

Fig. 44 - Horizon detection demonstration application flow diagram

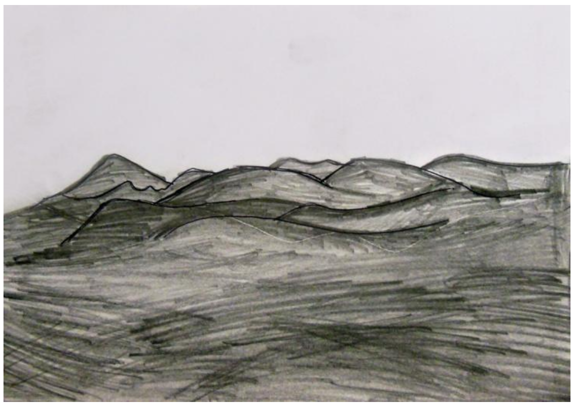
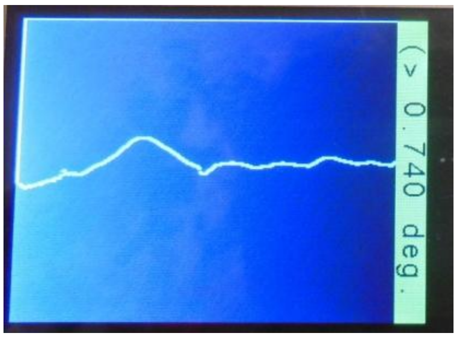
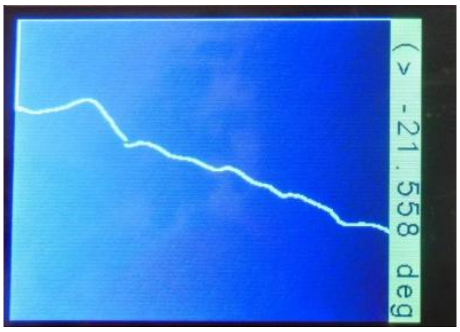


Fig. 45 - Idealized scenery test image

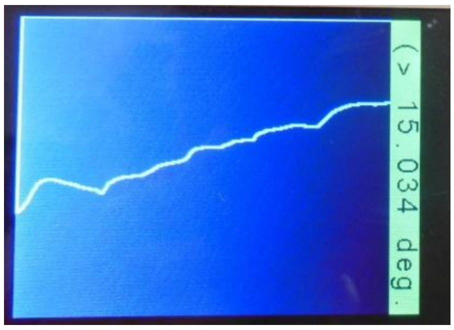
Fig. 29 shows idealized pattern image used for obtaining the test screens. The image was rolled to both left and right and the screens showed in fig. 46. The image distance from the lens was approximately 20 centimetres. The environment was stable (in terms of the scene light conditions) and the threshold level was fixed on certain level. If moved to other test situation it has to be improved with a calibration or possibility of setting a threshold.



a)



b)



c)

Fig. 46 - Display 3 possible situations of the output of horizon detection algorithm.

a) Horizontal direction (roll angle close to zero: 0,740°)

b) Negative roll angle (-21,558°)

c) Positive roll angle (15,034°)

6.2. RGB CMOS Camera Using Integrated LCD Display

For purposes of the easy demonstration I prepared easy demo of RGB camera displaying data on the LCD display. This is variation on Bc. Daniel Toms´ grayscale camera from his bachelor thesis [9].

6.2.1. CMOS Readout Configuration

The resolution of the LCD display is 320 x 240 pixels. The resolution of the RGB sensor is 2048 x 1536 pixels. If I use simple conversion method (described in section 4.2.4) I reduce maximal resolution to 1024 x 768 pixels. I want to show the image from as wide view angle as possible so I do not want to use ROI from the full resolution. If I use skipping 3 rows and 3 columns which is possible on MT9T031 I have 351 x 256 pixels resolutions. To get the precise dimensions for the display I use then windowing (ROI) which leads only to small crop of the image (31 pixels in x-direction and 16 pixels in y-direction). To eliminate the possible aliasing I also set binning to the CMOS sensor (2 columns / 2 rows for binning). The skipping settings of the sensor have to be higher than binning settings (3 skipping and 2 binning). This condition is satisfactory.

6.2.2. MCU Application

The application is very easy. After pressing the user button on STM32F429I-Disco kit the sensor configuration proceeds (including GPIO reconfiguration). Then the image is captured in the specified configuration and saved to SDRAM. After saving it the transfer function from the section 4.2.4 is called and image is translated to RGB888. The displaying part follows. The GPIOs and LTDC are reconfigured, because CMOS sensor is on the same wires. After that the image from SDRAM is displayed in RGB888 format on the LCD.

The application is only demonstrative because I need to reconfigure the GPIO. For real application this is slow and unusable.

6.3. Audio Examples for Student Applications Using STM32F4 Discovery with Integrated CS43L22 Audio Codec

The examples are focused on expansion of SUMONDI demo application developed in my bachelor thesis [12] with the audio output. The audio output in this case means that after recognizing the number of dots on dices, the sound for the number will be played. The solution also includes the command for throwing dices. All the sounds can be played in CZ or EN version. The hardware (audio codec CS43L22 [17]) is present on STM32F4 board. This topic was already mentioned in the bachelor thesis of Ing. Marek Bílý [10]. There is one complication with the camera interface and audio codec interface. Some signals are shared (both peripherals are connected to the same pins of processor). There are two possible ways to solve this. The easier solution proposed is to use the second kit of the same type just for audio processing and to connect it through some simple interface to the "master" kit which will provide the information based on the image processing to select the appropriate audio track. The other possible solution is multiplexing in time domain between those two interfaces on the same wires.

6.3.1. Decentralized Solution with Two STM32F4 Discovery Kits

For the first insight to the codec programming and adjusting its settings the sound was generated on another STM32F4 Discovery kit than the image processing one. The information about the desired sample (number, command to throw) was simply sent through I/O parallel interface based on four GPIO pins on each kit side. This transfer can be easily changed to serial (UART, SPI) communication as well, which would be useful in case we need to code more than 13 states as I did. Basic setup is shown in fig. 47.

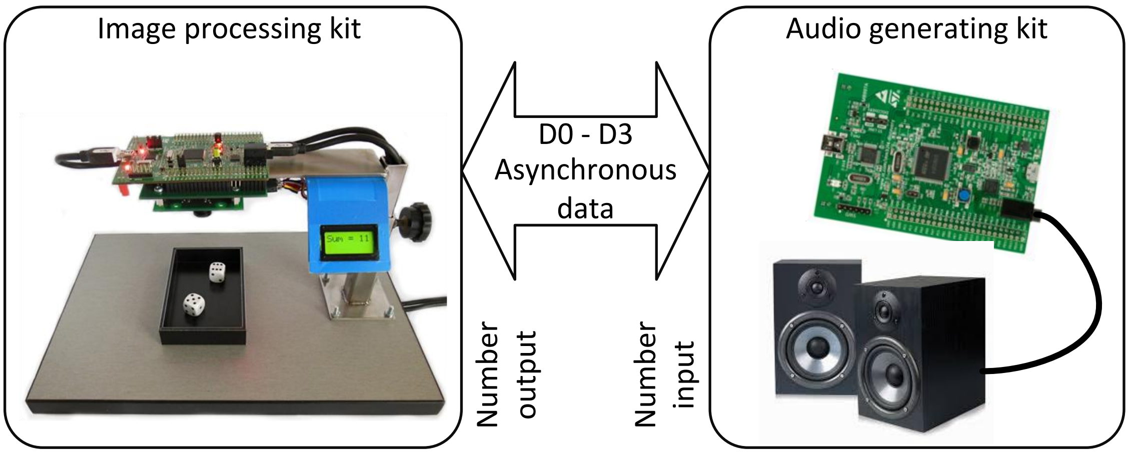


Fig. 47 - Decentralized audio solution setup

6.3.2. Integrated Solution Using Time Multiplex between CMOS Sensor and Audio Codec

In order to decrease the size and increase the compactness of the SUMONDI demo it appears necessary to integrate both functions (image processing and audio) to one STM32F4 Discovery kit. Time multiplexing principle between CMOS DCMI interface and codec I^{2}C and I^{2}S interfaces has to be implemented. The root of the problem is that with current hardware developed in my bachelor thesis the pins are mapped on the same pins (it was not supposed to use the audio output with this demo). The integrated setup is sketched in fig. 48.

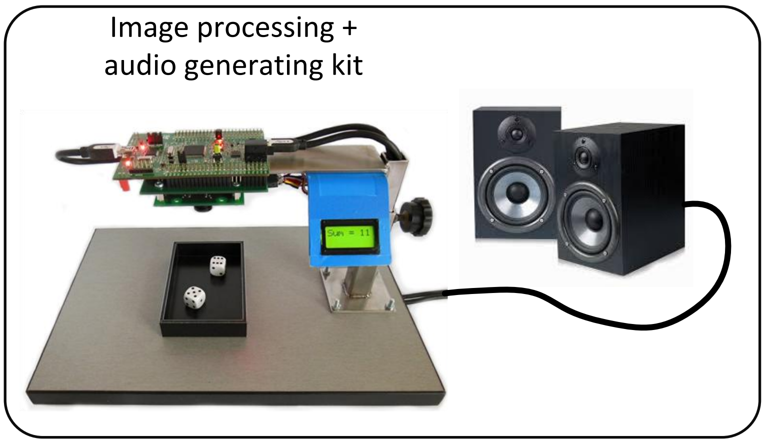


Fig. 48 - Integrated audio solution setup

The dual - language function was demanded. To provide best usability, the language can be changed anytime, also during the program run, not only in the calibration stage. The language selection is set according to jumper position. In fig. 49 there can be seen the English language selection via connecting PD8 and PD10 pins. In fig. 50 there can be seen Czech language selection via connecting PD9 and PD10 pins. If no jumper is present, the sound is switched off; this can be seen in fig. 51.

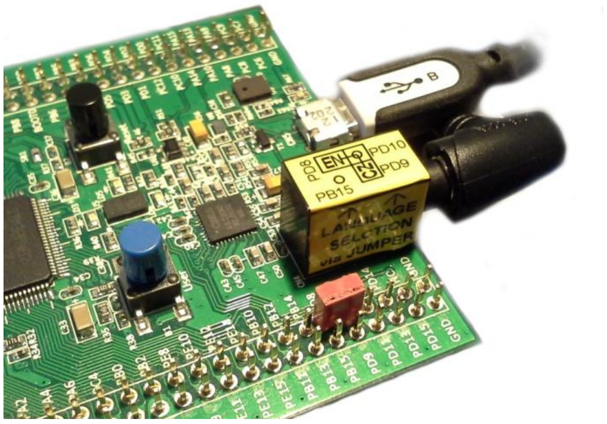


Fig. 49 - Selection of the English sound version

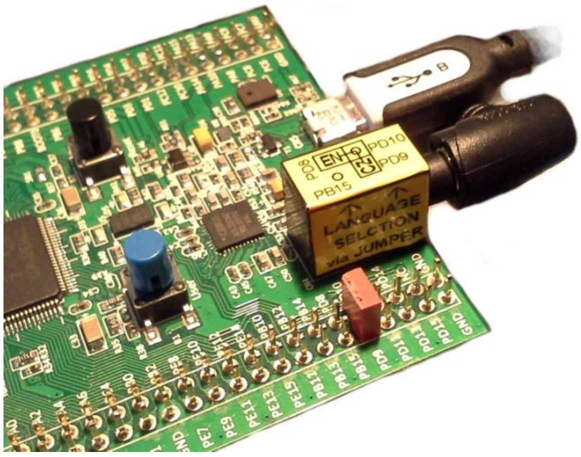


Fig. 50 - Selection of the Czech sound version

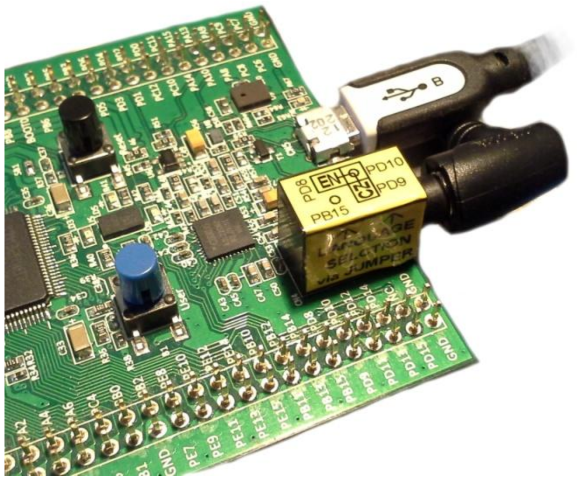


Fig. 51 - No sound selection

The audio add-on for SUMONDI demo is described in fig. 52.The new parts of the program in MCU are highlighted in the light blue colour. I have few notes concerning this diagram. The reconfiguration of the audio codec takes some time. Due to this fact it is performed only if the language selector is present or if a change in the scene occurred. This saves time in case there is no change in the scene or if the user does not need an audio output.

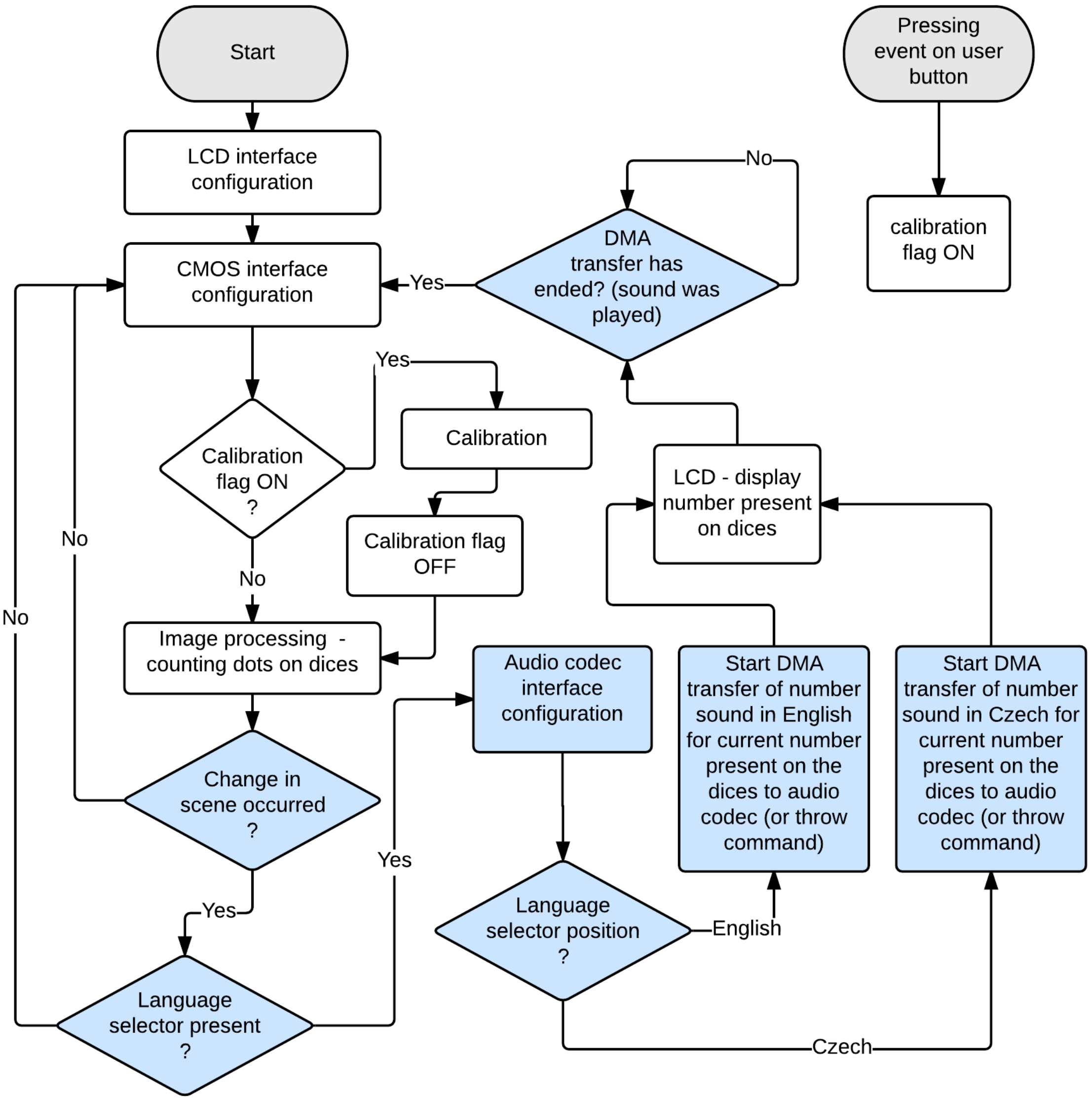

Start
LCD interface configuration
CMOS interface configuration
Calibration flag ON ?
Yes
No
Calibration
Calibration flag OFF
Image processing - counting dots on dices
Change in scene occurred ?
No
Yes
Language selector present ?
No
Yes
Audio codec interface configuration
Language selector position ?
English
Czech
Start DMA transfer of number sound in English for current number present on the dices to audio codec (or throw command)
Start DMA transfer of number sound in Czech for current number present on the dices to audio codec (or throw command)
LCD - display number present on dices
DMA transfer has ended? (sound was played)
No
Yes
Pressing event on user button
calibration flag ON

Fig. 52 - SUMONDI program change - usage of audio output

The specific parameters of audio files included in MCU are described in great detail in the following section 6.3.3.

6.3.3. Audio Samples Recording and Pre-processing in PC

There can be seen the graphical interpretation of the original record of my counting sounds from one to twelve (this record is the Czech version mentioned before) performed by basic Windows Sound Recorder in the fig. 53. The process of obtaining numbers for 16-bit audio codec was following: the record has to be transfered from wma to wav, which is standard audio format without compression. As soon as we have a wav file, it can be opened by a number of applications for the sound treatment. I have chosen Audacity® which is quite useful freeware application.

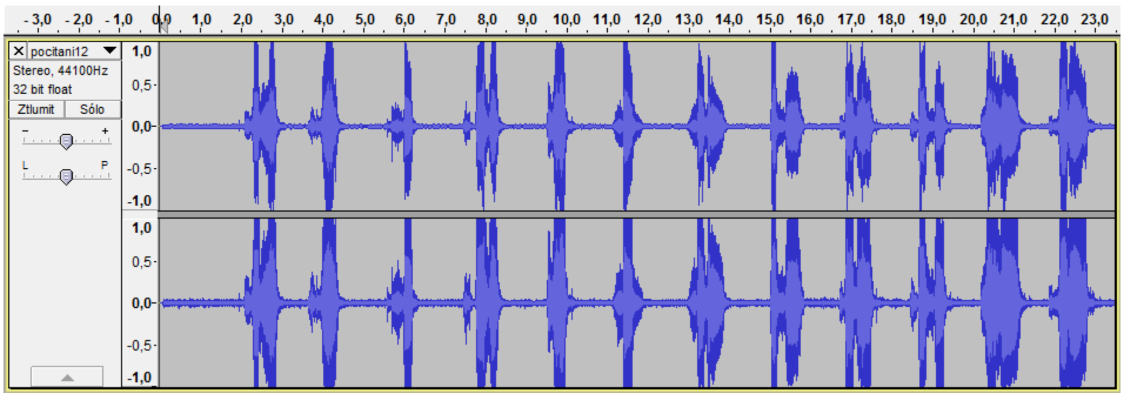


Fig. 53 - Original audio record of counting from 1 to 12 in CZ

The following step is to split the record to twelve sub-tracks corresponding to the specified numbers. These tracks were resampled to 11025 KHz from default 44100 kHz. The stereo files were changed to mono ones. Now the conversion to the "excel-type" table of values via MATLAB function **vawread** can be performed. In this moment I have the data ready to be moved to flash-memory space on STM32F4 Discovery. This part of the conversion process can be seen in fig. 54. In order to provide as stable solution as possible (in terms of codec stability) I decided to adjust the record sizes to exactly the same size. This was useful to the certain point, but some records were longer than I expected in the first stage of the development. That is why the possibility of longer record was added later (in the Czech version this was used in two out of thirteen records, the same in the English version). It can be also seen in fig. 54.

The usage of the flash memory, which has the volume of 1 MB on this kit, is presented in tab. 9. It is obvious that there is not enough space left for other sounds. This could be solved using external flash memory like an USB stick, but that is not part of this example.

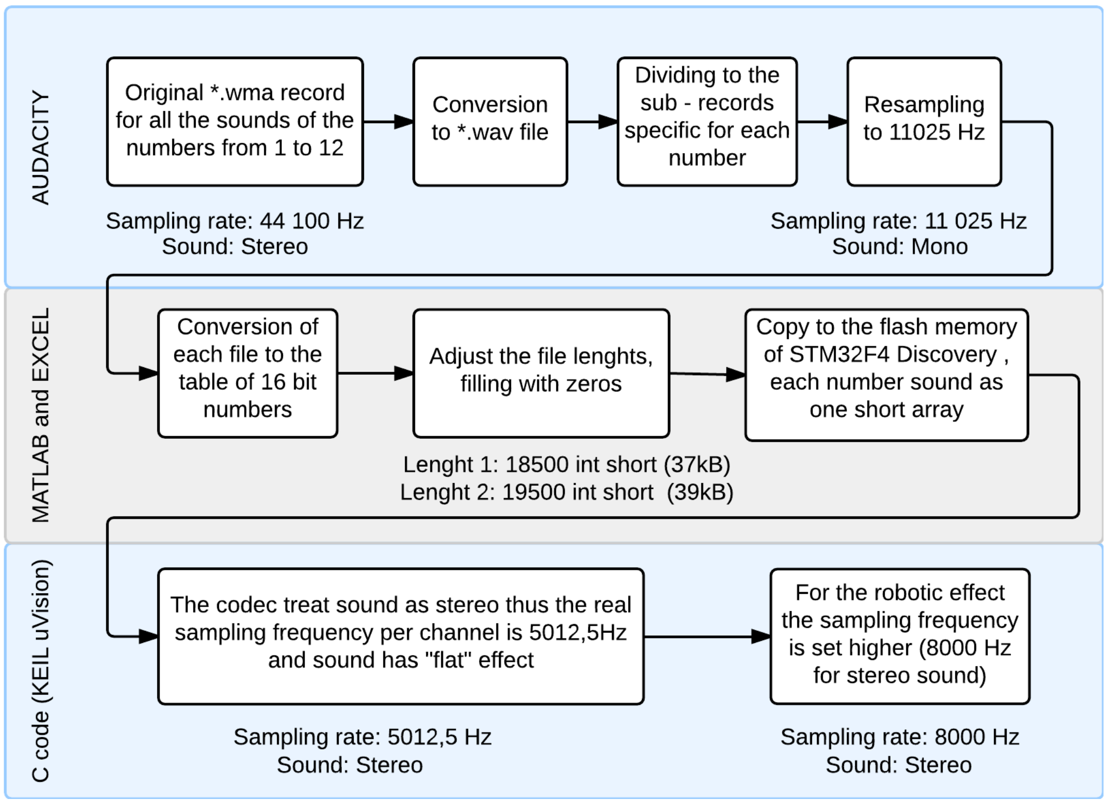

AUDACITY
Original *.wma record for all the sounds of the numbers from 1 to 12
Conversion to *.wav file
Dividing to the sub - records specific for each number
Resampling to 11025 Hz
Sampling rate: 44 100 Hz
Sound: Stereo
Sampling rate: 11 025 Hz
Sound: Mono
MATLAB and EXCEL
Conversion of each file to the table of 16 bit numbers
Adjust the file lenghts, filling with zeros
Copy to the flash memory of STM32F4 Discovery , each number sound as one short array
Lenght 1: 18500 int short (37kB)
Lenght 2: 19500 int short (39kB)
C code (KEIL uVision)
The codec treat sound as stereo thus the real sampling frequency per channel is 5012,5Hz and sound has "flat" effect
For the robotic effect the sampling frequency is set higher (8000 Hz for stereo sound)
Sampling rate: 5012,5 Hz
Sound: Stereo
Sampling rate: 8000 Hz
Sound: Stereo

Fig. 54 - Audio conversion process diagram

CZ version		EN version	
sound track	**size**	**sound track**	**size**
"jedna"	37kB	"one"	37kB
"dvě"	37kB	"two"	37kB
"tři"	37kB	"three"	37kB
"čtyři"	37kB	"four"	37kB
"pět"	37kB	"five"	37kB
"šest"	37kB	"six"	37kB
"sedm"	37kB	"seven"	37kB
"osm"	37kB	"eight"	37kB
"devět"	37kB	"nine"	37kB
"deset"	37kB	"ten"	37kB
"jedenáct"	**39kB**	"eleven"	37kB
"dvanáct"	**39kB**	"twelve"	**39kB**
"hoď kostky"	37kB	"throw dices"	**39kB**
	448kB		**448kB**
Flash size used	896 kB		

Tab. 9 - Flash memory usage - audio samples

7. Conclusion

The diploma thesis was focused on creating the CMOS camera based on the STM32F429I-Disco kit and on existing CMOS module boards. The camera can be used as an autonomous measurement unit and also as a CMOS sensors' parameters measurement device. The assignment requires both HW and SW part development.

The hardware interface board called "CONN.BOARD F429" to be placed between existing CMOS sensor module and Disco kit was developed and tested successfully in 2 variants. In the first variant of this interface board the connector is rotated in the way the sensor can be placed in the same orientation as the LCD on the kit. This is typically used in applications where the image is displayed directly on the LCD. This more convenient concept needs larger PCB area and more complicated signal routing. The second version requires little less PCB area but the sensor is rotated by 90° to the built-in LCD. The "CONN.BOARD F429" boards also provide the possibility to be attached to a holder by four 3-mm mechanical holes. Six boards were fabricated and then hand - assembled.

The MCU firmware for communicating with the CMOS sensor via DCMI and I^{2}C was developed. Various DMA settings based on the transferred image size were tested. Various readout possibilities of three different image sensors were implemented.

Image filtering including thresholding, inversion, four-direction Prewitt edge detection and integrating image for the bright object localization were implemented on the MCU level.

To demonstrate the sensor readout possibilities an easy C# application was written as the display for the image output. This output can be saved in PC to a bitmap for further processing. The application provides the automatic sensor identification because various CMOS sensors can be attached

The laser height measurement demo-application was presented as an easy optometric embedded application suitable as an example assignment for student semester project. The software for this application was derived from the overall STM32F429 based camera. As an output the built-in LCD display along with libraries from STMicroelectronics® was used.

The horizon detection methods were studied and tested using MATLAB® and also a basic application was programmed on the MCU level.

For the higher-resolution sensors the CS Mount flange basis was drawn and printed on 3D printer as an alternative to the phenolic paper variation. Further the easy mechanics for the laser height measurement demo was presented as well.

As a side product the audio generation output was tested on STM32F4 Discovery kit to produce universal human interface output for videometric applications.

All the parts of the assignment of this thesis focused on STM32F429 Based CMOS Camera for Teaching Purposes were fulfilled. This means that the desired hardware (interface board called “CONN.BOARD F429”) and software (firmware for the camera, PC application) were developed. The platform is prepared for usage in videometry classes - both for image sensor parameter measurement and autonomous students’ measurement/regulation applications. This was demonstrated by an easy laser based height measurement demo.

8. List of Resources

[1] STMicroelectronics: DS9405: STM32F429 Datasheet. [online]. 2015, DocID024030 Rev5, p. 226 [quot. 2015-01-25]. Available from: http://www.st.com/

[2] STMicroelectronics: RM0090: Reference Manual. [online]. 2015, DocID018909 Rev9, p. 1718 [quot. 2015-01-25]. Available from: http://www.st.com/

[3] Yiu J.: *The definitive Guide to the ARM Cortex-M3*, Elsevier, 2007

[4] STMicroelectronics: UM1472: Discovery kit for STM32F407/417 lines. [online]. 2012, DocID022256 Rev2, p. 38 [quot. 2015-03-15].
Available from: http://www.st.com/web/catalog/tools/FM116/SC959/SS1532/PF252419

[5] STMicroelectronics: UM1670: Discovery kit for STM32 F429/439 lines. [online]. 2013, DocID025175 Rev 1, p. 35 [quot. 2015-03-16].
Available from: http://www.st.com/web/catalog/tools/FM116/SC959/SS1532/PF259090

[6] HLAVÁČ, Václav. Matematická morfologie: presentation. [online]. p. 42 [quot. 2015-04-17]. Available from: http://cmp.felk.cvut.cz/~hlavac/TeachPresCz/11DigZprObr/71-06MatMorfolGrayCz.pdf

[7] ZAFARIFAR, Bahman, Hans WEDA and Peter H.N. DE WITH. *Horizon detection based on sky-color and edge features*. 2008, p. 9. [quot. 2015-04-17], DOI: SPIE-IS&T Vol. 6822 682220.

[8] ST MICROELECTRONICS: Embedded graphics on STM32F4: presentation [online]. [quot. 2015-01-04]. Available from: "http://www.compel.ru/wordpress/wp-content/uploads/2013/11/1_LTDC_ChromeART.pdf"

[9] TOMS, Daniel. *Image Processing for an Optical Spot Tracking* [online]. Prague, 2014 [quot. 2015-04-05]. Bachelor thesis. CTU. Supervisor: doc. Ing. Jan Fischer, CSc.

[10] BÍLÝ, Marek. *Signal and Image Processing Using STM32F405* [online]. Prague, 2012 [quot. 2015-04-22]. Bachelor thesis. CTU. Supervisor: doc. Ing. Jan Fischer, CSc.

[11] NOVÁK, Tomáš. *C Library for work with sensor CMOS* [online]. CTU. Prague, 2013

[12] DOKOUPIL, Vojtěch. *Image Processing Using Embedded Microcontroller* [online]. Prague, 2013 [quot. 2015-04-22]. Bachelor thesis. CTU. Supervisor: doc. Ing. Jan Fischer, CSc.

[13] APTINA: 1/3-Inch Wide-VGA CMOS Digital Image Sensor MT9V034: datasheet. [online]. 2015, rev.F 03/15 EN, p. 61 [quot. 2015-04-14].
Available from: http://www.aptina.com/support/documentation.jsp?t=0&q=137

[14] APTINA: 1/2-Inch Megapixel CMOS Digital Image Sensor MT9M001: datasheet. [online]. 2015, rev.L 03/15 EN, p. 28 [quot. 2015-02-14].
Available from: http://www.aptina.com/products/image_sensors/mt9m001c12stm/

[15] APTINA: 1/2-Inch Megapixel CMOS Digital Image Sensor MT9M001: register reference. [online]. 2010, rev.A 03/10 EN, p. 13 [quot. 2015-02-14].
Available from: http://www.aptina.com/products/image_sensors/mt9m001c12stm/

[16] APTINA: 1/2-Inch 3-Megapixel CMOS Digital Image Sensor MT9T031: datasheet. [online]. 2011, rev.E 05/11 EN, p. 46 [quot. 2015-02-14].
Available from: https://www.aptina.com/assets/downloadDocument.do?id=808

[17] CIRRUS: DS792F2: Audio Codec CS43L22 datasheet. [online]. 2010, 03/10 EN, p. 66 [quot. 2015-03-14].
Available from: http://www.cirrus.com/en/pubs/proDatasheet/CS43L22_F2.pdf

Annex 1 – Assembly Instruction for the CONN.BOARD F429 Version1/2

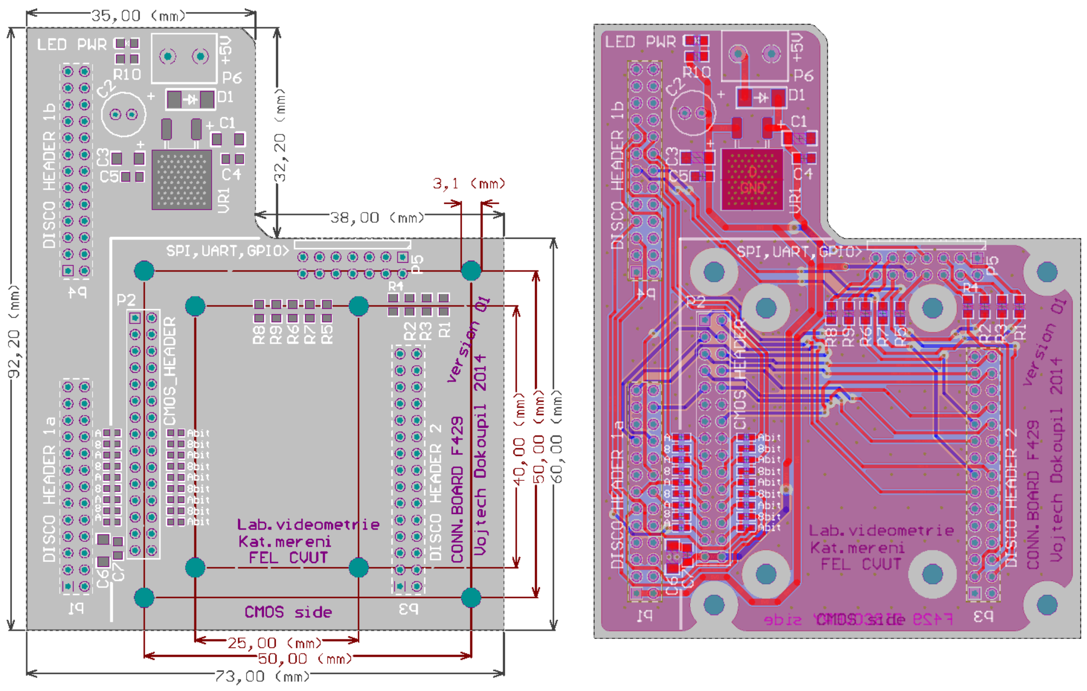


Fig. 55 - Mechanical (dimensions included) and routing drawing of CONN.BOARD F429 version1

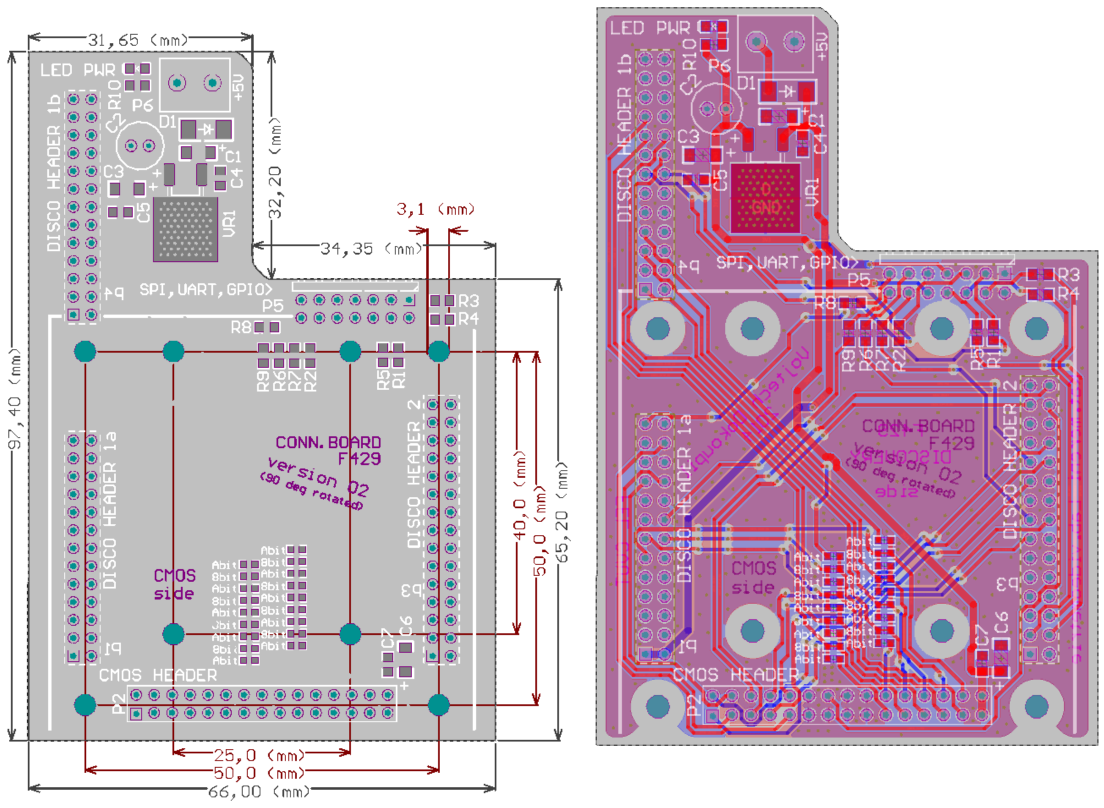


Fig. 56 - Mechanical (dimensions included) and routing drawing of CONN.BOARD F429 version2

Tab. 10 - Bill of materials for assembling CONN.BOARD F429 v1/v2

Bill of materials		
Schematic	**Value**	**Package**
R1 - R9	220 Ω (470 Ω)	0805
R10	470 Ω	0805
C1	10 µF/16 V	SMD A
C2	100 µF/16 V	Elyt TH radial, pitch 2.5mm
C3	10 µF/16 V	SMD A
C4	100 nF	0805
C5	100 nF	0805
C6	10 µF/16 V	SMD A
C7	100 nF	0805
D1	SUF 4007	MELF
P1	13x2	Header 2.54 mm F - straight
P4	13x2	Header 2.54 mm F - straight
P3	15x2	Header 2.54 mm F - straight
P2	6x2	Header 2.54 mm M - straight
P5	7x2	Header 2.54 mm M - 90° angular
P6	1x2	Screw terminal, pitch 5 mm
LED	RED/GREEN	0805
VR1	LF33	DPAK TO-252
J1-J18	0 Ω	Solder bridge/zero resistor *)

**)These bridges are used for changing DCMI data bus width from 8 to 10 bits or the other direction, for more information on this see fig. 7. Newly fabricated PCB has routed 8 bit version on board by default and none of J1-J18 has to be assembled.*

For better understanding I attached also schematics (see fig. 59). In schematics you can see power supply block, signal routing between specific header pins etc.

WARNING:

Power supply connected to 5V terminal (P6) must not be higher than 5 V. Higher voltage level can cause damage on Disco kit and other 5 V peripherals possibly attached! The voltage regulator (VR1) regulates 3V3 part of the design (CMOS sensor) but the 5V part has no additional regulator (except some Discovery kit protection on the kit itself)

BEST PRACTICE - ASSEMBLY:

I recommend soldering VR1 at first. Voltage regulator has big thermal pad and it needs to be soldered longer and probably with higher temperature. This could lead to displacement of previously soldered SMD components. Right after the voltage regulator I found convenient to attach P2, P5 and P6. These headers are soldered from the bottom side and longer soldering process on BOTTOM side can cause problems with components assembled on the TOP layer. Last remarkable note I recommend is connection to P1, P4 and P3 headers - in other words the headers which are connected to the Discovery kit. In order to eliminate possible mechanical tension and unwanted disconnections on the header pins in the future it is useful to connect Disco kit to the headers before soldering them. Then solder few pins to fix the position with Disco kit and after that for safety soldering of the rest pins the Discovery kit can be removed.

Fig. 57 and Fig. 58 show assembly plan for TOP board side of both versions of the CONN.BOARD F429. Only the headers described above are soldered from BOTTOM side (mirrored names and dashed lining of components positions).

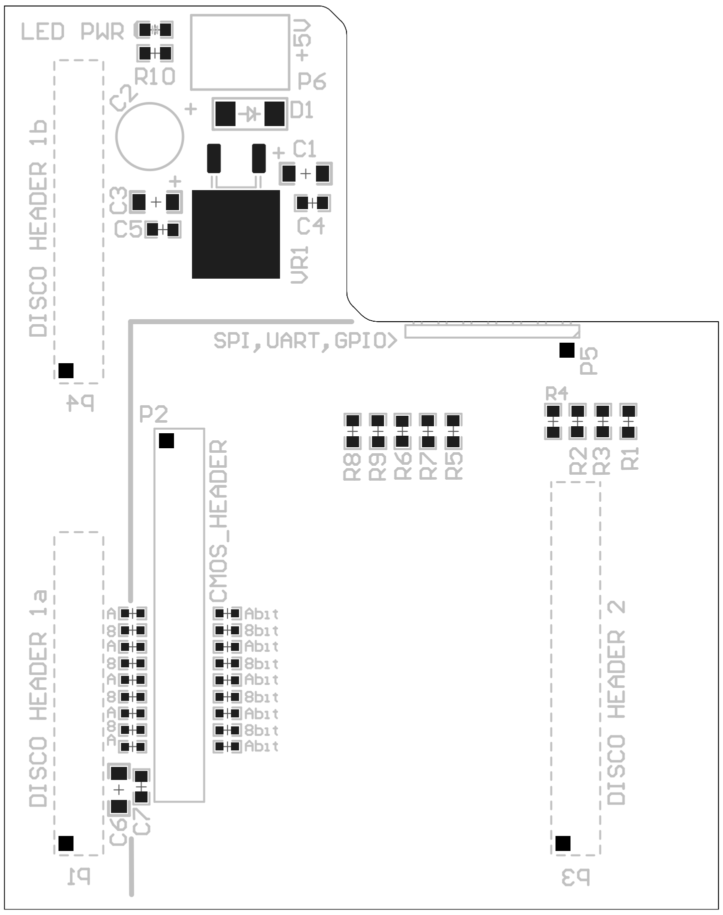


Fig. 57 - Assembly plan (TOP view) of CONN.BOARD F429 version1

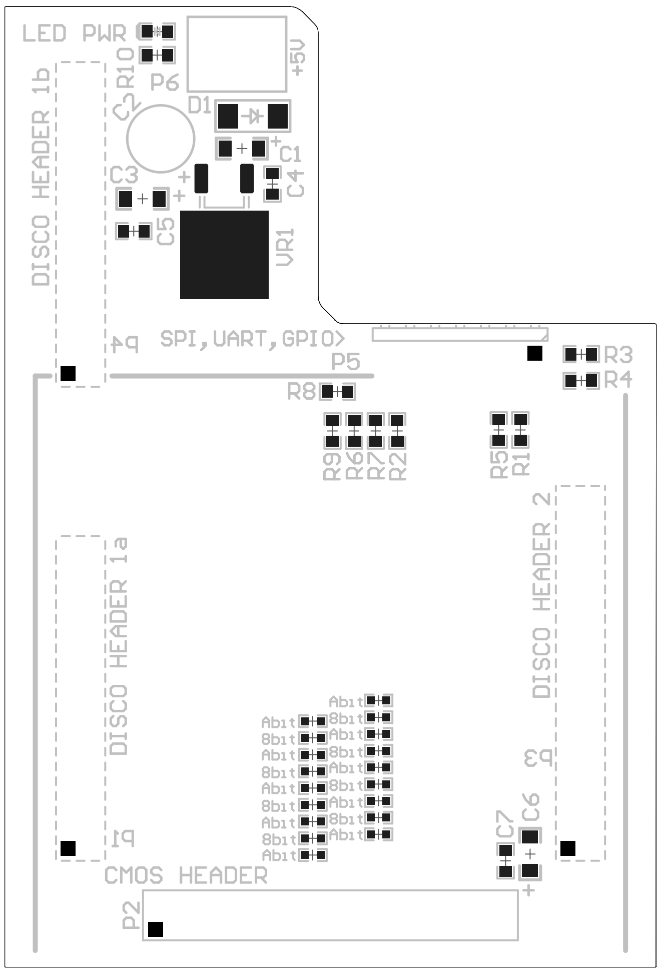


Fig. 58 - Assembly plan (TOP view) of CONN.BOARD F429 version2

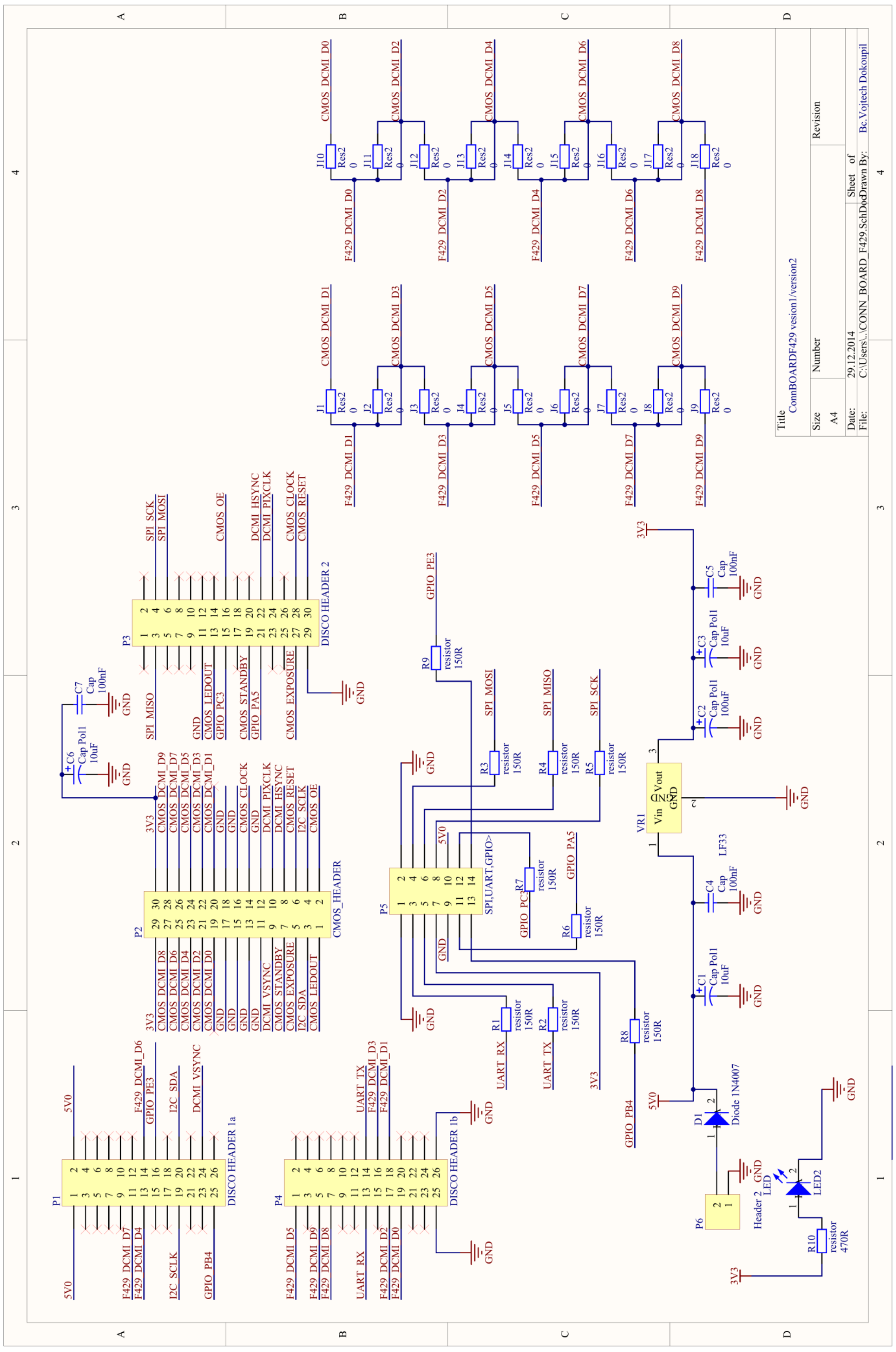


Fig. 59 - CONN.BOARD F429 v1/v2 schematics

Annex 2 – CS Mount Flange Basis Design

Designing holder block for CS mount ring:

I need to redesign holder block for fixing CS ring for CS lens attachment. The last version of this block used in Videomtery Lab is from the phenolic paper and I decided to change the shape and material according to the new possibilities. I wanted to try 3D printing of this base. At first I have to prepare easy 3D model which I did using the program *Design Spark Mechanical*. From this file I can easily generate a model in *.stl format suitable for 3D printing. I use then 3D printer in my job to print this prototype. The block is central symmetric and so there is only a front view with all inner and outer dimensions (fig. 60). I also attached a *.stl file on the data disk. Fig. 61 shows 3D model of this block. The Aluminium CS ring can be mounted directly into plastic but after my experience I suggest creating wider hole (wider than presented 24.6 mm in diameter).

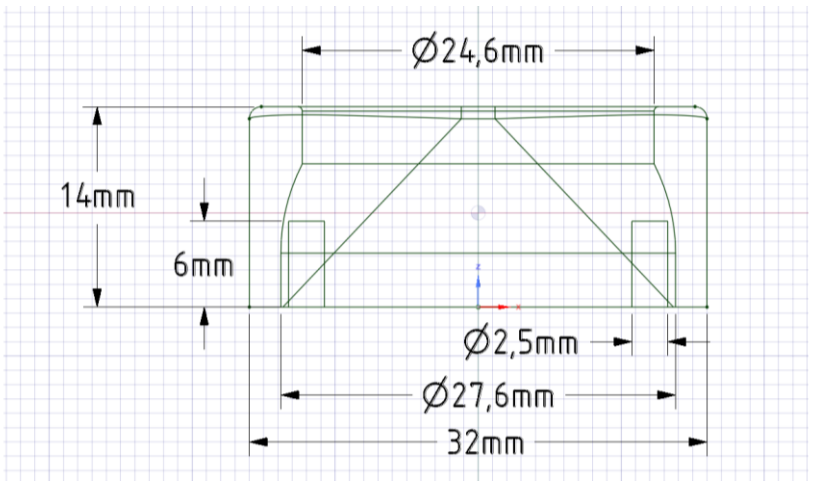


Fig. 60 - CS ring base dimensioning

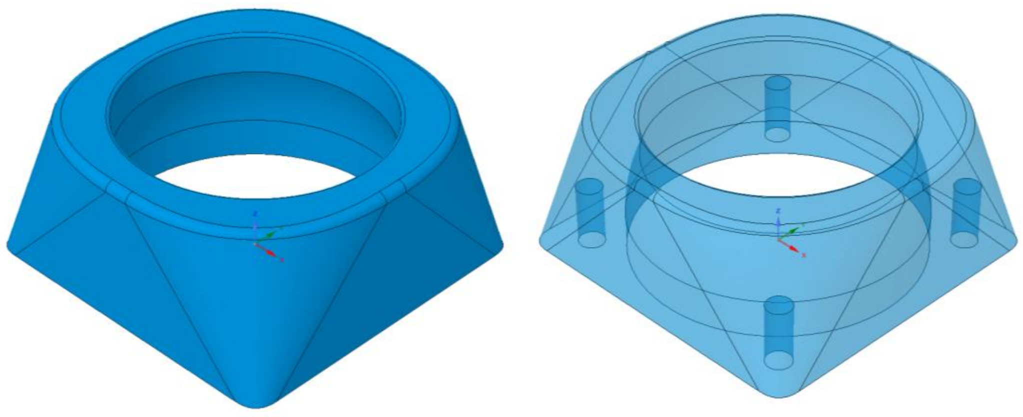


Fig. 61 - 3D filled and transparent model of CS ring base

Annex 3 – Photo Gallery

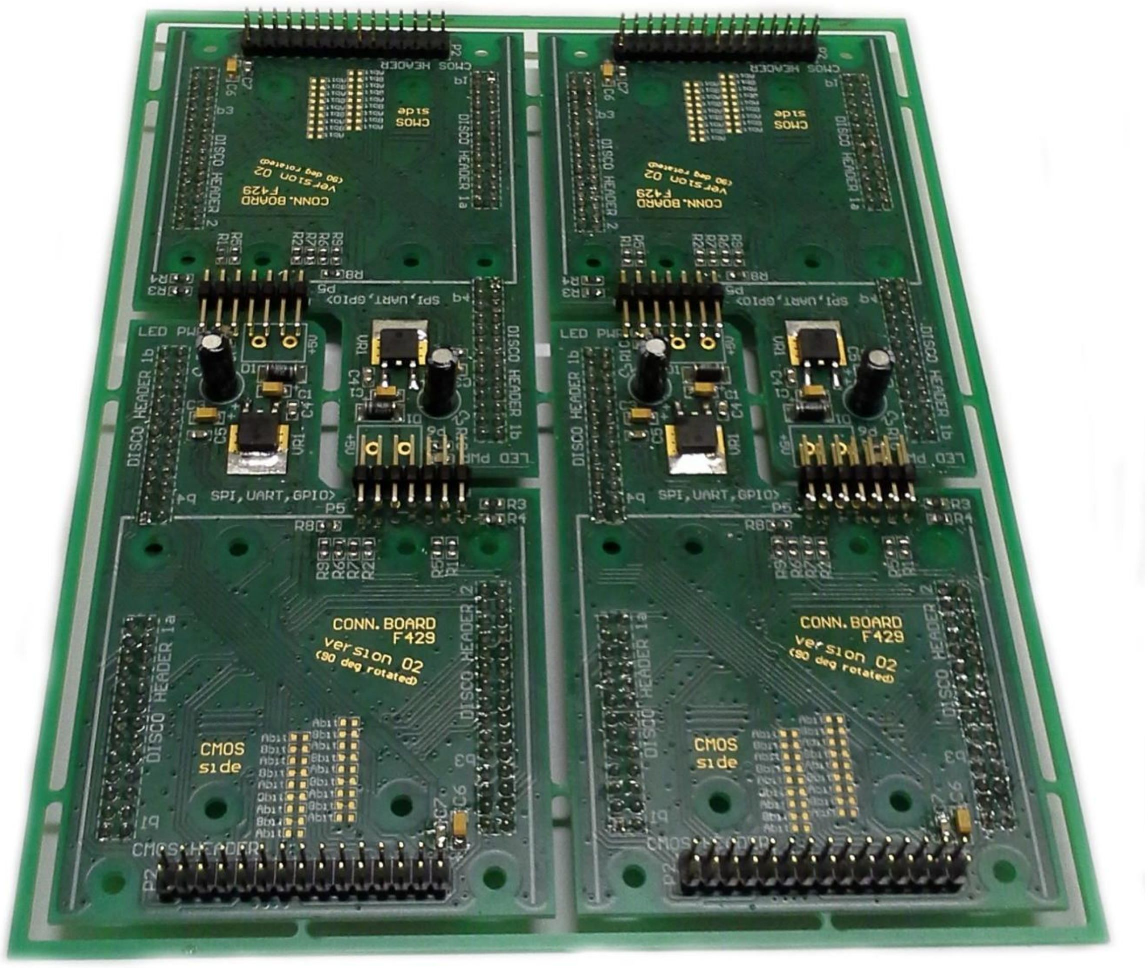


Fig. 62 - CONN.BOARDs F429 hand assembly (version 2)

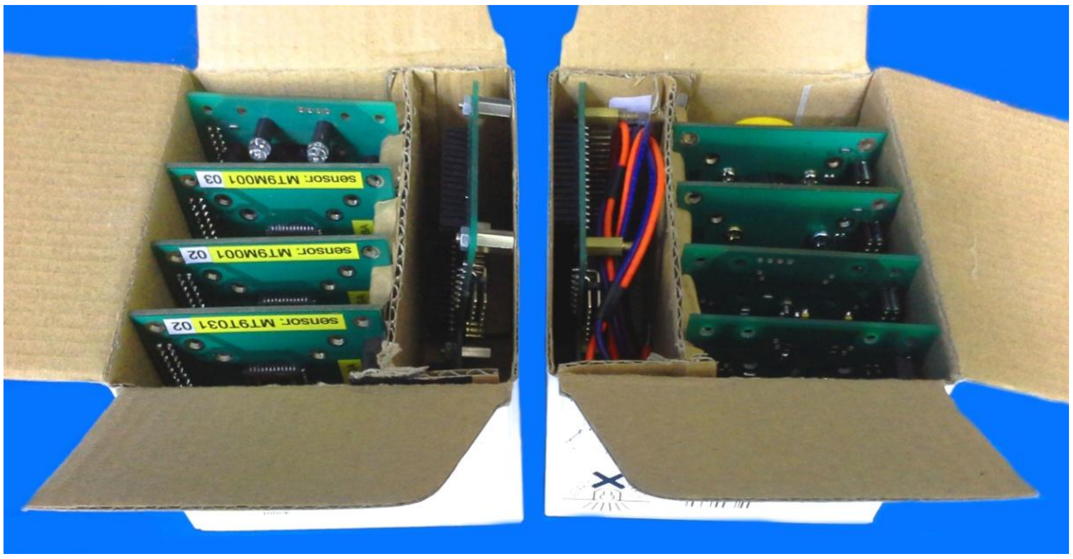


Fig. 63 - Transportation boxes for image sensor modules

Annex 4 – Attached CD Contents

- Diploma Thesis in PDF form
- Applications and Programs
  - Projects for MCU
    - STM32F429I Camera Application
    - STM32F429I Laser Height Measurement Application
    - STM32F429I RGB Compact Camera
    - STM32F429I Horizon Detection Application
    - STM32F4 Audio output demo
  - PC application
    - STM32F429 CMOS Camera Display
- Diploma Thesis Resources
- Additional materials