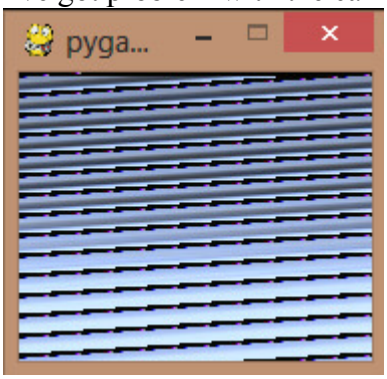
Hello,

I've got problem with the camera module. I reveive this picture:



It's changing if I cover the lens or turn into light.
I attach the main.c, please help me :)
I'd be very happy if you could find my mistake.
Thanks!

```
#include <stm32f4xx.h>
#include <stm32f4xx_i2c.h>
#include <stm32f4xx_usart.h>
#include "stm32f4_discovery.h"


#define SLAVE_ADDRESS 0x42 // the slave address (example)
#define DCMI_DR_ADDRESS       0x50050028


#define REG_GAIN       0x00    /* Gain lower 8 bits (rest in vref) */
#define REG_BLUE       0x01    /* blue gain */
#define REG_RED        0x02    /* red gain */
#define REG_VREF       0x03    /* Pieces of GAIN, VSTART, VSTOP */
#define REG_COM1       0x04    /* Control 1 */
#define COM1_CCIR656   0x40    /* CCIR656 enable */
#define REG_BAVE       0x05    /* U/B Average level */
#define REG_GbAVE      0x06    /* Y/Gb Average level */
#define REG_AECHH      0x07    /* AEC MS 5 bits */
#define REG_RAVE       0x08    /* V/R Average level */
#define REG_COM2       0x09    /* Control 2 */
#define COM2_SSLEEP    0x10    /* Soft sleep mode */
#define REG_PID        0x0a    /* Product ID MSB */
#define REG_VER        0x0b    /* Product ID LSB */
#define REG_COM3       0x0c    /* Control 3 */
#define COM3_SWAP      0x40    /* Byte swap */
#define COM3_SCALEEN   0x08    /* Enable scaling */
#define COM3_DCWEN     0x04    /* Enable downsamp/crop/window */
#define REG_COM4       0x0d    /* Control 4 */
#define REG_COM5       0x0e    /* All "reserved" */
#define REG_COM6       0x0f    /* Control 6 */
#define REG_AECH       0x10    /* More bits of AEC value */
#define REG_CLKRC      0x11    /* Clocl control */
#define CLK_EXT        0x40    /* Use external clock directly */
#define CLK_SCALE      0x3f    /* Mask for internal clock scale */
#define REG_COM7       0x12    /* Control 7 */
#define COM7_RESET     0x80    /* Register reset */
#define COM7_FMT_MASK  0x38
#define COM7_FMT_VGA   0x00
```

```
#define COM7_FMT_CIF    0x20    /* CIF format */
#define COM7_FMT_QVGA   0x10    /* QVGA format */
#define COM7_FMT_QCIF   0x08    /* QCIF format */
#define COM7_RGB        0x04    /* bits 0 and 2 - RGB format */
#define COM7_YUV        0x00    /* YUV */
#define COM7_BAYER      0x01    /* Bayer format */
#define COM7_PBAYER     0x05    /* "Processed bayer" */
#define REG_COM8        0x13    /* Control 8 */
#define COM8_FASTAEC    0x80    /* Enable fast AGC/AEC */
#define COM8_AECSTEP    0x40    /* Unlimited AEC step size */
#define COM8_BFILT      0x20    /* Band filter enable */
#define COM8_AGC        0x04    /* Auto gain enable */
#define COM8_AWB        0x02    /* White balance enable */
#define COM8_AEC        0x01    /* Auto exposure enable */
#define REG_COM9        0x14    /* Control 9  - gain ceiling */
#define REG_COM10       0x15    /* Control 10 */
#define COM10_HSYNC     0x40    /* HSYNC instead of HREF */
#define COM10_PCLK_HB   0x20    /* Suppress PCLK on horiz blank */
#define COM10_HREF_REV  0x08    /* Reverse HREF */
#define COM10_VS_LEAD   0x04    /* VSYNC on clock leading edge */
#define COM10_VS_NEG    0x02    /* VSYNC negative */
#define COM10_HS_NEG    0x01    /* HSYNC negative */
#define REG_HSTART      0x17    /* Horiz start high bits */
#define REG_HSTOP       0x18    /* Horiz stop high bits */
#define REG_VSTART      0x19    /* Vert start high bits */
#define REG_VSTOP       0x1a    /* Vert stop high bits */
#define REG_PSHFT       0x1b    /* Pixel delay after HREF */
#define REG_MIDH        0x1c    /* Manuf. ID high */
#define REG_MIDL        0x1d    /* Manuf. ID low */
#define REG_MVFP        0x1e    /* Mirror / vflip */
#define MVFP_MIRROR     0x20    /* Mirror image */
#define MVFP_FLIP       0x10    /* Vertical flip */
#define REG_AEW         0x24    /* AGC upper limit */
#define REG_AEB         0x25    /* AGC lower limit */
#define REG_VPT         0x26    /* AGC/AEC fast mode op region */
#define REG_HSYST       0x30    /* HSYNC rising edge delay */
#define REG_HSYEN       0x31    /* HSYNC falling edge delay */
#define REG_HREF        0x32    /* HREF pieces */
#define REG_TSLB        0x3a    /* lots of stuff */
#define TSLB_YLAST      0x04    /* UYVY or VYUY - see com13 */
#define REG_COM11       0x3b    /* Control 11 */
#define COM11_NIGHT     0x80    /* NIght mode enable */
#define COM11_NMFR      0x60    /* Two bit NM frame rate */
#define COM11_HZAUTO    0x10    /* Auto detect 50/60 Hz */
#define COM11_50HZ      0x08    /* Manual 50Hz select */
#define COM11_EXP       0x02
#define REG_COM12       0x3c    /* Control 12 */
#define COM12_HREF      0x80    /* HREF always */
#define REG_COM13       0x3d    /* Control 13 */
#define COM13_GAMMA     0x80    /* Gamma enable */
#define COM13_UVSAT     0x40    /* UV saturation auto adjustment */
#define COM13_UVSWAP    0x01    /* V before U - w/TSLB */
#define REG_COM14       0x3e    /* Control 14 */
#define COM14_DCWEN     0x10    /* DCW/PCLK-scale enable */
#define REG_EDGE        0x3f    /* Edge enhancement factor */
#define REG_COM15       0x40    /* Control 15 */
#define COM15_R10F0     0x00    /* Data range 10 to F0 */
#define COM15_R01FE     0x80    /*            01 to FE */
```

```
#define COM15_R00FF     0xc0    /*          00 to FF */
#define COM15_RGB565    0x10    /* RGB565 output */
#define COM15_RGB555    0x30    /* RGB555 output */
#define REG_COM16       0x41    /* Control 16 */
#define COM16_AWBGAIN   0x08    /* AWB gain enable */
#define REG_COM17       0x42    /* Control 17 */
#define COM17_AECWIN    0xc0    /* AEC window - must match COM4 */
#define COM17_CBAR      0x08    /* DSP Color bar */
#define REG_CMATRIX_BASE 0x4f
#define CMATRIX_LEN 6
#define REG_CMATRIX_SIGN 0x58
#define REG_BRIGHT      0x55    /* Brightness */
#define REG_CONTRAS     0x56    /* Contrast control */
#define REG_GFIX        0x69    /* Fix gain control */
#define REG_REG76       0x76    /* OV's name */
#define R76_BLKPCOR     0x80    /* Black pixel correction enable */
#define R76_WHTPCOR     0x40    /* White pixel correction enable */
#define REG_RGB444      0x8c    /* RGB 444 control */
#define R444_ENABLE     0x02    /* Turn on RGB444, overrides 5x5 */
#define R444_RGBX       0x01    /* Empty nibble at end */
#define REG_HAECC1      0x9f    /* Hist AEC/AGC control 1 */
#define REG_HAECC2      0xa0    /* Hist AEC/AGC control 2 */
#define REG_BD50MAX     0xa5    /* 50hz banding step limit */
#define REG_HAECC3      0xa6    /* Hist AEC/AGC control 3 */
#define REG_HAECC4      0xa7    /* Hist AEC/AGC control 4 */
#define REG_HAECC5      0xa8    /* Hist AEC/AGC control 5 */
#define REG_HAECC6      0xa9    /* Hist AEC/AGC control 6 */
#define REG_HAECC7      0xaa    /* Hist AEC/AGC control 7 */
#define REG_BD60MAX     0xab    /* 60hz banding step limit */


#define OV7660_CTL_GAIN 0x00
#define OV7660_CTL_BLUE 0x01
#define OV7660_CTL_RED 0x02
#define OV7660_CTL_VREF 0x03
#define OV7660_CTL_COM1 0x04
#define OV7660_CTL_BAVE 0x05
#define OV7660_CTL_GEAVE 0x06
#define OV7660_CTL_AECHH 0x07
#define OV7660_CTL_RAVE 0x08
#define OV7660_CTL_COM2 0x09
#define OV7660_CTL_PID 0x0a
#define OV7660_CTL_VER 0x0b
#define OV7660_CTL_COM3 0x0c
#define OV7660_CTL_COM4 0x0d
#define OV7660_CTL_COM5 0x0e
#define OV7660_CTL_COM6 0x0f
#define OV7660_CTL_AECH 0x10
#define OV7660_CTL_CLKRC 0x11
#define OV7660_CTL_COM7 0x12
#define OV7660_CTL_COM8 0x13
#define OV7660_CTL_COM9 0x14
#define OV7660_CTL_COM10 0x15
/* RSVD 0x16 is Reserved */
#define OV7660_CTL_HSTART 0x17
#define OV7660_CTL_HSTOP 0x18
#define OV7660_CTL_VSTRT 0x19
#define OV7660_CTL_VSTOP 0x1a
```

```
#define OV7660_CTL_PSHFT 0x1b
#define OV7660_CTL_MIDH 0x1c
#define OV7660_CTL_MIDL 0x1d
#define OV7660_CTL_MVFP 0x1e
#define OV7660_CTL_LAEC 0x1f
#define OV7660_CTL_BOS 0x20
#define OV7660_CTL_GBOS 0x21
#define OV7660_CTL_GROS 0x22
#define OV7660_CTL_ROS 0x23
#define OV7660_CTL_AEW 0x24
#define OV7660_CTL_AEB 0x25
#define OV7660_CTL_VPT 0x26
#define OV7660_CTL_BBIAS 0x27
#define OV7660_CTL_GbBIAS 0x28
/* RSVD 0x29 is Reserved */
#define OV7660_CTL_EXHCH 0x2a
#define OV7660_CTL_EXHCL 0x2b
#define OV7660_CTL_RBIAS 0x2c
#define OV7660_CTL_ADVFL 0x2d
#define OV7660_CTL_ADVFH 0x2e
#define OV7660_CTL_YAVE 0x2f
#define OV7660_CTL_HSYST 0x30
#define OV7660_CTL_HSYEN 0x31
#define OV7660_CTL_HREF 0x32
#define OV7660_CTL_CHLF 0x33
#define OV7660_CTL_ARBLM 0x34
/* RSVD 0x35 is Reserved */
/* RSVD 0x36 is Reserved */
#define OV7660_CTL_ADC 0x37
#define OV7660_CTL_ACOM 0x38
#define OV7660_CTL_OFON 0x39
#define OV7660_CTL_TSLB 0x3a
#define OV7660_CTL_COM11 0x3b
#define OV7660_CTL_COM12 0x3c
#define OV7660_CTL_COM13 0x3d
#define OV7660_CTL_COM14 0x3e
#define OV7660_CTL_EDGE 0x3f
#define OV7660_CTL_COM15 0x40
#define OV7660_CTL_COM16 0x41
#define OV7660_CTL_COM17 0x42
/* RSVD 0x43 is Reserved */
/* RSVD 0x44 is Reserved */
/* RSVD 0x45 is Reserved */
/* RSVD 0x46 is Reserved */
/* RSVD 0x47 is Reserved */
/* RSVD 0x48 is Reserved */
/* RSVD 0x49 is Reserved */
/* RSVD 0x4a is Reserved */
/* RSVD 0x4b is Reserved */
/* RSVD 0x4c is Reserved */
/* RSVD 0x4d is Reserved */
/* RSVD 0x4e is Reserved */
#define OV7660_CTL_MTX1 0x4f
#define OV7660_CTL_MTX2 0x50
#define OV7660_CTL_MTX3 0x51
#define OV7660_CTL_MTX4 0x52
#define OV7660_CTL_MTX5 0x53
#define OV7660_CTL_MTX6 0x54
```

```
#define OV7660_CTL_MTX7 0x55
#define OV7660_CTL_MTX8 0x56
#define OV7660_CTL_MTX9 0x57
#define OV7660_CTL_MTXS 0x58
/* RSVD 0x59 is Reserved */
/* RSVD 0x60 is Reserved */
/* RSVD 0x61 is Reserved */
#define OV7660_CTL_LCC1 0x62
#define OV7660_CTL_LCC2 0x63
#define OV7660_CTL_LCC3 0x64
#define OV7660_CTL_LCC4 0x65
#define OV7660_CTL_LCC5 0x66
#define OV7660_CTL_MANU 0x67
#define OV7660_CTL_MANV 0x68
#define OV7660_CTL_HV 0x69
#define OV7660_CTL_GGAIN 0x6a
#define OV7660_CTL_DBLV 0x6b
/* 6c-7b GSP */
/* 7c-8a GST */
/* RSVD 0x8b is Reserved */
/* RSVD 0x8c is Reserved */
/* RSVD 0x8d is Reserved */
/* RSVD 0x8e is Reserved */
/* RSVD 0x8f is Reserved */
/* RSVD 0x90 is Reserved */
/* RSVD 0x91 is Reserved */
#define OV7660_CTL_DM_LNL 0x92
#define OV7660_CTL_DM_LNH 0x93
/* RSVD 0x94 is Reserved */
/* RSVD 0x95 is Reserved */
/* RSVD 0x96 is Reserved */
/* RSVD 0x97 is Reserved */
/* RSVD 0x98 is Reserved */
/* RSVD 0x99 is Reserved */
/* RSVD 0x9a is Reserved */
/* RSVD 0x9b is Reserved */
/* RSVD 0x9c is Reserved */
#define OV7660_CTL_BD50ST 0x9d
#define OV7660_CTL_BD60ST 0x9e
void TimingDelay_Decrement(void);
void Delay(uint32_t nTime);
unsigned long int i;
static __IO uint32_t TimingDelay;
uint8_t temp;
uint8_t received_data[2];
uint8_t frame_buffer[176*144*2];


void init_usart(void){
GPIO_InitTypeDef GPIO_InitStructure;
USART_InitTypeDef USART_InitStructure;


/* enable peripheral clock for USART2 */
RCC_APB1PeriphClockCmd(RCC_APB1Periph_USART2, ENABLE);
```

```
/* GPIOA clock enable */
RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOA, ENABLE);


/* GPIOA Configuration:  USART2 TX on PA2 */
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_2;
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF;
GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_UP ;
GPIO_Init(GPIOA, &GPIO_InitStructure);


/* Connect USART2 pins to AF2 */
// TX = PA2
GPIO_PinAFConfig(GPIOA, GPIO_PinSource2, GPIO_AF_USART2);


USART_InitStructure.USART_BaudRate = 115200 ;//115200;//460800;
USART_InitStructure.USART_WordLength = USART_WordLength_8b;
USART_InitStructure.USART_StopBits = USART_StopBits_1;
USART_InitStructure.USART_Parity = USART_Parity_No;
USART_InitStructure.USART_HardwareFlowControl = USART_HardwareFlowControl_None;
USART_InitStructure.USART_Mode = USART_Mode_Tx;
USART_Init(USART2, &USART_InitStructure);


USART_Cmd(USART2, ENABLE); // enable USART2
}


void init_DCMI(void){

GPIO_InitTypeDef GPIO_InitStructure;
DCMI_InitTypeDef DCMI_InitStructure;
  DMA_InitTypeDef  DMA_InitStructure;


RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOA | RCC_AHB1Periph_GPIOB |
RCC_AHB1Periph_GPIOC, ENABLE);

/* Connect DCMI pins to AF13 */
  GPIO_PinAFConfig(GPIOA, GPIO_PinSource6, GPIO_AF_DCMI); //pclk


  GPIO_PinAFConfig(GPIOA, GPIO_PinSource4, GPIO_AF_DCMI); //href

  GPIO_PinAFConfig(GPIOB, GPIO_PinSource6, GPIO_AF_DCMI); //D5
  GPIO_PinAFConfig(GPIOB, GPIO_PinSource7, GPIO_AF_DCMI); //vsync
  GPIO_PinAFConfig(GPIOB, GPIO_PinSource8, GPIO_AF_DCMI); //D6
  GPIO_PinAFConfig(GPIOB, GPIO_PinSource9, GPIO_AF_DCMI); //D7

  GPIO_PinAFConfig(GPIOC, GPIO_PinSource6, GPIO_AF_DCMI); //D0
  GPIO_PinAFConfig(GPIOC, GPIO_PinSource7, GPIO_AF_DCMI); //D1
  GPIO_PinAFConfig(GPIOC, GPIO_PinSource8, GPIO_AF_DCMI); //D2
  GPIO_PinAFConfig(GPIOC, GPIO_PinSource9, GPIO_AF_DCMI); //D3
  GPIO_PinAFConfig(GPIOC, GPIO_PinSource11, GPIO_AF_DCMI);//D4
```

```
  /* DCMI GPIO configuration */
    /* D0..D4 */
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_6 | GPIO_Pin_7 | GPIO_Pin_8 |
                                  GPIO_Pin_9 | GPIO_Pin_11;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_100MHz;
    GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
    GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_UP;
    GPIO_Init(GPIOC, &GPIO_InitStructure);


    /* D5..D7, VSYNC) */
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_6 | GPIO_Pin_7 | GPIO_Pin_8 | GPIO_Pin_9;
    GPIO_Init(GPIOB, &GPIO_InitStructure);


    /* PCLK(PA6), HSYNC */
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_4 | GPIO_Pin_6;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF;
    GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
    GPIO_Init(GPIOA, &GPIO_InitStructure);

    /*** Configures the DCMI to interface with the OV9655 camera module ***/
    /* Enable DCMI clock */
    RCC_AHB2PeriphClockCmd(RCC_AHB2Periph_DCMI, ENABLE);


    DCMI_DeInit();


    /* DCMI configuration */
    DCMI_InitStructure.DCMI_CaptureMode =
  DCMI_CaptureMode_SnapShot;//DCMI_CaptureMode_SnapShot;//DCMI_CaptureMode_Continuous;
    DCMI_InitStructure.DCMI_SynchroMode = DCMI_SynchroMode_Hardware;
    DCMI_InitStructure.DCMI_PCKPolarity = DCMI_PCKPolarity_Rising;
    DCMI_InitStructure.DCMI_VSPolarity = DCMI_VSPolarity_High;
    DCMI_InitStructure.DCMI_HSPolarity = DCMI_HSPolarity_Low;
    DCMI_InitStructure.DCMI_CaptureRate = DCMI_CaptureRate_All_Frame;
    DCMI_InitStructure.DCMI_ExtendedDataMode = DCMI_ExtendedDataMode_8b;


    /* DCMI configuration */
    DCMI_Init(&DCMI_InitStructure);
  //DCMI_JPEGCmd(ENABLE);

  /* Configures the DMA2 to transfer Data from DCMI */
    /* Enable DMA2 clock */
    RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_DMA2, ENABLE);

    /* DMA2 Stream1 Configuration */
    DMA_DeInit(DMA2_Stream1);


  DMA_InitStructure.DMA_Channel = DMA_Channel_1;
  DMA_InitStructure.DMA_PeripheralBaseAddr =  (uint32_t)(DCMI_BASE + 0x28);
  DMA_InitStructure.DMA_Memory0BaseAddr =  (uint32_t)frame_buffer;
  DMA_InitStructure.DMA_DIR = DMA_DIR_PeripheralToMemory;
```

```
DMA_InitStructure.DMA_BufferSize = sizeof(frame_buffer);///sizeof(uint8_t);//4;//0x9600 = Image size /4
DMA_InitStructure.DMA_PeripheralInc = DMA_PeripheralInc_Disable;
DMA_InitStructure.DMA_MemoryInc = DMA_MemoryInc_Enable;
DMA_InitStructure.DMA_PeripheralDataSize = DMA_PeripheralDataSize_Word;
DMA_InitStructure.DMA_MemoryDataSize = DMA_MemoryDataSize_Word;
DMA_InitStructure.DMA_Mode =
DMA_Mode_Normal;//DMA_Mode_Circular;//DCMI_CaptureMode_SnapShot;//// First take one picture
DMA_InitStructure.DMA_Priority = DMA_Priority_High;
DMA_InitStructure.DMA_FIFOMode = DMA_FIFOMode_Enable;
DMA_InitStructure.DMA_FIFOThreshold = DMA_FIFOThreshold_Full;
DMA_InitStructure.DMA_MemoryBurst = DMA_MemoryBurst_Single;
DMA_InitStructure.DMA_PeripheralBurst = DMA_PeripheralBurst_Single;


  /* DMA2 IRQ channel Configuration */
  DMA_Init(DMA2_Stream1, &DMA_InitStructure);
}


void init_I2C2(void){


GPIO_InitTypeDef GPIO_InitStruct;
I2C_InitTypeDef I2C_InitStruct;


// enable APB1 peripheral clock for I2C1
RCC_APB1PeriphClockCmd(RCC_APB1Periph_I2C2, ENABLE);
// enable clock for SCL and SDA pins
RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOA | RCC_AHB1Periph_GPIOB, ENABLE);


// Connect MCO1 pin to AF0
  // Connect to AF0 is default after reset
  GPIO_PinAFConfig(GPIOA, GPIO_PinSource8, GPIO_AF_MCO);

  // Configure MCO1 pin(PA8) in alternate function
  GPIO_InitStruct.GPIO_Pin = GPIO_Pin_8;
  GPIO_InitStruct.GPIO_Speed = GPIO_Speed_100MHz;
  GPIO_InitStruct.GPIO_Mode = GPIO_Mode_AF;
  GPIO_InitStruct.GPIO_OType = GPIO_OType_PP;
  GPIO_InitStruct.GPIO_PuPd = GPIO_PuPd_UP;
  GPIO_Init(GPIOA, &GPIO_InitStruct);

  // HSI clock selected to output on MCO1 pin(PA8)
  RCC_MCO1Config(RCC_MCO1Source_HSI, RCC_MCO1Div_1);

/* setup SCL and SDA pins
* You can connect I2C1 to two different
* pairs of pins:
* 1. SCL on PB6 and SDA on PB7
* 2. SCL on PB8 and SDA on PB9
*/
GPIO_InitStruct.GPIO_Pin = GPIO_Pin_10 | GPIO_Pin_11; // we are going to use PB6 and PB7
GPIO_InitStruct.GPIO_Mode = GPIO_Mode_AF; // set pins to alternate function
GPIO_InitStruct.GPIO_Speed = GPIO_Speed_50MHz; // set GPIO speed
GPIO_InitStruct.GPIO_OType = GPIO_OType_OD; // set output to open drain --> the line has to be only
pulled low, not driven high
```

```
GPIO_InitStruct.GPIO_PuPd = GPIO_PuPd_UP; // enable pull up resistors
GPIO_Init(GPIOB, &GPIO_InitStruct); // init GPIOB


// Connect I2C1 pins to AF
GPIO_PinAFConfig(GPIOB, GPIO_PinSource10, GPIO_AF_I2C2); // SCL
GPIO_PinAFConfig(GPIOB, GPIO_PinSource11, GPIO_AF_I2C2); // SDA


// configure I2C1
I2C_InitStruct.I2C_ClockSpeed = 100000; // 100kHz
I2C_InitStruct.I2C_Mode = I2C_Mode_I2C; // I2C mode
I2C_InitStruct.I2C_DutyCycle = I2C_DutyCycle_2; // 50% duty cycle --> standard
I2C_InitStruct.I2C_OwnAddress1 = 0x00; // own address, not relevant in master mode
I2C_InitStruct.I2C_Ack = I2C_Ack_Disable; // disable acknowledge when reading (can be changed later on)
I2C_InitStruct.I2C_AcknowledgedAddress = I2C_AcknowledgedAddress_7bit; // set address length to 7 bit
addresses
I2C_Init(I2C2, &I2C_InitStruct); // init I2C1


// enable I2C1
I2C_Cmd(I2C2, ENABLE);
}


/* This function issues a start condition and
 * transmits the slave address + R/W bit
 *
 * Parameters:
 * I2Cx --> the I2C peripheral e.g. I2C1
 * address --> the 7 bit slave address
 * direction --> the tranmission direction can be:
 * I2C_Direction_Tranmitter for Master transmitter mode
 * I2C_Direction_Receiver for Master receiver
 */
void I2C_start(I2C_TypeDef* I2Cx, uint8_t address, uint8_t direction){
// wait until I2C1 is not busy anymore
while(I2C_GetFlagStatus(I2Cx, I2C_FLAG_BUSY));

// Send I2C1 START condition
I2C_GenerateSTART(I2Cx, ENABLE);


// wait for I2C1 EV5 --> Slave has acknowledged start condition
while(!I2C_CheckEvent(I2Cx, I2C_EVENT_MASTER_MODE_SELECT));


// Send slave Address for write
I2C_Send7bitAddress(I2Cx, address, direction);


/* wait for I2C1 EV6, check if
* either Slave has acknowledged Master transmitter or
* Master receiver mode, depending on the transmission
* direction
*/
if(direction == I2C_Direction_Transmitter){
while(!I2C_CheckEvent(I2Cx, I2C_EVENT_MASTER_TRANSMITTER_MODE_SELECTED));
```

```
}
else if(direction == I2C_Direction_Receiver){
while(!I2C_CheckEvent(I2Cx, I2C_EVENT_MASTER_RECEIVER_MODE_SELECTED));
}
}


/* This function transmits one byte to the slave device
 * Parameters:
 * I2Cx --> the I2C peripheral e.g. I2C1
 * data --> the data byte to be transmitted
 */
void I2C_write(I2C_TypeDef* I2Cx, uint8_t data){
I2C_SendData(I2Cx, data);
// wait for I2C1 EV8_2 --> byte has been transmitted
while(!I2C_CheckEvent(I2Cx, I2C_EVENT_MASTER_BYTE_TRANSMITTED));
}

/* This function reads one byte from the slave device
 * and acknowledges the byte (requests another byte)
 */
uint8_t I2C_read_ack(I2C_TypeDef* I2Cx){
uint8_t data;
// enable acknowledge of recieved data
I2C_AcknowledgeConfig(I2Cx, ENABLE);
// wait until one byte has been received
while( !I2C_CheckEvent(I2Cx, I2C_EVENT_MASTER_BYTE_RECEIVED) );
// read data from I2C data register and return data byte
data = I2C_ReceiveData(I2Cx);
return data;
}


/* This function reads one byte from the slave device
 * and doesn't acknowledge the recieved data
 */
uint8_t I2C_read_nack(I2C_TypeDef* I2Cx){
uint8_t data;
// disabe acknowledge of received data
I2C_AcknowledgeConfig(I2Cx, DISABLE);
// wait until one byte has been received
while( !I2C_CheckEvent(I2Cx, I2C_EVENT_MASTER_BYTE_RECEIVED) );
// read data from I2C data register and return data byte
data = I2C_ReceiveData(I2Cx);
return data;
}


/* This funtion issues a stop condition and therefore
 * releases the bus
 */
void I2C_stop(I2C_TypeDef* I2Cx){
// Send I2C1 STOP Condition
I2C_GenerateSTOP(I2Cx, ENABLE);
}


uint8_t I2C_writereg(uint8_t reg, uint8_t data){
```

```
uint8_t tmp;

I2C_start(I2C2, SLAVE_ADDRESS, I2C_Direction_Transmitter); // start a transmission in Master
transmitter mode
I2C_write(I2C2, reg); // write one byte to the slave
Delay(100);
I2C_write(I2C2, data); // write one byte to the slave
I2C_stop(I2C2); // stop the transmission

Delay(100);

I2C_start(I2C2, SLAVE_ADDRESS, I2C_Direction_Receiver); // start a transmission in Master receiver
mode
tmp = I2C_read_nack(I2C2);
I2C_stop(I2C2); // stop the transmission

Delay(100);

return tmp;
}


uint8_t I2C_readreg(uint8_t reg){
uint8_t tmp;

I2C_start(I2C2, SLAVE_ADDRESS, I2C_Direction_Transmitter); // start a transmission in Master
transmitter mode
I2C_write(I2C2, reg); // write one byte to the slave
I2C_stop(I2C2); // stop the transmission

Delay(100);

I2C_start(I2C2, SLAVE_ADDRESS, I2C_Direction_Receiver); // start a transmission in Master receiver
mode
tmp = I2C_read_nack(I2C2);
I2C_stop(I2C2); // stop the transmission


Delay(100);

return tmp;
}


void init_OV7660(void){
//   I2C_writereg(0x12, 0x80);
//   I2C_writereg(0x12, 0x00);


// // /* Configuration for QCIF format */
// //
// // // COM3 register: Enable format scaling
// // temp = I2C_readreg(0x0C);
// // I2C_writereg(0x0C, temp | 0x08);
// //
// // // COM7 register: Select QCIF format 0x08
//   temp = I2C_readreg(0x12);
//   I2C_writereg(0x12, 0x08);
```

```
// //
// //   I2C_writereg(REG_TSLB, 0x04);
// //
// //   I2C_writereg(0x11, 0x00);
// // I2C_writereg(0x6B, 0x20);
// //
// // //I2C_writereg(REG_COM7, 0x04); /* output format: rgb */


// I2C_writereg(REG_RGB444, 0x00); /* disable RGB444 */
// //
// I2C_writereg(REG_COM15, 0xD0); /* set RGB565 */
  I2C_writereg(OV7660_CTL_COM7, 0x80);
  I2C_writereg(OV7660_CTL_COM5, 0x80);
/* OV7660 Wakeup */
/* COM4 is Reserved : using default value 0x40 OR windows driver value 0x08 */
I2C_writereg(OV7660_CTL_COM4, 0x08);
/* Enable HREF at optical black, Use optical black line as BLC signal
   Reset all timing when format changes, Enable ADBLC option */
I2C_writereg(OV7660_CTL_COM6, 0xc3);
/* windows 0x00, default 0x00, trying 0x60 to enable CCIR656 format */
//I2C_writereg(OV7660_CTL_COM1, 0xc3);
/* default is 0x40, windows sets it to 0x00 */
I2C_writereg(OV7660_CTL_AECH, 0x40);
/* default is 0x00, windows sets it to 0x40 */
I2C_writereg(OV7660_CTL_CLKRC, 0x40);
/* Set O/P format - RGB Selection, Set O/P format Raw RGB */ //0x05
I2C_writereg(OV7660_CTL_COM7, 0x08);
/* default is 0x8f, windows used 0xf8 */
/* Enable fast AGC/AEC algorithm, AEC - Step size limit = 1/16 x AEC */
/* Banding & Reserved are disabled. AGC, AEC enabled, 0x85 */
I2C_writereg(OV7660_CTL_COM8, 0xb8);
/* video appears jagged w/o these  ADC writes */
I2C_writereg(OV7660_CTL_ADC, 0x0f);
I2C_writereg(OV7660_CTL_ACOM, 0x02);
I2C_writereg(OV7660_CTL_OFON, 0x43);
/* video appears jagged w/o this write */
/* Default 0x0c sets format to uYvY, Windows driver 0x00 sets format to YuYv */
I2C_writereg(OV7660_CTL_TSLB, 0x00);
/* Not doing this write makes the video look green */
/* Manual Banding Filter MSB , set B & R channel pre-gain */
I2C_writereg(OV7660_CTL_HV, 0x90);
/*  No video stream w/o these ADVFL/ADVFH write totally black */
I2C_writereg(OV7660_CTL_ADVFL, 0xf6);
  I2C_writereg(OV7660_CTL_ADVFH, 0x0b);
/* Setting BLUE to 0x78; RED to 0x78 to get natural colors in artificial light */
I2C_writereg(OV7660_CTL_BLUE, 0x78);
/* Setting RED to 0x50 to get natural colors in natural light */
I2C_writereg(OV7660_CTL_RED,  0x50);
}
int main(void){
NVIC_InitTypeDef NVIC_InitStructure;

SysTick_Config(SystemCoreClock/1000);
init_usart();
init_I2C2(); // initialize I2C peripheral
init_OV7660();
init_DCMI();
```

```
/* Enable DMA2 stream 1 and DCMI interface then start image capture */


NVIC_PriorityGroupConfig(NVIC_PriorityGroup_1);
NVIC_InitStructure.NVIC_IRQChannel = DCMI_IRQn;
  NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 1;
  NVIC_InitStructure.NVIC_IRQChannelSubPriority = 1;
  NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
  NVIC_Init(&NVIC_InitStructure);
  DCMI_ITConfig(DCMI_IT_VSYNC, ENABLE);
  DCMI_ITConfig(DCMI_IT_LINE, ENABLE);
  DCMI_ITConfig(DCMI_IT_FRAME, ENABLE);
  DCMI_ITConfig(DCMI_IT_OVF, ENABLE);
  DCMI_ITConfig(DCMI_IT_ERR, ENABLE);

  DMA_Cmd(DMA2_Stream1, ENABLE);
  DCMI_Cmd(ENABLE);


  /* Insert 100ms delay: wait 100ms */
  Delay(100);


  DCMI_CaptureCmd(ENABLE);

while(1){


STM_EVAL_LEDInit(LED4);
STM_EVAL_LEDOn(LED4);
STM_EVAL_LEDInit(LED5);
STM_EVAL_LEDOn(LED5);


while(1){
Delay(0x000FF);
//USART_SendData(USART2, 'h');
STM_EVAL_LEDToggle(LED4);
}
}
}


/**
  * @brief  Inserts a delay time.
  * @param  nTime: specifies the delay time length, in milliseconds
  * @retval None
  */
void Delay(uint32_t nTime)
{
  TimingDelay = nTime;


  while(TimingDelay != 0);


}
```

```
/**
  * @brief  Decrements the TimingDelay variable.
  * @param  None
  * @retval None
  */
void TimingDelay_Decrement(void)
{
  if (TimingDelay != 0x00)
  {
    TimingDelay--;
  }
}


void SendPicture(void)
{
while(USART_GetFlagStatus(USART2, USART_FLAG_TXE) == RESET);
USART_SendData(USART2, 'O');
while(USART_GetFlagStatus(USART2, USART_FLAG_TXE) == RESET);
USART_SendData(USART2, 'K');

for(i=0;i<50688;i++)
{
while(USART_GetFlagStatus(USART2, USART_FLAG_TXE) == RESET);
USART_SendData(USART2, frame_buffer[i]);
}
while(1);
}
```