Hi all

I 'm using Aptina MT9P031 image sensor with the STM32F7 Discovery. I have a problem.

I did connected the camera to stm32f7 DCMI connector. I've setting up the camera with I2C
This is my MT9P031 Library. Please see the MT9P031_Configuration function

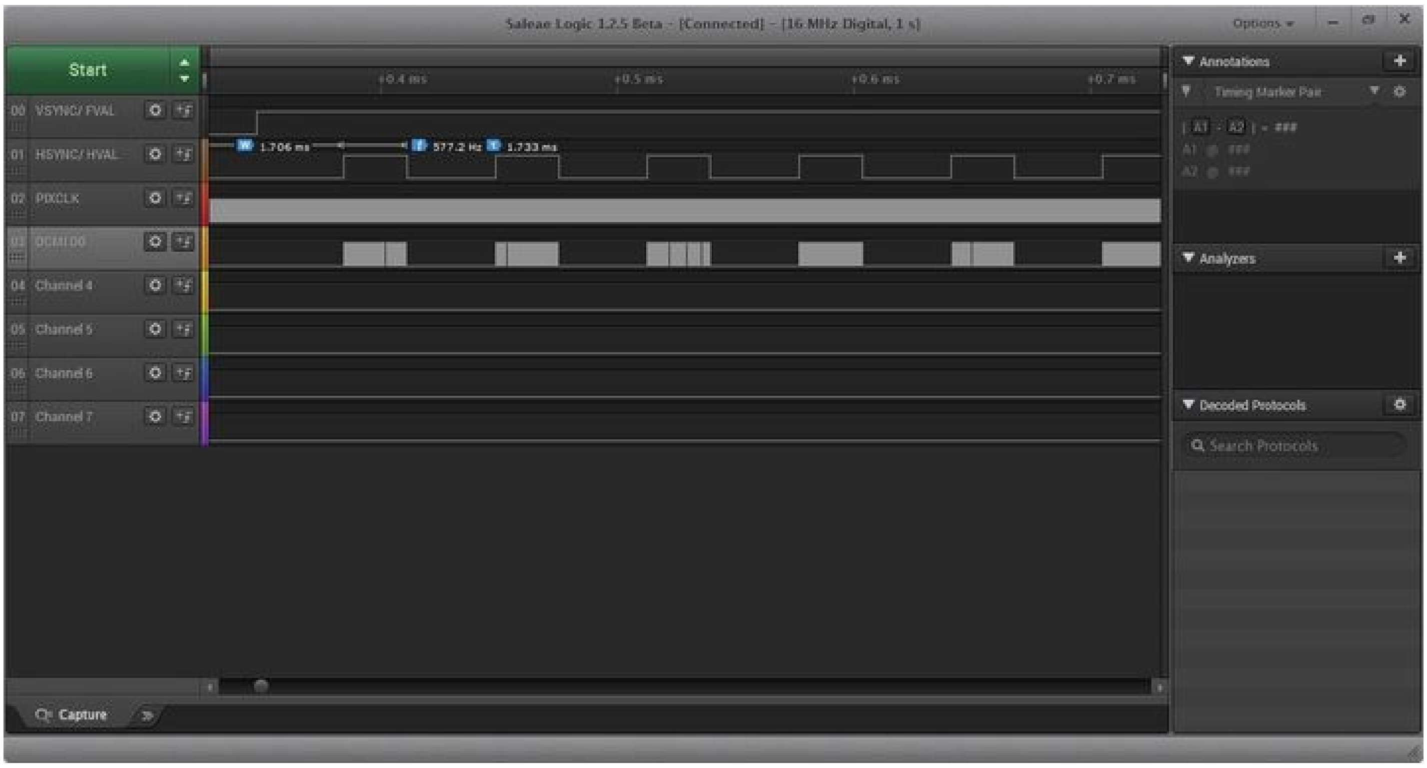
```
/**
  ******************************************************************************
  * @file    MT9P031.c
  * @author  Ferhat YOL
  * @version V1.0.0
  * @date    25-June-2015
  * @brief   MT9P031 Driver Kütüphanesidir.
  *
  @verbatim
  ******************************************************************************
  */

/* Includes ------------------------------------------------------------------*/
#include "stm32746g_discovery.h"
#include "MT9P031.h"
/* Private typedef -----------------------------------------------------------*/
/* Private define ------------------------------------------------------------*/
/* Bits definitions ----------------------------------------------------------*/
/* Private macro -------------------------------------------------------------*/
/* Private variables ---------------------------------------------------------*/
     DCMI_HandleTypeDef  hDcmiHandler;
     I2C_HandleTypeDef hI2cExtHandler = {0};

/* Private function prototypes -----------------------------------------------*/
/* Private functions ---------------------------------------------------------*/
/* Pin Map -------------------------------------------------------------------*/

/**
  * @brief  Configures the OV9655 DCMI
  * @param  None
  * @retval None
  */
uint8_t MT9P031_Configuration(void){
  DCMI_HandleTypeDef *phdcmi;
  uint8_t status = CAMERA_ERROR;
    uint16_t reg_val=0;

  /* Get the DCMI handle structure */
  phdcmi = &hDcmiHandler;

  /*** Configures the DCMI to interface with the camera module ***/
  /* DCMI configuration */
  phdcmi->Init.CaptureRate      = DCMI_CR_ALL_FRAME;
  phdcmi->Init.HSPolarity       = DCMI_HSPOLARITY_HIGH;
  phdcmi->Init.SynchroMode      = DCMI_SYNCHRO_HARDWARE;
  phdcmi->Init.VSPolarity       = DCMI_VSPOLARITY_HIGH;
  phdcmi->Init.ExtendedDataMode = DCMI_EXTEND_DATA_8B;
  phdcmi->Init.PCKPolarity      = DCMI_PCKPOLARITY_RISING;
  phdcmi->Instance              = DCMI;

  /* Power up camera */
  MT9P031_PwrUp();
    /* Configures the I2C peripheral */
    MT9P031_SCCB_Init(&hI2cExtHandler);
    /* Configures the DMA and GPIO */
    MT9P031_Msp_Init(&hDcmiHandler);
    /* Configures the DCMI peripheral */
    HAL_DCMI_Init(phdcmi);
  /* Configures the MT9P031 */
    if(MT9P031_SCCB_Read(MT9P031_I2C_ADDRESS, Chip_Verison, ®_val)==0xFF)
    {
        return CAMERA_NOT_DETECTED;
    }
    else
    {
       if(reg_val!=0x1801)
         {
            return CAMERA_NOT_SUPPORTED;
         }
    }
    /*  Software Reset */
    MT9P031_SCCB_Write(MT9P031_I2C_ADDRESS, Reset, 0x0001);
    MT9P031_SCCB_Write(MT9P031_I2C_ADDRESS, Reset, 0x0000);

    MT9P031_SCCB_Write(MT9P031_I2C_ADDRESS, Red_Gain, 0x7680);
    MT9P031_SCCB_Write(MT9P031_I2C_ADDRESS, Green1_Gain, 0x7680);
    MT9P031_SCCB_Write(MT9P031_I2C_ADDRESS, Green2_Gain, 0x7680);
    MT9P031_SCCB_Write(MT9P031_I2C_ADDRESS, Blue_Gain, 0x7680);
    MT9P031_SCCB_Write(MT9P031_I2C_ADDRESS, Shutter_Width_Lower, 0x01DF);
    MT9P031_SCCB_Write(MT9P031_I2C_ADDRESS, Shutter_Width_Upper, 0x0000);
    MT9P031_SCCB_Write(MT9P031_I2C_ADDRESS, Shutter_Delay, 0x0459);
    MT9P031_SCCB_Write(MT9P031_I2C_ADDRESS, Horizontal_Blank, 0x0000);
    MT9P031_SCCB_Write(MT9P031_I2C_ADDRESS, Vertical_Blank, 0x0019);
    MT9P031_SCCB_Write(MT9P031_I2C_ADDRESS, Column_Size, 0x027F);
    MT9P031_SCCB_Write(MT9P031_I2C_ADDRESS, Row_Size, 0x01DF);
    MT9P031_SCCB_Write(MT9P031_I2C_ADDRESS, Column_Start, 0x0010);
    MT9P031_SCCB_Write(MT9P031_I2C_ADDRESS, Row_Start, 0x0036);

    MT9P031_CAMERA_ContinuousStart();

  status=CAMERA_OK;
  return status;
}

/**
  * @brief  Configures the MCO Module
  * @param  None
  * @retval None
  */
void MT9P031_MCO_Configuration(void)
{
    HAL_RCC_MCOConfig(RCC_MCO1,RCC_MCO1SOURCE_PLLCLK,RCC_MCO_DIV4);
}

/**
  * @brief  Configures the DCMI to interface with the OV9655 camera module.
  * @param  None
  * @retval None
  */
void MT9P031_Msp_Init(DCMI_HandleTypeDef *hdcmi)
{
  static DMA_HandleTypeDef hdma_handler;
  GPIO_InitTypeDef gpio_init_structure;

  /*** Enable peripherals and GPIO clocks ***/
  /* Enable DCMI clock */
  __HAL_RCC_DCMI_CLK_ENABLE();

  /* Enable DMA2 clock */
  __HAL_RCC_DMA2_CLK_ENABLE();

  /* Enable GPIO clocks */
  __HAL_RCC_GPIOA_CLK_ENABLE();
  __HAL_RCC_GPIOD_CLK_ENABLE();
  __HAL_RCC_GPIOE_CLK_ENABLE();
  __HAL_RCC_GPIOG_CLK_ENABLE();
  __HAL_RCC_GPIOH_CLK_ENABLE();

  /*** Configure the GPIO ***/
  /* Configure DCMI GPIO as alternate function */
```

```
gpio_init_structure.Pin       = GPIO_PIN_4 | GPIO_PIN_6;
gpio_init_structure.Mode      = GPIO_MODE_AF_PP;
gpio_init_structure.Pull      = GPIO_PULLUP;
gpio_init_structure.Speed     = GPIO_SPEED_HIGH;
gpio_init_structure.Alternate = GPIO_AF13_DCMI;
HAL_GPIO_Init(GPIOA, &gpio_init_structure);

gpio_init_structure.Pin       = GPIO_PIN_3;
gpio_init_structure.Mode      = GPIO_MODE_AF_PP;
gpio_init_structure.Pull      = GPIO_PULLUP;
gpio_init_structure.Speed     = GPIO_SPEED_HIGH;
gpio_init_structure.Alternate = GPIO_AF13_DCMI;
HAL_GPIO_Init(GPIOD, &gpio_init_structure);

gpio_init_structure.Pin       = GPIO_PIN_5 | GPIO_PIN_6;
gpio_init_structure.Mode      = GPIO_MODE_AF_PP;
gpio_init_structure.Pull      = GPIO_PULLUP;
gpio_init_structure.Speed     = GPIO_SPEED_HIGH;
gpio_init_structure.Alternate = GPIO_AF13_DCMI;
HAL_GPIO_Init(GPIOE, &gpio_init_structure);

gpio_init_structure.Pin       = GPIO_PIN_9;
gpio_init_structure.Mode      = GPIO_MODE_AF_PP;
gpio_init_structure.Pull      = GPIO_PULLUP;
gpio_init_structure.Speed     = GPIO_SPEED_HIGH;
gpio_init_structure.Alternate = GPIO_AF13_DCMI;
HAL_GPIO_Init(GPIOG, &gpio_init_structure);

gpio_init_structure.Pin       = GPIO_PIN_9 | GPIO_PIN_10  | GPIO_PIN_11  |\
                                GPIO_PIN_12 | GPIO_PIN_14;
gpio_init_structure.Mode      = GPIO_MODE_AF_PP;
gpio_init_structure.Pull      = GPIO_PULLUP;
gpio_init_structure.Speed     = GPIO_SPEED_HIGH;
gpio_init_structure.Alternate = GPIO_AF13_DCMI;
HAL_GPIO_Init(GPIOH, &gpio_init_structure);

/*** Configure the DMA ***/
/* Set the parameters to be configured */
hdma_handler.Init.Channel             = DMA_CHANNEL_1;
hdma_handler.Init.Direction           = DMA_PERIPH_TO_MEMORY;
hdma_handler.Init.PeriphInc           = DMA_PINC_DISABLE;
hdma_handler.Init.MemInc              = DMA_MINC_ENABLE;
hdma_handler.Init.PeriphDataAlignment = DMA_PDATAALIGN_HALFWORD;
hdma_handler.Init.MemDataAlignment    = DMA_MDATAALIGN_HALFWORD;
hdma_handler.Init.Mode                = DMA_CIRCULAR;
hdma_handler.Init.Priority            = DMA_PRIORITY_HIGH;
hdma_handler.Init.FIFOMode            = DMA_FIFOMODE_DISABLE;
hdma_handler.Init.FIFOThreshold       = DMA_FIFO_THRESHOLD_FULL;
hdma_handler.Init.MemBurst            = DMA_MBURST_SINGLE;
hdma_handler.Init.PeriphBurst         = DMA_PBURST_SINGLE;

hdma_handler.Instance = DMA2_Stream1;

/* Associate the initialized DMA handle to the DCMI handle */
__HAL_LINKDMA(hdcmi, DMA_Handle, hdma_handler);

/*** Configure the NVIC for DCMI and DMA ***/
/* NVIC configuration for DCMI transfer complete interrupt */
HAL_NVIC_SetPriority(DCMI_IRQn, 5, 0);
HAL_NVIC_EnableIRQ(DCMI_IRQn);

/* NVIC configuration for DMA2D transfer complete interrupt */
HAL_NVIC_SetPriority(DMA2_Stream1_IRQn, 5, 0);
HAL_NVIC_EnableIRQ(DMA2_Stream1_IRQn);

/* Configure the DMA stream */
HAL_DMA_Init(hdcmi->DMA_Handle);
```

```
}

/**
  * @brief  Initializes I2C HAL.
  * @param  i2c_handler : I2C handler
  * @retval None
  */
void MT9P031_SCCB_Init(I2C_HandleTypeDef *i2c_handler)
{
      GPIO_InitTypeDef  gpio_init_structure;

    /* Enable GPIO and I2C1 clock */
    __HAL_RCC_GPIOB_CLK_ENABLE();

  if(HAL_I2C_GetState(i2c_handler) == HAL_I2C_STATE_RESET)
  {
     /* External, camera and Arduino connector  I2C configuration */
    i2c_handler->Instance = I2C1;
    i2c_handler->Init.Timing           = DISCOVERY_I2Cx_TIMING;
    i2c_handler->Init.OwnAddress1      = 0;
    i2c_handler->Init.AddressingMode   = I2C_ADDRESSINGMODE_7BIT;
    i2c_handler->Init.DualAddressMode  = I2C_DUALADDRESS_DISABLE;
    i2c_handler->Init.OwnAddress2      = 0;
    i2c_handler->Init.GeneralCallMode  = I2C_GENERALCALL_DISABLE;
    i2c_handler->Init.NoStretchMode    = I2C_NOSTRETCH_DISABLE;

    /* Configure I2C SCL Pin */
    gpio_init_structure.Pin = GPIO_PIN_8;
    gpio_init_structure.Mode = GPIO_MODE_AF_OD;
    gpio_init_structure.Pull = GPIO_NOPULL;
    gpio_init_structure.Speed = GPIO_SPEED_FAST;
    gpio_init_structure.Alternate = GPIO_AF4_I2C1;
    HAL_GPIO_Init(GPIOB, &gpio_init_structure);

    /* Configure I2C SDA Pin */
    gpio_init_structure.Pin = GPIO_PIN_9;
    HAL_GPIO_Init(GPIOB, &gpio_init_structure);

    /*** Configure the I2C peripheral ***/
    /* Enable I2C clock */
    __HAL_RCC_I2C1_CLK_ENABLE();
    /* Force the I2C peripheral clock reset */
    __HAL_RCC_I2C1_FORCE_RESET();
    /* Release the I2C peripheral clock reset */
    __HAL_RCC_I2C1_RELEASE_RESET();

    HAL_I2C_Init(i2c_handler);
  }
}

/**
  * @brief  CANERA power up
  * @retval None
  */
void MT9P031_PwrUp(void)
{
  GPIO_InitTypeDef gpio_init_structure;

  /* Enable GPIO clock */
  __HAL_RCC_GPIOH_CLK_ENABLE();

  /*** Configure the GPIO ***/
  /* Configure DCMI GPIO as alternate function */
  gpio_init_structure.Pin       = GPIO_PIN_13;
  gpio_init_structure.Mode      = GPIO_MODE_OUTPUT_PP;
  gpio_init_structure.Pull      = GPIO_NOPULL;
  gpio_init_structure.Speed     = GPIO_SPEED_HIGH;
  HAL_GPIO_Init(GPIOH, &gpio_init_structure);
  /* De-assert the camera POWER_DOWN pin (active high) */
  HAL_GPIO_WritePin(GPIOH, GPIO_PIN_13, GPIO_PIN_RESET);

  HAL_Delay(3);     /* POWER_DOWN de-asserted during 3ms */
}

/**
  * @brief  Starts the camera capture in continuous mode.
  * @param  buff: pointer to the camera output buffer
  * @retval None
  */
void MT9P031_CAMERA_ContinuousStart(void)
{
  /* Start the camera capture */
  HAL_DCMI_Start_DMA(&hDcmiHandler, DCMI_MODE_CONTINUOUS, (uint32_t) CAMERA_FRAME_BUFFER,
VGA_SIZE);
}

/**
  * @brief  MT9P031 writes single data.
  * @param  Addr: I2C address
  * @param  Reg: Register address
  * @param  Value: Data to be written
  * @retval None
  */
uint8_t MT9P031_SCCB_Write(uint8_t Addr, uint8_t Reg, uint16_t Value)
{
    uint8_t status = HAL_ERROR;
    uint8_t val[2];

    val[1] = Value & 0x00FF;
    val[0] = (Value & 0xFF00)>>8;

  status = HAL_I2C_Mem_Write(&hI2cExtHandler, Addr, (uint8_t)Reg, I2C_MEMADD_SIZE_8BIT,
(uint8_t*)&val, 2, 1000);

    return status;
}

/**
  * @brief  MT9P031 reads single data.
  * @param  Addr: I2C address
  * @param  Reg: Register address
  * @retval Read data
  */
uint8_t MT9P031_SCCB_Read(uint8_t Addr, uint8_t Reg, uint16_t *Value)
{
  uint8_t read_value[2];

    if(HAL_I2C_Mem_Read(&hI2cExtHandler, Addr, (uint8_t)Reg, I2C_MEMADD_SIZE_8BIT,
(uint8_t*)&read_value, 2, 1000)==HAL_ERROR)
  {
    return 0xFF;
    }

    *Value=((uint16_t)read_value[0]<<8) | read_value[1];
  return 0x00;
}

/**
  * @brief  BSP_DCMI_interrupt function.
  * @param  none
  * @retval none
  */
void BSP_DCMI_IRQ_Handler(void)
{
   HAL_DCMI_IRQHandler(&hDcmiHandler);
```

```
}

/* interrupt functions */

/**
  * @brief  Line event callback
  * @param  hdcmi: pointer to the DCMI handle
  * @retval None
  */
void HAL_DCMI_LineEventCallback(DCMI_HandleTypeDef *hdcmi)
{
    // user Code
         HAL_GPIO_TogglePin(GPIOC, GPIO_PIN_7);
}

/**
  * @brief  VSYNC event callback
  * @param  hdcmi: pointer to the DCMI handle
  * @retval None
  */
void HAL_DCMI_VsyncEventCallback(DCMI_HandleTypeDef *hdcmi)
{
  // user Code
}

/**
  * @brief  Frame event callback
  * @param  hdcmi: pointer to the DCMI handle
  * @retval None
  */
void HAL_DCMI_FrameEventCallback(DCMI_HandleTypeDef *hdcmi)
{
  // user Code
    HAL_GPIO_TogglePin(GPIOC, GPIO_PIN_6);
}

/**
  * @brief  Error callback
  * @param  hdcmi: pointer to the DCMI handle
  * @retval None
  */
void HAL_DCMI_ErrorCallback(DCMI_HandleTypeDef *hdcmi)
{
  // user Code

}



/**
  * @brief  Handles DMA interrupt request.
  * @retval None
  */
void BSP_CAMERA_DMA_IRQHandler(void)
{
  HAL_DMA_IRQHandler(hDcmiHandler.DMA_Handle);
}
```

MT9P031 Camera currently working, I can see it with logic analyzer.



Also works all interrupts. (Vsync, Frame and Line)

I'm getting data from D0-D7 pins. But DCMI_DR register always zero. I don't see any change while debugging.

What could be the problem. Please help me.