Hi everyone,
I'm trying to communicate with a light-to-digital sensor; first configure it and then receive its readings
I'm using stm32f4-discovery kit and I'm trying to communicate with TSL2581cs sensor
I followed this sequence
- checked BUSY flag
- sent a START condition
- checked EV5 event (is start bit sent?)
- sent slave address with the LSB = 0 (write)
- checked EV6 event (is slave address sent?)
@ here is a picture of the I2C registers in this moment

- sent command (the address of the data register in the sensor which I'll read from)
- checked EV8 event (is data byte sent?)
- Sent RESTART (repeated start condition) to change the direction of connection and change master mode from transmitter to receiver mode
- sent the slave address again with the LSB = 1 (read) ### this byte is not written in the data register, instead, zeros are written in the data register of the stm32f4 ??
@here is a picture of the I2C registers at this moment

- check EV6 event (is slave address sent?)
- start receiving data from the sensor ...

here is my complete code

<b>

```
#include <stm32f4xx.h>

// this slave address belongs to the STM32F4-Discovery board's
#define SLAVE_ADDRESS 0x39 // the slave address (example)
#define TIME_OUT 0X7FFF // time out constant
#define I2Cxx I2C1

/*******************************************************************************/
////////////////////////////////////////////////////////////////////////////////
/////////////////////////// Initialization GPIO, I2C ///////////////////////////
////////////////////////////////////////////////////////////////////////////////
/*******************************************************************************/
void I2C1_init(void){

  GPIO_InitTypeDef GPIO_InitStruct;
  GPIO_InitTypeDef GPIO_InitStruct2;
  I2C_InitTypeDef I2C_InitStruct;

  // enable APB1 peripheral clock for I2C1
  RCC_APB1PeriphClockCmd(RCC_APB1Periph_I2C1, ENABLE);
  // enable clock for SCL and SDA pins
  RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOB, ENABLE);

  RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOA, ENABLE);

  GPIO_InitStruct2.GPIO_Pin = GPIO_Pin_1 | GPIO_Pin_2 | GPIO_Pin_3 ;
  GPIO_InitStruct2.GPIO_Mode = GPIO_Mode_OUT;     // set pins to alternate function
  GPIO_InitStruct2.GPIO_Speed = GPIO_Speed_50MHz; // set GPIO speed
```

```
  GPIO_InitStruct2.GPIO_OType = GPIO_OType_PP; // set output to open drain --> the line has to be only pulled low, not driven high
  GPIO_InitStruct2.GPIO_PuPd = GPIO_PuPd_UP; // enable pull up resistors
  GPIO_Init(GPIOA, &GPIO_InitStruct2); // init GPIOB

  /* setup SCL and SDA pins
  * You can connect the I2C1 functions to two different
  * pins:
  * 1. SCL on PB6 or PB8
  * 2. SDA on PB7 or PB9
  */
  GPIO_InitStruct.GPIO_Pin = GPIO_Pin_8 | GPIO_Pin_9;   // we are going to use PB6 and PB9
  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_PinSource8, GPIO_AF_I2C1); // SCL
  GPIO_PinAFConfig(GPIOB, GPIO_PinSource9, GPIO_AF_I2C1); // SDA


  // configure I2C1
  I2C_InitStruct.I2C_ClockSpeed = 200000; // 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(I2C1, &I2C_InitStruct); // init I2C1

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



/*******************************************************************************/
////////////////////////////////////////////////////////////////////////////////
/////////////////////////// Configure the sensor ///////////////////////////////
////////////////////////////////////////////////////////////////////////////////
/*******************************************************************************/
/*Configure the sensor
* The Frame
* S | Slave Addr(0X39) | R/W'(0) | ACK. | COMMAND(11000000-control reg.) | ACK.
* | DATA_to_write(00xx0011-power on the sensor and ADC) | ACK.' | STOP
*
* parameters :
* - address : the address of the slave (sensor)
* - command : the address of the register to write in (must start with 110)
* - data : the data to be written in the destination register
*/
ErrorStatus I2C_Sensor_Config(uint8_t address, uint8_t command, uint8_t data)
{
  /*
```

```
* EV5 : start bit sent (BUSY, MSL, SB)
* EV6 : slave acknowledged address (BUSY, MSL, ADDR)
* EV8 : DR ready for new byte to transmit, transmit buffer empty (BUSY, MSL, TxE)
* EV9 : new byte received in the DR, receive buffer not empty (BUSY, MSL, RxNE)
*/

uint32_t time_out; //time out variable
ErrorStatus status = SUCCESS; //the status of progress

// 0# Wait until I2C1 is not busy any more ## Check flag BUSY = 0
time_out = TIME_OUT;
while(time_out != 0)
{
if(I2C_GetFlagStatus(I2Cxx, I2C_FLAG_BUSY) == 1)
{
time_out --;
if(time_out == 0)
{
status = ERROR;
}
}
else
time_out = 0;
}

if (status != ERROR)
{
// 1# Send I2C1 START condition           ## Signal START on the bus
I2C_GenerateSTART(I2Cxx, ENABLE);

// 2# Wait for I2C1 EV5(BUSY,MSL,SB)--> Slave has acknowledged start condition ## START correct
time_out = TIME_OUT;
while(time_out != 0)
{
if(I2C_CheckEvent(I2Cxx, I2C_EVENT_MASTER_MODE_SELECT) == 0)
{
time_out --;
if(time_out == 0)
{
status = ERROR;
}
}
else
time_out = 0;
}
}

if (status != ERROR)
{
// 3# Send slave Address for write
I2C_Send7bitAddress(I2Cxx, address, I2C_Direction_Transmitter);

/* 4# wait for I2Cx EV6 (BUSY, MSL, ADDR, TXE,TRA), check if
* Slave has acknowledged the address or not  ##Slave address acknowledged
*/
```

```
time_out = TIME_OUT;
while(time_out != 0)
{
if(I2C_CheckEvent(I2Cxx, I2C_EVENT_MASTER_TRANSMITTER_MODE_SELECTED) == 0)
{
time_out --;
if(time_out == 0)
{
status = ERROR;
}
}
else
time_out = 0;
}
}

if(status != ERROR)
{
// 5# send COMMAND = 11000000 (control register's address)
I2C_SendData(I2Cxx, command);

// 6# wait for EV8 (TRA, BUSY, MSL, TXE ) ##DR ready for a new byte to transmit
time_out = TIME_OUT;
while(time_out != 0)
{
if(I2C_CheckEvent(I2Cxx, I2C_EVENT_MASTER_BYTE_TRANSMITTING) == 0)
{
time_out --;
if(time_out == 0)
{
status = ERROR;
}
}
else
time_out = 0;
}
}

if(status != ERROR)
{
// 7# send DATA = 00XX0011 (Power on the ADC and the sensor)
I2C_SendData(I2Cxx, data);

// 8# wait for EV8 (TRA, BUSY, MSL, TXE ) ##DR ready for a new byte to transmit
time_out = TIME_OUT;
while(time_out != 0)
{
if(I2C_CheckEvent(I2Cxx, I2C_EVENT_MASTER_BYTE_TRANSMITTING) == 0)
{
time_out --;
if(time_out == 0)
{
status = ERROR;
}
}
else
```

```
  time_out = 0;
  }
  }

  // 9# Generate STOP to terminate the frame
  I2C_GenerateSTOP(I2Cxx, ENABLE);

  GPIO_ResetBits ( GPIOA, GPIO_Pin_1);


  return status;
}

/*******************************************************************************/
////////////////////////////////////////////////////////////////////////////////
/////////////////////////// Read Data From the Sensor///////////////////////////
////////////////////////////////////////////////////////////////////////////////
/*******************************************************************************/
/* Read Data From the Sensor
* The Frame - read the first register (DATA0LOW)
* S | Slave Addr(0X39) | R/W'(0) | ACK. | COMMAND(11000104-DATA0LOW) | ACK.
* | Sr | Slave Addr(0x39) | R/W'(1) | ACK. | DATA_read | ACK.' | STOP
* The Frame - read the first register (DATA0HIGH)
* S | Slave Addr(0X39) | R/W'(0) | ACK. | COMMAND(11000105-DATA0HIGH) | ACK.
* | Sr | Slave Addr(0x39) | R/W'(1) | ACK. | DATA_read | ACK.' | STOP
* parameters :
* - address : the address of the slave (sensor)
* - command : the address of the register to write in (must start with 110)
* - data : the data to be written in the destination register
* Returns :
*      - the data read from the sensor
*/
uint8_t ReadData(uint8_t address, uint8_t command)
{
  // a variable to receive data
  static volatile uint8_t received_data = 0x01;
  static volatile uint32_t time_out;
    static volatile ErrorStatus status = SUCCESS;
  static volatile ErrorStatus end = SUCCESS;

    time_out = TIME_OUT;
    while(time_out != 0)
    {
  // 0# check busy flag
        if (I2C_GetFlagStatus(I2Cxx, I2C_FLAG_BUSY) == 1)
        {
            time_out--;
            if (time_out == 0)
            {
                status = ERROR;
            }
        }
        else
        {
            time_out = 0;
        }
```

```
  }

            ////////////////////////////

  if (status != ERROR)
   {
  // 1# Generate start condition
      I2C_GenerateSTART(I2Cxx, ENABLE);

      time_out = TIME_OUT;
      while(time_out != (uint32_t)0)
  {
  // 2# Check EV5 ## start bit sent
          if (I2C_CheckEvent(I2Cxx, I2C_EVENT_MASTER_MODE_SELECT) == 0)
          {
             time_out--;
             if (time_out == 0)
             {
                status = ERROR;
             }
          }
          else
          {
             time_out = 0;
          }
       }
   }

  ////////////////////////////////

    if (status != ERROR)
    {
  // 3# Send slave address
  // Note, I2C_Direction_Transmitter makes R/Wr' = 0 (Write)
  I2C_Send7bitAddress(I2Cxx, address, I2C_Direction_Transmitter);

      time_out = TIME_OUT;
      while(time_out != (uint32_t)0)
      {
  // 4# Check EV6 ## is address sent successfuly ?
          if (I2C_CheckEvent(I2Cxx, I2C_EVENT_MASTER_TRANSMITTER_MODE_SELECTED) == 0)
          {
             time_out--;
             if (time_out == 0)
             {
                status = ERROR;
             }
          }
          else
          {
             time_out = 0;
          }
       }
    }

  ////////////////////////////////////
```

```
 if (status != ERROR)
{
// 5# Send data (data register's address to read from)
I2C_SendData(I2Cxx, command);      // SubAddress

time_out = TIME_OUT;
while(time_out != 0)
{
// 6# Check BTF (Byte Transfer Finished)
if(I2C_GetFlagStatus(I2Cxx, I2C_FLAG_BTF) == RESET)
{
time_out--;
if (time_out == 0)
{
status = ERROR;
}
}
else
{
time_out = 0;
}
}
}

////////////////////////////////////

  if (status != ERROR)
  {
// 7# Generate Repeated Start
     I2C_GenerateSTART(I2Cxx, ENABLE);

     time_out = TIME_OUT;
     while(time_out != (uint32_t)0)
     {
// 8# Check EV5 (is start bit sent?)
        if (I2C_CheckEvent(I2Cxx, I2C_EVENT_MASTER_MODE_SELECT) == 0)
        {
           time_out--;
           if (time_out == 0)
           {
              status = ERROR;
           }
        }
        else
        {
           time_out = 0;
        }
     }
  }

////////////////////////////////////
if (status != ERROR)
  {
```

```
// 9# Send slave address
// note I2C_Direction_Receiver make R/Wr' = 1 (Read)
//I2C1->DR = 0x73;
I2C_Send7bitAddress(I2Cxx, 0x73, I2C_Direction_Receiver); //here's the problem

time_out = TIME_OUT;
while(time_out != 0)
{
// 10# Check ADDR flag
if(I2C_GetFlagStatus(I2Cxx, I2C_FLAG_ADDR) == RESET)
{
time_out--;
if (time_out == 0)
{
status = ERROR;
}
}
else
{
time_out = 0;
}
}
  }
////////////////////////////////////

if (status != ERROR)
{
// 11# Disable Acknowledge from slave
I2C_AcknowledgeConfig(I2Cxx, DISABLE);

// 12# Clear ADDR register by reading SR1 then SR2 register (SR1 has already been read)
(void)I2Cxx->SR2;


// 13# Send STOP Condition
I2C_GenerateSTOP(I2Cxx, ENABLE);

time_out = TIME_OUT;
while(time_out != (uint32_t)0)
{
// 14# Check RxNE flag (is there a received byte?)
if (I2C_GetFlagStatus(I2Cxx, I2C_FLAG_RXNE) == 0)
{
time_out--;
if (time_out == 0)
{
status = ERROR;
}
}
else
{
time_out = 0;
}
}

received_data = I2C_ReceiveData(I2Cxx);

time_out = TIME_OUT;
```

```
  while(time_out != (uint32_t)0)
  {
  // 15# Check STOP flag
  if ((I2Cxx->CR1 & I2C_CR1_STOP) != 0)
  {
  time_out--;
  if (time_out == 0)
  {
  status = ERROR;
  }
  }
  else
  {
  time_out = 0;
  }
  }
  // 16# Re-Enable Acknowledgement to be ready for another reception
  I2C_AcknowledgeConfig(I2Cxx, ENABLE);
  }

  end = status;
  return received_data;
}

/* This function delays with a delay d*/
void delay(uint32_t d)
{
  while(d--)
  ;
}

/* This function blinks the RGB LED*/
void blink(void)
{
  GPIO_SetBits ( GPIOA, GPIO_Pin_1 | GPIO_Pin_2 | GPIO_Pin_3); //set all
  delay(20*1000000);
  GPIO_ResetBits ( GPIOA, GPIO_Pin_1 | GPIO_Pin_2 | GPIO_Pin_3); //reset all
  delay(20*1000000);
  //blue
  GPIO_SetBits ( GPIOA, GPIO_Pin_1);
  delay(20*1000000);
  GPIO_ResetBits ( GPIOA, GPIO_Pin_1);
  //green
  GPIO_SetBits ( GPIOA, GPIO_Pin_2);
  delay(20*1000000);
  GPIO_ResetBits ( GPIOA, GPIO_Pin_2);
  //red
  GPIO_SetBits ( GPIOA, GPIO_Pin_3);
  delay(20*1000000);
  GPIO_ResetBits ( GPIOA, GPIO_Pin_3);
  delay(20*1000000);
  GPIO_ResetBits ( GPIOA, GPIO_Pin_1 | GPIO_Pin_2 | GPIO_Pin_3); //reset all
  delay(20*1000000);
}
```

```
/* Main Function */
int main(void){

  ErrorStatus error;
  static volatile uint8_t received_data;
  error += 0;
  // a variable to receive data
  received_data += 0;

  // initialize I2C peripheral
  I2C1_init();


  while(1)
  {
  //blink the RGB LED
  blink();

  //send data to the sensor to configure it
  error = I2C_Sensor_Config(SLAVE_ADDRESS<<1, 0xC0, 0x03);

  //read data from the sensor
  received_data = ReadData(SLAVE_ADDRESS<<1, 0xD4);
  }
}
</b>
```

Now the question is .. why I can't write the slave address (with the LSB = 1 (read)) in the data register to send it although I've written it in the data register when the LSB = 0 (write)??