Hello,

I have seen that similar issue was noticed by quite a few other people. I have read everything I could find on the web, but I still do not know what is the issue (did not find anything that could help me resolve it).

I have been trying to get the I2C communication working (with an IR array sensor 16x4: http://www.heimannsensor.com/products_imaging_2d.php) but only to partial success. There are two devices on the I2C line - EEPROM and SENSOR (RAM, where the measurements are populated) with different addresses. I am able to read the data from the EEPROM,but cannot read the data from the SENSOR.

Today I finally got my hands on the oscilloscope and measured the signal on both lines. Here is the relevant code that I am using (in "main" there are two comments in the code, "WORKS OK!!!" and "DOES NOT WORKI!!!"):

```
#define EEPROM_address_write    0xA0 // address
0x50                                                                    // I2C
EEPROM address
#define EEPROM_address_read     0xA1
#define sensor_address_write    0xC0 // address
0x60                                                                    // I2C
SENSOR address
#define sensor_address_read     0xC1

/**
  * @brief   Main program
  * @param  None
  * @retval None
  */
int main(void)
{
/*!< At this stage the microcontroller clock setting is already configured,
     this is done through SystemInit() function which is called from startup
     file (startup_stm32l1xx_md.s) before to branch to application main.
     To reconfigure the default setting of SystemInit() function, refer to
     system_stm32l1xx.c file
*/
        // Enable HSI Clock
  RCC_HSICmd(ENABLE);

    // Wait till HSI is ready
  while (RCC_GetFlagStatus(RCC_FLAG_HSIRDY) == RESET) {}

    RCC_SYSCLKConfig(RCC_SYSCLKSource_HSI);

    // SysTick 1 msec interrupts (1000 = 1ms, 100 = 10 ms ...)
  if(SysTick_Config(SystemCoreClock / 1000))
    {
        // capture error
        while(1);
    }

    // Initialize I2C
    Init_I2C2();

    // WORKS OK!!!
//  I2C_start(I2C2,EEPROM_address_write,I2C_Direction_Transmitter);
//  I2C_write(I2C2,0x00);
//  I2C_stop(I2C2);
//  I2C_start(I2C2,EEPROM_address_read,I2C_Direction_Receiver);
//  test = I2C_read_ack(I2C2);
//  I2C_stop(I2C2);

    // DOES NOT WORK!!!
    I2C_start(I2C2,sensor_address_write,I2C_Direction_Transmitter);
```

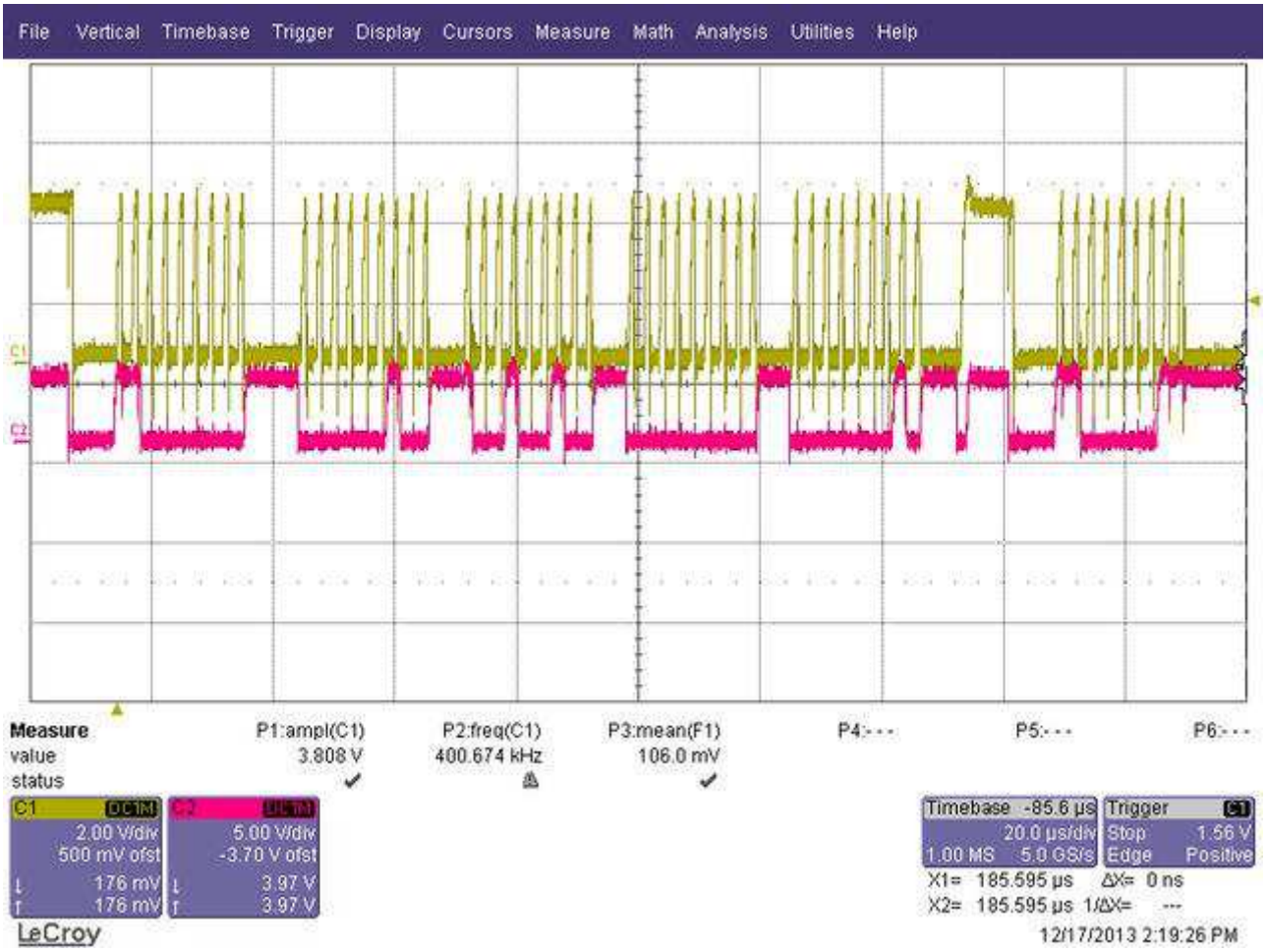
```
    I2C_write(I2C2,0x02);
    I2C_write(I2C2,0x92);
    I2C_write(I2C2,0x00);
    I2C_write(I2C2,0x01);
    I2C_stop(I2C2);
    I2C_start(I2C2,sensor_address_read,I2C_Direction_Receiver);
    // uint8_t pointer to REG_
    pD_g = (uint8_t*) REG_;
    (*pD_g) = I2C_read_ack(I2C2);
    pD_g++;
    (*pD_g) = I2C_read_ack(I2C2);
    I2C_stop(I2C2);
}


/**
  * @brief  To initialize the I2C ports
  * @caller main
  * @param None
  * @retval None
  */
void Init_I2C2(void){

    GPIO_InitTypeDef GPIO_InitStruct;
    I2C_InitTypeDef I2C_InitStruct;

    // Enable APB1 peripheral clock for I2C2
    RCC_APB1PeriphClockCmd(RCC_APB1Periph_I2C2, ENABLE);
    // Enable clock for SCL and SDA pins
    RCC_AHBPeriphClockCmd(RCC_AHBPeriph_GPIOB, ENABLE);

    // Setup SCL and SDA pins
    GPIO_InitStruct.GPIO_Pin = GPIO_Pin_10 | GPIO_Pin_11;                                   //
using PB10 and PB11
    GPIO_InitStruct.GPIO_Mode =
GPIO_Mode_AF;                                                       // set pins to
alternate function
    GPIO_InitStruct.GPIO_Speed =
GPIO_Speed_40MHz;                                               // 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 (disabled pullup = GPIO_PuPd_NOPULL, enabled pullup = GPIO_PuPd_UP)
    GPIO_Init(GPIOB,
&GPIO_InitStruct);                                                                      //
init GPIOB

    // Connect I2C2 pins to AF

    GPIO_PinAFConfig(GPIOB,GPIO_PinSource10,GPIO_AF_I2C2);                              // SCL
    GPIO_PinAFConfig(GPIOB,GPIO_PinSource11,GPIO_AF_I2C2);                              // SDA

    // Configure I2C2
    I2C_InitStruct.I2C_ClockSpeed =
400000;                                                             // 400kHz
    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_Enable;                                                          // disable
acknowledge when reading (can be changed later on)
    I2C_InitStruct.I2C_AcknowledgedAddress = I2C_AcknowledgedAddress_7bit;  // set address length
```

```
    I2C_InitStruct.I2C_AcknowledgedAddress = I2C_AcknowledgedAddress_7bit; // set address length
to 7 bit addresses
    I2C_Init(I2C2,
&I2C_InitStruct);                                                          // init I2C2

    // Enable I2C2
    I2C_Cmd(I2C2, ENABLE);
}

/**
  * @brief  To issue a start condition and transmit the slave address + R/W bit
  * @caller main
  * @param  I2C_TypeDef* I2Cx,
                        uint8_t address,
                        uint8_t direction
  * @retval None
  */
/* This function issues a start condition and
 * transmits the slave address + R/W bit
 *
 * Parameters:
 *      I2Cx --> the I2C peripheral e.g. I2C2
 *      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 I2C2 is not busy anymore
    while(I2C_GetFlagStatus(I2Cx, I2C_FLAG_BUSY));

    // Send I2C2 START condition
    I2C_GenerateSTART(I2Cx, ENABLE);

    // Wait for I2C2 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 I2C2 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));

    }
}

/**
  * @brief  To transmit one byte to slave device
  * @caller main
  * @param  I2C_TypeDef* I2Cx,
                        uint8_t data
  * @retval None
  */
/* This function transmits one byte to the slave device
 * Parameters:
 *      I2Cx --> the I2C peripheral e.g. I2C2
 *      data --> the data byte to be transmitted
 */
void I2C_write(I2C_TypeDef* I2Cx, uint8_t data)
{
```

```
{
    I2C_SendData(I2Cx, data);
    // wait for I2C2 EV8_2 --> byte has been transmitted
    while(!I2C_CheckEvent(I2Cx, I2C_EVENT_MASTER_BYTE_TRANSMITTED));
}

/**
  * @brief  To read one byte from slave device and Ack
  * @caller main
  * @param  I2C_TypeDef* I2Cx
  * @retval data
  */
/* This function reads one byte from the slave device
 * and acknowledges the byte (requests another byte)
 */
uint8_t I2C_read_ack(I2C_TypeDef* I2Cx){
    // 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
    return I2C_ReceiveData(I2Cx);
}

/**
  * @brief  To read one byte from slave device and Nack
  * @caller main
  * @param  I2C_TypeDef* I2Cx
  * @retval 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){
    // disabe acknowledge of received data
    // nack also generates stop condition after last byte received
    // see reference manual for more info
    I2C_AcknowledgeConfig(I2Cx, DISABLE);
    I2C_GenerateSTOP(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
    return I2C_ReceiveData(I2Cx);
}

/**
  * @brief  To generate the stop condition and release the I2C bus
  * @caller main
  * @param  I2C_TypeDef* I2Cx
  * @retval None
  */
/* This funtion issues a stop condition and therefore

 * releases the bus
 */
void I2C_stop(I2C_TypeDef* I2Cx){
    // Send I2C2 STOP Condition
    I2C_GenerateSTOP(I2Cx, ENABLE);
}

/**
  * @brief  Inserts a delay time.
  * @param  nTime: specifies the delay time length, in 1 ms.
  * @retval None
  */
void Delay(__IO uint32_t nTime)
{
    // delay 1000 ms introduces 2000 ms delay (don't know why)
    // to compensate for this, divide argument by 2
  TimingDelay = (uint32_t) nTime/2;
```

```
  TimingDelay = (uint32_t) nTime/2;
  while(TimingDelay != 0);
}

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

#ifdef  USE_FULL_ASSERT

/**
  * @brief  Reports the name of the source file and the source line number
  *         where the assert_param error has occurred.
  * @param  file: pointer to the source file name
  * @param  line: assert_param error line source number
  * @retval None
  */
void assert_failed(uint8_t* file, uint32_t line)
{
  /* User can add his own implementation to report the file name and line number,
     ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) */

  // Infinite loop
  while (1)
  {
  }
}

#endif
```

So if I uncomment the first part ("WORKS OK!!!") and comment the second part ("DOES NOT WORK!!!), I am able to receive the data. If I do it the other way around I get stuck. Please see the image below. It is seen that when the sensor should transmitt the data back (please see the rightmost part), the SDA line is held high. The SDA line should go low on the 8-th cycle, right? Does that mean the master did not receive the acknowledge bit? No clock is then generated and the program hangs waiting (EV6 check I think).



Does anybody else have (had) the same problem. And more importantly, did anybody solve this, since there were quite a few questions regarding this, but no answer that could help me. What could be the cause of this? I do not understand how one I2C address works, but the other does not. The connection is clearly not the problem, since the same line is used.

I have also tried two different IR array sensors with two different pullup resistors (4k7 ohm) on both lines, connected to 3.3V and also 5V. I also tried to lower the clock speed.
Another thing that puzzles me is that for a repeated start I need to first generate the stop condition and start the I2C again... Putting only start does not work in both cases.

Is there any other driver that I can use to communicate with the I2C bus?

I hope my question is somewhat clear... Please offer some advice.

Thank you and best regards,
K