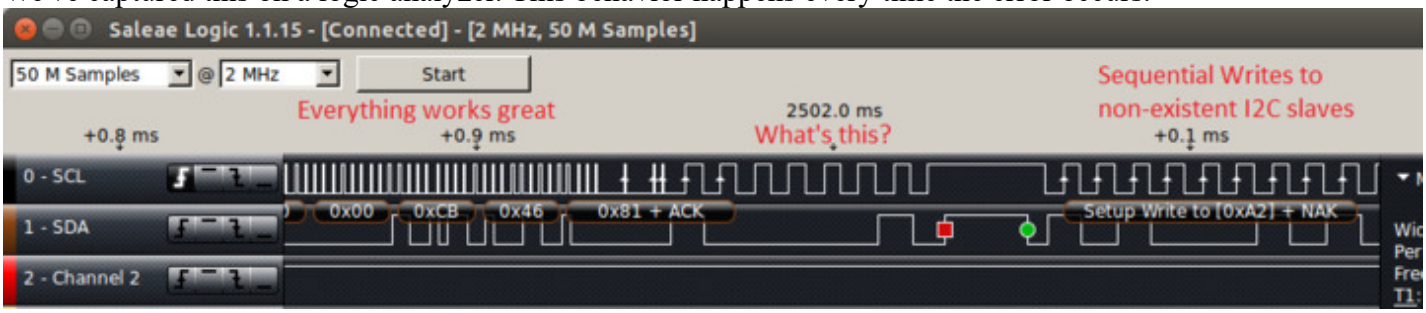
Hi,

I'm using I2C STM32F207ZET6. It usually works well, and we can usually read and write with no problems. However, there is a consistent error that pops up where, if we are reading a number of bytes sequentially, the reads will fail. After that, it looks like the STM32 goes through several I2C addresses polling for an available slave (I'm not sure why this is happening now).

We've captured this on a logic analyzer. This behavior happens every time the error occurs.



This picture shows the I2C bus correctly reading sequential data from an EEPROM. At some point, the STM32 acks a received byte and then the clock duty cycle changes for a bit. After that, it looks like the STM32 tries to cycle through various slave addresses (starting from the address of my EEPROM+2).

Here's how I'm setting up the I2C bus:

```
//Setup the i2c bus
 GPIO_InitTypeDef GPIO_InitStructure;
 I2C_InitTypeDef  I2C_InitStruct;

// I2C2 clock enable
 RCC_APB1PeriphClockCmd(RCC_APB1Periph_I2C2, ENABLE);
 // GPIOF clock enable
 RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOF, ENABLE);

 // Connect I2C2 pins to AF4 ************************************************
 GPIO_PinAFConfig(GPIOF, GPIO_PinSource0, GPIO_AF_I2C2);
 GPIO_PinAFConfig(GPIOF, GPIO_PinSource1, GPIO_AF_I2C2);

 // Configure I2C2 GPIOs ************************************************
 GPIO_InitStructure.GPIO_Pin = GPIO_Pin_0 | GPIO_Pin_1;
 GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF;
 GPIO_InitStructure.GPIO_Speed = GPIO_Speed_2MHz;
 GPIO_InitStructure.GPIO_OType = GPIO_OType_OD;
 GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL;
 GPIO_Init(GPIOF, &GPIO_InitStructure);

 // Configure I2C2 ************************************************************
 // I2C DeInit
 I2C_DeInit(I2C2);

 // Enable the I2C peripheral
 I2C_Cmd(I2C2, ENABLE);

 // Set the I2C structure parameters
 I2C_InitStruct.I2C_Mode = I2C_Mode_I2C;
 I2C_InitStruct.I2C_DutyCycle = I2C_DutyCycle_2;
 I2C_InitStruct.I2C_OwnAddress1 = 0xFE;
 I2C_InitStruct.I2C_Ack = I2C_Ack_Enable;
 I2C_InitStruct.I2C_AcknowledgedAddress = I2C_AcknowledgedAddress_7bit;
 I2C_InitStruct.I2C_ClockSpeed = 200000; // full speed clock rate causes I2C bus to periodically
fail, so using 200k instead

 // Initialize the I2C peripheral w/ selected parameters
 I2C_Init(I2C2, &I2C_InitStruct);
```

Here's the code I'm using to read from the EEPROM.

```
bool rx_Eeprom_Data(int addr, unsigned char * data, int data_len)
{
  uint32_t timeout = TIMEOUT_MAX;
  uint8_t tx_data[2];

  tx_data[1] = addr & 0xFF;
  tx_data[0] = addr >> 8;

  // Generate the Start Condition
  I2C_GenerateSTART(I2C2, ENABLE);

  // Test on I2C2 EV5 and clear it
  timeout = TIMEOUT_MAX; // Initialize timeout value
  while(!I2C_CheckEvent(I2C2, I2C_EVENT_MASTER_MODE_SELECT))
  {
    // If the timeout delay is exeeded, exit with error code
    if ((timeout--) == 0) return false;
  }

   // Send DCMI selcted device slave Address for write
  I2C_Send7bitAddress(I2C2, EEPROM_ADDX, I2C_Direction_Transmitter);
   // Test on I2C2 EV6 and clear it
  timeout = TIMEOUT_MAX; // Initialize timeout value

  while(!I2C_CheckEvent(I2C2, I2C_EVENT_MASTER_TRANSMITTER_MODE_SELECTED))
  {
    // If the timeout delay is exeeded, exit with error code
    if ((timeout--) == 0) return false;
  }

  for (int j = 0; j < 2; j++)
  {
    // Send Data
    I2C_SendData(I2C2, tx_data[j]);

    // Test on I2C2 EV8 and clear it
    timeout = TIMEOUT_MAX; // Initialize timeout value
    while(!I2C_CheckEvent(I2C2, I2C_EVENT_MASTER_BYTE_TRANSMITTED))
    {
        // If the timeout delay is exeeded, exit with error code
        if ((timeout--) == 0) return false;
    }
  }

    // Generate the Start Condition
  I2C_GenerateSTART(I2C2, ENABLE);

  // Test on I2C2 EV5 and clear it
  timeout = TIMEOUT_MAX; // Initialize timeout value
  while(!I2C_CheckEvent(I2C2, I2C_EVENT_MASTER_MODE_SELECT))
  {
    // If the timeout delay is exeeded, exit with error code
    if ((timeout--) == 0) return false;
  }

   // Send DCMI selcted device slave Address for read
  I2C_Send7bitAddress(I2C2, EEPROM_ADDX, I2C_Direction_Receiver);
   // Test on I2C2 EV6 and clear it
  timeout = TIMEOUT_MAX; // Initialize timeout value

  while(!I2C_CheckEvent(I2C2, I2C_EVENT_MASTER_TRANSMITTER_MODE_SELECTED))
```

```
  {
    // If the timeout delay is exeeded, exit with error code
    if ((timeout--) == 0) return false;
  }


  for (int j = 0; j < data_len - 1; j++)
  {

    // Prepare an ACK for the next data received
    I2C_AcknowledgeConfig(I2C2, ENABLE);

    // Test on I2C2 EV8 and clear it
    timeout = TIMEOUT_MAX; // Initialize timeout value
    while(!I2C_CheckEvent(I2C2, I2C_EVENT_MASTER_BYTE_TRANSMITTED))
    {
        // If the timeout delay is exeeded, exit with error code
        if ((timeout--) == 0) return false;
    }

    data[j] = I2C_ReceiveData(I2C2);
  }

    // Prepare an NACK for the next data received
    I2C_AcknowledgeConfig(I2C2, DISABLE);

    // Test on I2C2 EV8 and clear it
    timeout = TIMEOUT_MAX; // Initialize timeout value
    while(!I2C_CheckEvent(I2C2, I2C_EVENT_MASTER_BYTE_TRANSMITTED))
    {
        // If the timeout delay is exeeded, exit with error code
        if ((timeout--) == 0) return false;
    }

   // Send I2C2 STOP Condition
  I2C_GenerateSTOP(I2C2, ENABLE);

  data[data_len - 1] = I2C_ReceiveData(I2C2);

  return true;
}
```