I2C busy flag strange behaviour

I've been using STMCUBE combined with Keil for some time now. For most part I like the HAL library and the documentation for STM32f1xx drivers is quite good.

I'm doing a project where I am using STM32f103rb Nucleo card combined with an MPU6050 gyro/accelerometer. I use code generation tool STM32CubeMX in order to generate initiation function. However when I want to implement I2C I have a strange problem. STM32Cube generates all necessary initiation steps, handle is set up then GPIO pins set as OD, then finally the clock is enabled using the macro `__HAL_RCC_I2C1_CLK_ENABLE()` , however when this macro is run within the `HAL_I2C_MspInit` , the I2C busy flag seems to be set, and is not cleared, hence I can't communicate with the MPU6050 device.

I noticed that if I put something (for example a measuring probe) on the SDA line while macro `__HAL_RCC_I2C1_CLK_ENABLE()` is run, the busy flag is not set and my I2C communication works until I reset the micro controller.

Another(better than putting a physical probe?) way that seems to work is that after macro `__HAL_RCC_I2C1_CLK_ENABLE()` is run, I use macros `__HAL_RCC_I2C1_FORCE_RESET()` and `__HAL_RCC_I2C1_RELEASE_RESET()` . This way my I2C communication works fine.

I think it's strange and I can't really explain the behaviour. But since I added the force reset and release reset macros, I haven't had any I2C problem, it works perfectly.

Let me know if I need to share some more code.

microcontroller i2c stm32f10x stm32cubemx hal-library

edited Nov 8 '16 at 8:02 asked Nov 7 '16 at 7:31
Nelle
11 1 3

1 The I2C pull up resistors are alright? – Bence Kaulics Nov 7 '16 at 8:50

Yes, Im using 2.7k pullups, works fine – Nelle Nov 7 '16 at 10:20

What is the SDA line's state initially and after you touch it with the probe. What change occurs? – Bence Kaulics Nov 7 '16 at 10:24

It is high initially, theres a short "spike" that makes it low when I have the probe connected during CLK_ENABLE, after that its high again – Nelle Nov 7 '16 at 12:17

I'm thinking whether, the probe manages to simulate a stop condition, which clears the i2c busy flag. But then I wonder why is the bus busy in the first place, since nothing but the init function has been executed. And if I'm right, how come the FORCE_RESET and RELEASE_RESET macros manages to do it, and is it a stable solution? – Nelle Nov 8 '16 at 7:49

3 Answers

ST has released an errata sheet called:

STM32F100xC, STM32F10 0xD and STM32F100xE high-density value line device limitations.

The interesting point here is:

> 2.9.7 I2C analog filter may provide wrong value, locking BUSY flag and preventing master mode entry

There is a detailed 15 step workaround that worked for me, surprisingly for an STM32F446, so I2C peripherals of every STM32 CORTEX-M series might be affected.

During this operation, the lines must not be actively pulled up or down by a bus member. So if you connect two I2C interfaces of the same MCU to the bus, first set up the pins of both to Alternate Function/Open Drain, then call the routine, as a transition of logical levels is required.

Here is an example with HAL libraries I use after the first initialization and during runtime, if an error occurs. As said above, this is for STM32F4, libraries for SMT32F1 might differ a bit.

```
struct I2C_Module
{
  I2C_HandleTypeDef   instance;
  uint16_t            sdaPin;
  GPIO_TypeDef*       sdaPort;
  uint16_t            sclPin;
  GPIO_TypeDef*       sclPort;
```

```
};

void I2C_ClearBusyFlagErratum(struct I2C_Module* i2c)
{
  GPIO_InitTypeDef GPIO_InitStructure;

  // 1. Clear PE bit.
  i2c->instance.Instance->CR1 &= ~(0x0001);

  //  2. Configure the SCL and SDA I/Os as General Purpose Output Open-Drain, High level
(Write 1 to GPIOx_ODR).
  GPIO_InitStructure.Mode         = GPIO_MODE_OUTPUT_OD;
  GPIO_InitStructure.Alternate    = I2C_PIN_MAP;
  GPIO_InitStructure.Pull         = GPIO_PULLUP;
  GPIO_InitStructure.Speed        = GPIO_SPEED_FREQ_HIGH;

  GPIO_InitStructure.Pin          = i2c->sclPin;
  HAL_GPIO_Init(i2c->sclPort, &GPIO_InitStructure);
  HAL_GPIO_WritePin(i2c->sclPort, i2c->sclPin, GPIO_PIN_SET);

  GPIO_InitStructure.Pin          = i2c->sdaPin;
  HAL_GPIO_Init(i2c->sdaPort, &GPIO_InitStructure);
  HAL_GPIO_WritePin(i2c->sdaPort, i2c->sdaPin, GPIO_PIN_SET);

  // 3. Check SCL and SDA High level in GPIOx_IDR.
  while (GPIO_PIN_SET != HAL_GPIO_ReadPin(i2c->sclPort, i2c->sclPin))
  {
    asm("nop");
  }

  while (GPIO_PIN_SET != HAL_GPIO_ReadPin(i2c->sdaPort, i2c->sdaPin))
  {
    asm("nop");
  }

  // 4. Configure the SDA I/O as General Purpose Output Open-Drain, Low level (Write 0 to
GPIOx_ODR).
  HAL_GPIO_WritePin(i2c->sdaPort, i2c->sdaPin, GPIO_PIN_RESET);

  //  5. Check SDA Low level in GPIOx_IDR.
  while (GPIO_PIN_RESET != HAL_GPIO_ReadPin(i2c->sdaPort, i2c->sdaPin))
  {
    asm("nop");
  }

  // 6. Configure the SCL I/O as General Purpose Output Open-Drain, Low level (Write 0 to
GPIOx_ODR).
  HAL_GPIO_WritePin(i2c->sclPort, i2c->sclPin, GPIO_PIN_RESET);

  //  7. Check SCL Low level in GPIOx_IDR.
  while (GPIO_PIN_RESET != HAL_GPIO_ReadPin(i2c->sclPort, i2c->sclPin))
  {
    asm("nop");
  }

  // 8. Configure the SCL I/O as General Purpose Output Open-Drain, High level (Write 1
to GPIOx_ODR).
  HAL_GPIO_WritePin(i2c->sclPort, i2c->sclPin, GPIO_PIN_SET);

  // 9. Check SCL High level in GPIOx_IDR.
  while (GPIO_PIN_SET != HAL_GPIO_ReadPin(i2c->sclPort, i2c->sclPin))
  {
    asm("nop");
  }

  // 10. Configure the SDA I/O as General Purpose Output Open-Drain , High level (Write 1
to GPIOx_ODR).
  HAL_GPIO_WritePin(i2c->sdaPort, i2c->sdaPin, GPIO_PIN_SET);

  // 11. Check SDA High level in GPIOx_IDR.
  while (GPIO_PIN_SET != HAL_GPIO_ReadPin(i2c->sdaPort, i2c->sdaPin))
  {
    asm("nop");
  }

  // 12. Configure the SCL and SDA I/Os as Alternate function Open-Drain.
  GPIO_InitStructure.Mode         = GPIO_MODE_AF_OD;
  GPIO_InitStructure.Alternate    = I2C_PIN_MAP;

  GPIO_InitStructure.Pin          = i2c->sclPin;
  HAL_GPIO_Init(i2c->sclPort, &GPIO_InitStructure);

  GPIO_InitStructure.Pin          = i2c->sdaPin;
  HAL_GPIO_Init(i2c->sdaPort, &GPIO_InitStructure);

  // 13. Set SWRST bit in I2Cx_CR1 register.
  i2c->instance.Instance->CR1 |= 0x8000;

  asm("nop");

  // 14. Clear SWRST bit in I2Cx_CR1 register.
  i2c->instance.Instance->CR1 &= ~0x8000;

  asm("nop");
  // 15. Enable the I2C peripheral by setting the PE bit in I2Cx_CR1 register
  i2c->instance.Instance->CR1 |= 0x0001;

  // Call initialization function.
  HAL_I2C_Init(&(i2c->instance));
}
```

edited Dec 18 '17 at 14:39
Ricardo
4,217 12 35 67

answered Jan 18 '17 at 13:55
AxelBe
31 3

Based on AxelBe response I created this version of the fix, without blocking operations, and adding the DeInit of the I2C instance. Hope you find this useful.

```
typedef struct
 {
    I2C_HandleTypeDef* instance;
    uint16_t sdaPin;
    GPIO_TypeDef* sdaPort;
    uint16_t sclPin;
    GPIO_TypeDef* sclPort;
} I2C_Module_t;

static uint8_t wait_for_gpio_state_timeout(GPIO_TypeDef *port, uint16_t pin,
GPIO_PinState state, uint32_t timeout)
 {
    uint32_t Tickstart = HAL_GetTick();
    uint8_t ret = TRUE;
    /* Wait until flag is set */
    for(;(state != HAL_GPIO_ReadPin(port, pin)) && (TRUE == ret);)
    {
        /* Check for the timeout */
        if (timeout != HAL_MAX_DELAY)
        {
            if ((timeout == 0U) || ((HAL_GetTick() - Tickstart) > timeout))
            {
                ret = FALSE;
            }
            else
            {
            }
        }
        asm("nop");
    }
    return ret;
}

static void I2C_ClearBusyFlagErratum(I2C_Module_t* i2c, uint32_t timeout)
{
    GPIO_InitTypeDef GPIO_InitStructure;

    I2C_HandleTypeDef* handler = NULL;

    handler = i2c->instance;

    // 1. Clear PE bit.
    CLEAR_BIT(handler->Instance->CR1, I2C_CR1_PE);

    //  2. Configure the SCL and SDA I/Os as General Purpose Output Open-Drain, High
level (Write 1 to GPIOx_ODR).
    HAL_I2C_DeInit(handler);

    GPIO_InitStructure.Mode = GPIO_MODE_OUTPUT_OD;
    GPIO_InitStructure.Pull = GPIO_NOPULL;

    GPIO_InitStructure.Pin = i2c->sclPin;
    HAL_GPIO_Init(i2c->sclPort, &GPIO_InitStructure);

    GPIO_InitStructure.Pin = i2c->sdaPin;
    HAL_GPIO_Init(i2c->sdaPort, &GPIO_InitStructure);

    // 3. Check SCL and SDA High level in GPIOx_IDR.
    HAL_GPIO_WritePin(i2c->sdaPort, i2c->sdaPin, GPIO_PIN_SET);
    HAL_GPIO_WritePin(i2c->sclPort, i2c->sclPin, GPIO_PIN_SET);

    wait_for_gpio_state_timeout(i2c->sclPort, i2c->sclPin, GPIO_PIN_SET, timeout);
    wait_for_gpio_state_timeout(i2c->sdaPort, i2c->sdaPin, GPIO_PIN_SET, timeout);

    // 4. Configure the SDA I/O as General Purpose Output Open-Drain, Low level (Write 0
to GPIOx_ODR).
    HAL_GPIO_WritePin(i2c->sdaPort, i2c->sdaPin, GPIO_PIN_RESET);

    // 5. Check SDA Low level in GPIOx_IDR.
    wait_for_gpio_state_timeout(i2c->sdaPort, i2c->sdaPin, GPIO_PIN_RESET, timeout);

    // 6. Configure the SCL I/O as General Purpose Output Open-Drain, Low level (Write 0
to GPIOx_ODR).
    HAL_GPIO_WritePin(i2c->sclPort, i2c->sclPin, GPIO_PIN_RESET);
    // 7. Check SCL Low level in GPIOx_IDR.
    wait_for_gpio_state_timeout(i2c->sclPort, i2c->sclPin, GPIO_PIN_RESET, timeout);

    // 8. Configure the SCL I/O as General Purpose Output Open-Drain, High level (Write 1
to GPIOx_ODR).
    HAL_GPIO_WritePin(i2c->sclPort, i2c->sclPin, GPIO_PIN_SET);

    // 9. Check SCL High level in GPIOx_IDR.
    wait_for_gpio_state_timeout(i2c->sclPort, i2c->sclPin, GPIO_PIN_SET, timeout);

    // 10. Configure the SDA I/O as General Purpose Output Open-Drain , High level (Write
1 to GPIOx_ODR).
    HAL_GPIO_WritePin(i2c->sdaPort, i2c->sdaPin, GPIO_PIN_SET);

    // 11. Check SDA High level in GPIOx_IDR.
    wait_for_gpio_state_timeout(i2c->sdaPort, i2c->sdaPin, GPIO_PIN_SET, timeout);

    // 12. Configure the SCL and SDA I/Os as Alternate function Open-Drain.
    GPIO_InitStructure.Mode = GPIO_MODE_AF_OD;
    GPIO_InitStructure.Alternate = GPIO_AF4_I2C2;

    GPIO_InitStructure.Pin = i2c->sclPin;
    HAL_GPIO_Init(i2c->sclPort, &GPIO_InitStructure);

    GPIO_InitStructure.Pin = i2c->sdaPin;
    HAL_GPIO_Init(i2c->sdaPort, &GPIO_InitStructure);

    // 13. Set SWRST bit in I2Cx_CR1 register.
    SET_BIT(handler->Instance->CR1, I2C_CR1_SWRST);
    asm("nop");

    /* 14. Clear SWRST bit in I2Cx_CR1 register. */
    CLEAR_BIT(handler->Instance->CR1, I2C_CR1_SWRST);
    asm("nop");

    /* 15. Enable the I2C peripheral by setting the PE bit in I2Cx_CR1 register */
    SET_BIT(handler->Instance->CR1, I2C_CR1_PE);
    asm("nop");

    // Call initialization function.
    HAL_I2C_Init(handler);
}
```

I execute this as soon as I get the HAL_BUSY response on the I2C HAL functions, here is an example:

```
    /* HAL Write */
    status = HAL_I2C_Mem_Write(eeprom_handler.instance, (uint16_t)device_address,
mem_addr_masked, I2C_MEMADD_SIZE_8BIT, src, bytes_to_write, 1000);
    if (HAL_OK == status)
    {
        bytes_written = bytes_to_write;
    }
    else if (HAL_BUSY == status)
    {
        I2C_ClearBusyFlagErratum(&eeprom_handler, 1000);
    }
```

edited Dec 18 '17 at 16:12
Bulkin
107 4

answered May 17 '17 at 18:47
Hugo Arganda
11 1

This is less an answer and more a warning... I got stuck with the same problem and the code above helped (along with stretching the I2C reset and initialising the clock before the GPIO - see linked answers).

However I spent about half a day more on this than I needed to because of a newbie mistake: *when you reset the STM32, in my case with an external programmer attached via SwD, you are usually **not** resetting your I2C peripheral.*

If you add the code above, or some other workaround, remember to occasionally try a power cycle or hard reset of your board so the I2C peripherals get reset too.

answered Apr 29 at 11:09
Happy Heyoka
13 3