

[Log in](#) to create and rate content, and to follow, bookmark, and share content with other members.



Can't reset I2C in STM32F407 to release I2C lines

Question asked by [Hess.Marco](#) on Jan 12, 2017

Latest reply on Mar 18, 2018 by Martin Round



Like • 0



Comment • 20

I am using an STM32F407 on a customer board and have trouble with the I2C handling.

I am using the latest HAL `HAL_I2C_MasterTransmitIT` and `HAL_I2C_MasterReceiveIT` to poll data from a sensor.

The problem is that after a while the I2C lines lock up (both SDA and SCL low) and the `HAL_I2C_MasterTransmitIT` returns errors.

I do detect the error and try to handle the problem by deinit the I2C. I then reconfigure the I/O to software handle some clocking that release SDA if it is stuck by a slave on the bus. At that point I can see that the lines go high again so it is the I2C peripheral in the STM that is pulling it low (I can also see that on the scope with the low level slightly above ground while when the lines are pulled by a slave device, they go lower).

I then reinit the I2C peripheral but as soon as the I2C is activated and connected to the I/O lines, the SDA and SCL are pulled low again.

So I can't seem to reset the I2C without restarting the whole chip.

Is there a other way to properly reset the I2C?

Also, what can be causing this lockup in the first place?

1 person has this question

OUTCOMES

Helpful(2)

Visibility:  STM32 MCUs Forum • 7446 Views

Last modified on Jan 12, 2017 8:18 AM

Tags: stm32 i2c master

0

20 Replies



AvaTar

Jan 12, 2017 9:34 AM

Also, what can be causing this lockup in the first place?

A lock-up of the slave device. I would check that the communication sequences match that of the slave's datasheet. That might require a scope or logic analyzer to pin down the critical sequence and/or transition. Perhaps even a timing issue or an ignored clock stretch.

Is there a other way to properly reset the I2C?

If you can't find an issue with your software (I2C master), you might have a problem. I guess your (undisclosed) I2C slave does not have a reset pin - they usually have not. A stopgap solution would be to power-cycle the I2C slave via a GPIO pin of the master.

"Older" I2C devices had been more or less combinatorial devices, but more recent ones contain quite state machines. When you abort a communication sequence, you are lost in the woods.

 Actions

 Like • 0



wacławek.jan

@ AvaTar on Jan 12, 2017 10:13 AM

You can reset the STM32's I2C module by setting and then clearing the respective I2CxRST bit in RCC_APB1RSTR.

Maybe even easier, you can use the SWRST bit in I2C_CR1.

JW

⚙ Actions

👍 Like • 0



Hess.Marco

@wacławek.jan on Jan 13, 2017 2:02 AM

Hi wacławek.jan,

That was the type of answer I was hoping for. Thanks!

Found the SWRST bit in the I2C_CR1 register and when I assert that bit just before I deinit the I2C, I can see it clears the bus lines so it seems to reset the I2C.

So my reset functions now clears the I2C peripheral, bus lines are back up. HAL driver reports no errors and status ready.

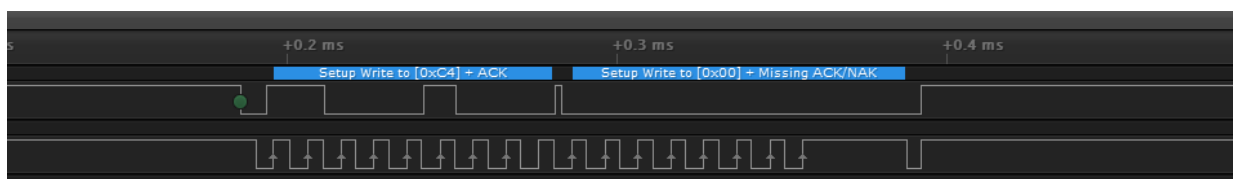
However, the next time it does a write operation, it locks up completely. HAL_I2C_MasterTransmitIT reports OK and then the SDA goes low for the start followed by the CLK also going low. However, both SDA and CLK now stay low until the timeout hits (20ms).

Again on the scope I can see from the low level on the SDA/SCL lines that is the master holding them low. If it is the slave the level would be lower.

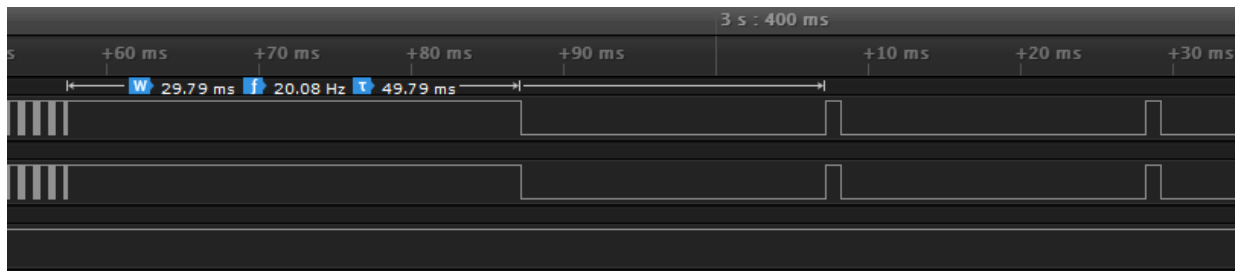
When I detect the timeout, I do my reset function again and that clears the SDA/CLK lines and then at the next write attempt the lines go low and stay low again.

So on my scope, I have this nice 1ms positive pulse repeating every 20ms :-(and communications does not recover.

I as able to trace on Saleae Logic the transactions leading up to the point of failure. It seems that the master just stops clocking and locks up in the middle of the operation.



After this, the bus stays high for 30ms or so and then both SCK and SDA go low. Then I hit a timeout and I start doing the resets resulting in the 20ms pulse train.



So I am still stuck :-(

Bit banging looks awfully good now :-(

Any further suggestions?

⚙ Actions

👍 Like • 0



wacławek.jan

@ Hess.Marco on Jan 13, 2017 11:18 AM

Want a suggestion? Drop Cube and write your own code.

You can also try the other reset method I mentioned (through APBxRSTR in RCC).

JW

⚙ Actions

👍 Like • 0



Hess.Marco

@ wacławek.jan on Jan 13, 2017 12:08 PM

Tried the RCC reset also and both with the SWRST. Still same problem :-(. I2C peripheral in the STM locks up hard after the reset on a first error.

When I look at that first image that I posted above, it looks like it clocked 8 bits and then the clock for the ACK/NAK is missing. Would the HAL driver have missed that event?

I am slowly getting used to the idea that I might have to drop the HAL code for the I2C but it is a not a happy thought. I already spend way more time on getting this @#\$% I2C to work properly than I have available.

Want to make sure I explored all other possible avenues before I go down that path.

 Actions

 Like • 0



Hess.Marco

@ AvaTar on Jan 13, 2017 2:02 AM

Thanks for the reply [AvaTar](#). I pretty much ruled out that it is the slave (a LIDAR Lite sensor) is causing this problem. When the bus is locked up, I pulled the slave off and the lines remained low, so it is pretty clear that the master is the one holding on to the bus. I also implemented a reset function that software clocks the CLK line to help clear the bus if it is the slave is the one holding on.

The problem I have is that during my reset operation, just the deinit and init of the I2C does not clear the bus lines

I can power cycle the slaves but I don't think the slaves are the problem.

Thanks for the response.

 Actions

 Like • 0



AvaTar

@ Hess.Marco on Jan 13, 2017 12:06 PM

Your first post sounded as the issue is probably on the slave side.

I pulled the slave off and the lines remained low, ...

I guess you are aware that the pull-up resistors must "stay in place" while pulling the slave, else you measure something else ...

BTW, I would second Jan's suggestion to drop Cube.

For the F407, you still have the SPL available, which exhibits much less "instabilities". Or go directly to the Ref.Manual, without library.

 Actions Like • 0**Hess.Marco**

@ AvaTar on Jan 13, 2017 12:15 PM

Yes, the pull ups are on my PCB.


The LIDAR Lite sensor is on a short cable attached to the PCB so it can be unplugged while the pull ups are still in place.

Have not used the SPL before, but it might be worth a try before going fully bare metal.

Thanks for the suggestions.

Regards,

Marco

 Actions Like • 0**waclawek.jan**

@ Hess.Marco on Jan 13, 2017 12:22 PM

Have not used the SPL before, but it might be worth a try before going fully bare metal.

No, its not.

My ~~4~~ 0.5c.

JW

 Actions Like • 0**Hess.Marco**

@ waclawek.jan on Jan 13, 2017 12:39 PM

Noted.



Like • 0

**waclawek.jan**

@AvaTar on Jan 13, 2017 12:19 PM

I guess you are aware that the pull-up resistors must "stay in place" while pulling the slave, else you measure something else ...

OH yes, have we talked about the pullup resistors already? What value are they? Have you had a look at the buses using an oscilloscope rather than just a LA? Are the levels and edges nice?

I recently had an encounter with I2C on a physically longer line being unreliable (on the slave side, though, and that slave did not implement glitch rejection which distinguishes good I2C incarnations from the poor ones). Setting the lowest slew rate (a.k.a. GPIO speed) helped more than anything else.

JW



Like • 0

**Hess.Marco**

@waclawek.jan on Jan 13, 2017 12:38 PM

Pullups originally were 4K7 with a 220R series to the STM. Later I lowered the pull ups to around 2K and a 50R series resistor but it did not make a difference as far as I could tell.

On an oscilloscope, the signals look good. Upgoing edge has a little rounding near the top but most of it is straight.

Low level on the slave side is less than 100mV and a bit lower than that when the slave ack's or sends data. So I can see a difference between master lows and slave lows.

I also played around with the bit rates and varies it anywhere between 10kHz to 200kHz (from the nominal 100kHz I intended to use) so when edges are a problem that should reduce when the clocks goes slower. No difference. Lockup occurs at around the same time.

I do have the GPIO on high speed. Should they be slow?

**wacławek.jan**

@ Hess.Marco on Jan 13, 2017 12:43 PM

I do have the GPIO on high speed. Should they be slow?

I don't say they *should* but that I had a good experience with that and it should make no harm as long as you are at 100kHz.

Shouldn't be related to your problem, though.

JW

**KIC8462852 EPIC204278916**

Jan 13, 2017 3:35 AM

I2C bus error recovery usually can happen when slave sends 0x00 and master resets during transmission, or in bus glitches. Bitbang 9 stop bits or more to get both lines raised in order to free the bus and be able to do a start bit again. Unless of a permanent clock stretching by slave, you can be sure of a turnaround bit within 9 clock periods. Smbus has clock stretch timeout as improvement feature. All the best!

**Piotrek.Piotrek**

May 1, 2017 5:24 PM

I would love to know if anybody managed to workaround this issue. I have exactly the same problem with i2c getting stuck after trying to reset it. (stm is pulling both lines down)

**KIC8462852 EPIC204278916**



@ Piotrek.Piotrek on May 1, 2017 6:01 PM

Bitbang 9 stop bits work for me even if the i2c lines are on a video cable with hot plug unplug. Without it, the bus hangs is few seconds of manipulation. Check bus idle before attempting to generate a start bit and then, when needed, run 9 stop bits prior to a start.

⚙ Actions

👍 Like • 0



Piotrek.Piotrek

@ KIC8462852 EPIC204278916 on May 1, 2017 7:28 PM

Hi, thanks for response! I spent all day trying different solutions and it looks like those are the steps to reliably restart stuck i2c (at least on my board STM32F446):

- > disable I2C
- > change sda & scl pin mode to output
- > generate 9 stop bits
- > change sda & scl pin mode back to alternate function
- > set SWRST flag in i2c control register
- > then enable i2c and initialize it

There's a helpful snippet of code in an answer to a very similar problem on stackexchange, which can be easily modified to generate more than one stop bit: [microcontroller - I2C busy flag strange behaviour - Electrical Engineering Stack Exchange](#)

1 person found this helpful

⚙ Actions

👍 Like • 2



Martin Round

Mar 17, 2018 12:59 PM

After a few days of frustration, I've pretty much proved to myself there are bugs in the HAL I2C (using interrupt) routines. I've an application that reads a MPU-6050 gyro via I2C. There are also other simple edge-detecting GPIO interrupts - nothing to do with the I2C. I find that the I2C works flawlessly for hours on end providing the other interrupts never occur. When activity begins on the GPIO lines causing other interrupts then the I2C comms stops at random - sometimes after a few minutes or sometimes within seconds. I've also found that using UART transmits with interrupt callback (to send log messages to a debug monitor) will also cause the I2C to stall.

If anyone has found a fix to this problem I'd be happy to know - I've spent hours searching but found nothing really helpful. I think it's time to ditch the HAL routines and research how to hit the I2C hardware more directly.

 Actions Like • 0**waclawek.jan**

@ Martin Round on Mar 17, 2018 3:20 PM

When activity begins on the GPIO lines causing other interrupts then the I2C comms stops at random - sometimes after a few minutes or sometimes within seconds.

I'm not going to defend Cube, but this may be not entirely software-related. Depending on particular setup, GPIO lines may introduce unexpected noise into the I2C lines (especially SDA), and the I2C machine may go in an unexpected way then. Check with exercising the GPIO and UART without having the interrupts enabled.

If this proves to be the problem, check and recheck the I2C timing settings, decrease pullup values, improve layout (grounds are often a neglected issue) or introduce timeouts/recovery mechanisms if these methods won't help.

If hardware proves to be innocent, it's time to ditch Cube (frankly, it's always a good time to ditch Cube, except perhaps when beginners try to make applications straightforwardly fitting the provided examples; but it's also a bad idea to take substantial measures without being sure of the primary cause of problem).

Also, note, that there are several rather different incarnations of the I2C in various STM32 subfamilies, so it always helps to state which model are you using.

JW

1 person found this helpful

 Actions Like • 0**Martin Round**

@ waclawek.jan on Mar 18, 2018 9:44 PM

Thanks for the suggestions. The chip is an STM32F103C8T6. All the signals (Power, I2C, and GPIO) look clean on a scope (I fitted extra decoupling capacitors to the chip's power pins a week or so back but it made no difference). The I2C is set to run at 400kHz (and the scope verifies that it is indeed running at that speed). I'm using 3k3 pull-ups on SDA and SCL.

I found that simply adding the following lines before entering my main while() loop - and changing nothing else - prevents the I2C from locking up even when the GPIO is in heavy use.

```
HAL_NVIC_DisableIRQ(EXTI1_IRQn);  
HAL_NVIC_DisableIRQ(EXTI9_5_IRQn);  
HAL_NVIC_DisableIRQ(EXTI2_IRQn);  
HAL_NVIC_DisableIRQ(EXTI3_IRQn);  
HAL_NVIC_DisableIRQ(EXTI4_IRQn);  
HAL_NVIC_DisableIRQ(EXTI15_10_IRQn);
```

The interrupt routines (when they are enabled) are very short - they just increment or decrement some 16-bit variables - and those variables aren't used at all by the I2C routines.

I also tried using the non-interrupt (blocking) I2C HAL routines - the I2C doesn't then lock up, but my program then does miss some edges on the GPIO interrupts - so it looks like those HAL routines must disable interrupts, meaning those are no use to me either.

I was going to try the more complicated DMA I2C HAL routines, but I think I'll still need an interrupt callback when the transactions are complete, so I'm not inclined to waste any more time on those.

I shall now peruse the chip's data sheet and see what's involved in bypassing the HAL I2C code. If I get stuck with that, I suppose I could always write some low level bit-banging code to just do I2C on any pair of GPIO pins - it will occupy a lot of the chip's time, but my time is more valuable than the chips!

 Actions

 Like • 0

Recommended Content

[2018 STM32 Wish List](#)

[STM32-C2C \(Cellular to Cloud\) Web Concierge Portal](#)

[Efficiently use DMA with UART RX on STM32](#)

[Having problems with STM32F7 TIM1 PWM Mode 1 Output pins](#)

[2018 Wish List](#)

Incoming Links

[Re: Problem communication STM32F405VG and I2C EEPROM \(M24M02-DRMN\)](#)

All rights reserved © 2016 STMicroelectronics

[Terms of Use](#) [Privacy Policy](#)



[Home](#) | [Top of page](#) | [Help](#)