200990[PostContent].html

Hi, I've got a weird problem with I'd like some advice on. I've configured an STM32F334 as a I2C Slave, and am using Sequential IT driven operation with variable length packets (e.g. first byte received is length byte, followed by data packet of variable length)

Everything works exactly as planned - EXCEPT in a certain situation where the receive ISR was taking too long, and subsequently the *HAL_I2C_EnableListen_IT()* command was not getting called in time for the next 'read' request coming on the I2C line from the Master.

Subsequently HAL_I2C_AddrCallback() was not called, and the packet was missed...to be expected.

What was not expected was that this would cause the STM to pull the SDA line LOW, and not release it. The STM code continued to operate as if nothing had gone wrong, except now the I2C bus was blocked.

* No error states reported by HAL_I2C_GetError()

* HAL_I2C_GetState() was reporting HAL_I2C_STATE_LISTEN as expected.

* HAL I2C ErrorCallback() was not called

I tried sending 9+ clock pulses from the master to release the bus, but to no avail.

I've changed my code so the ISR finishes much more quickly (and everything works normally) - but I'm worried as to how this could get the I2C to lock up so horribly? Especially with no errors being reported.

void HAL_I2C_AddrCallback(I2C_HandleTypeDef *hi2c, uint8_t TransferDirection, uint16_t AddrMatchCode)

```
TRACE PA1 HI;
     if(AddrMatchCode == 222)
     {
         gTransferDirection = TransferDirection;
        if (TransferDirection == I2C DIRECTION TRANSMIT)
        {
              /* Master is sending command and data payload*/
             gI2CReceiveIndex = 0;
             gI2CPayloadSizeReceived = 0;
              HAL_I2C_Slave_Sequential_Receive_IT(&hi2c1, &gI2CPayloadSize, 1, I2C_FIRST_FRAME); // setup to receive payload size first
        }
        else
        {
             switch (gI2CLastReadCommand)
              {
                   case HRD READ TEMPERATURES 16BIT:
                        gRxBuffer[0] = HI(gLedTemp);
                        gRxBuffer[1] = LO(gLedTemp);
                        HAL_I2C_Slave_Sequential_Transmit_IT(&hi2c1, (uint8_t *)gRxBuffer,2, I2C_FIRST_AND_LAST_FRAME);
                        break;
              }
         }
    }
}
// After listen is complete, re-enable listen ISR
void HAL_I2C_ListenCpltCallback(I2C_HandleTypeDef *hi2c)
{
    HAL_I2C_EnableListen_IT(&hi2c1);
}
void HAL_I2C_SlaveRxCpltCallback(I2C_HandleTypeDef *hi2c)
{
     if (gI2CPayloadSizeReceived == 0)
         gI2CPayloadSizeReceived = 1;
         HAL_I2C_Slave_Sequential_Receive_IT(&hi2c1, (uint8_t *)gRxBuffer, gI2CPayloadSize, I2C_LAST_FRAME);
         return;
TRACE PB0_HI;
    gI2CPayloadSizeReceived = 0;
    if (gTransferDirection == I2C_DIRECTION_TRANSMIT)
     {
         switch(gRxBuffer[0])
         {
              case HRD_SETUP_READ_CMD:
                   gI2CLastReadCommand = gRxBuffer[1]; // save read command byte
                   break;
              case HRD_COMMAND_WHICH_TAKES_TOO_LONG:
                   doSomethingWhichTakes50us(); // this causes SDA to be pulled LOW and can't recover
                   break;
         }
This timing snapshot shows the issue:
Point A: This is a trace on HAL I2C AddrCallback()
```

Point B: This is a trace on doSomethingWhichTakes50us() which causes the next I2C transmission to be missed

Point C: This is a trace on HAL_I2C_ListenCpltCallback() which calls HAL_I2C_EnableListen_IT() fractionally too late

9 % : 442 m% : 500 ps 9 % : 442 m% : 600 ps	9 s : 442 ms : 700 µs
A contraction of the second se	
I2C write command: HRD_COMMAND_WHICH_TAKES_TOO_LONG	
	released

Does anyone have any ideas on why this is happening, and how to get out of a sticky situation like this? How can I detect that this has even happened on the STM?

I'm using STM32Cube FW_F3 V1.9.0