AN1095: What to Do When the I2C Master Does Not Support Clock Stretching

The I2C Slave (I2CSLAVE0) interface of EFM8LB1 can support clock stretching for cases where the core may be temporarily prohibited from transmitting a byte or processing a received byte during an I2C transaction, and it can be used during a transfer in order to allow slower slave devices to communicate with faster masters.

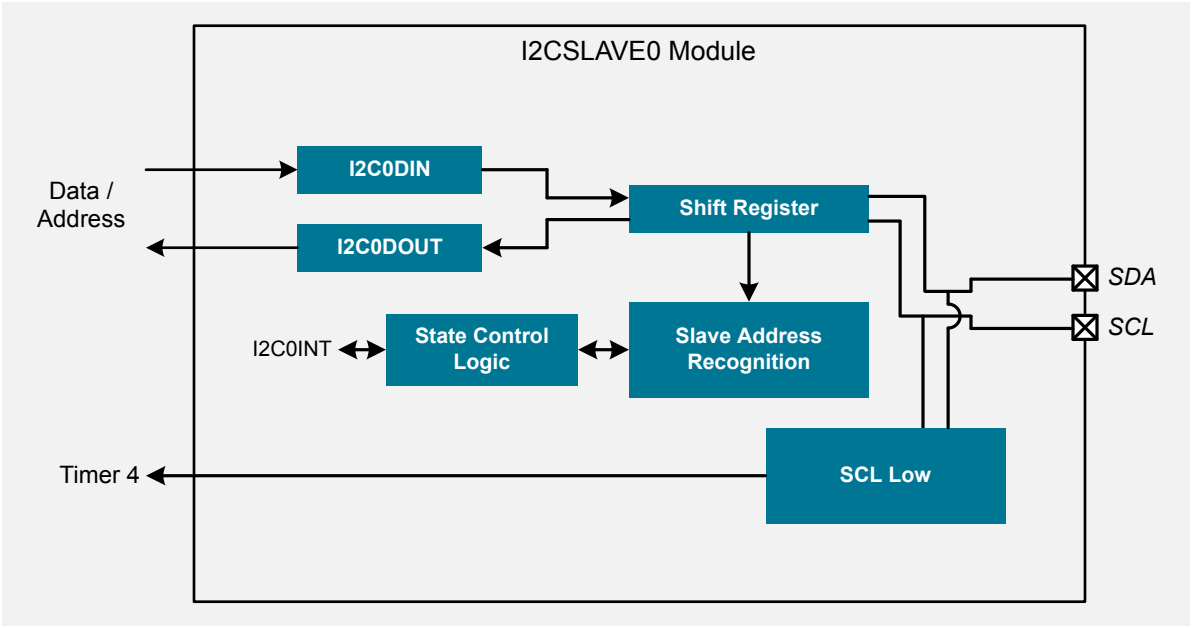
However, sometimes the host may not support clock stretching and will still try to transfer data through SDA despite the status of the I2C SCL. In this case, the data will be lost or be detected as an invalid value.

Generally, the effective way to avoid clock stretching is enhancing the system clock of the MCU to reduce the time the ISR takes to execute, but this method is limited by the maximum speed of the MCU.

This document will introduce how to deal with the case where I2C master doesn't support clock stretching by using of RXTH and TXTH in I2C fast plus mode.

KEY POINTS

- Clock stretching introduction.
- Avoiding the clock stretching caused by WR interrupts with RXTH.
- Avoiding the clock stretching caused by RD interrupts with TXTH.



1. Introduction

The I2C peripheral contains separate two-byte FIFOs for RX and TX paths, and the shift registers for I2C Write and Read transfers are also separate. All of the received data will be stored in the RX FIFO, and reading the I2C0DIN register can read data from RX FIFO until RXE is set to 1, indicating that there is no more data in the RX FIFO. All of the transmit data should be transferred into the TX FIFO by writing I2C0DOUT. The hardware sets the TXNF bit to 1, which indicates that there is more room available in the TX FIFO.

Using the Receive FIFO Request (RFRQ) and Transmit FIFO Request (TFRQ) interrupts is an effective way to enhance the real-time data processing of the system to avoid clock stretching.

The RXTH field of the I2C0FCN0 register will configure when the hardware sets the Receive FIFO Request bit (RFRQ). The RFRQ bit is set whenever the number of bytes in the RX FIFO exceeds the value of RXTH, and if the RFRQE bit be set to 1 at the same time, an I2C0 interrupt will be generated.

Similarly, the TXTH field of register I2C0FCN0 configures when hardware will set the Transmit FIFO Request bit (TFRQ). TFRQ is set whenever the number of bytes in the TX FIFO is equal to or less than the value in TXTH, and an I2C0 interrupt will be generated if the TFRQE bit is set to 1.

2. Avoiding the Clock Stretching Caused by WR Interrupts with RXTH

The I2C Write sequence with the I2C Slave peripheral consists of a series of interrupts and required actions in each interrupt. The write sequence consists of the following steps:

1. Initialize the I2C peripheral and set the RXTH to ZERO, also enabling the RFRQE at the same time. This ensures the Receive FIFO Request (RFRQ) interrupt will be triggered by each receiving byte.
2. An incoming START + Address + W byte causes the peripheral to automatically ACK a matching address if BUSY is cleared to 0.
3. Firmware reads one or more bytes of data from the RX FIFO on each Receive FIFO Request (RFRQ) interrupt.
4. The master sends a STOP when the entire data transfer completes.

The FACS bit in register I2C0ADM controls whether I2C0INT will be set (clock stretching also be enforced at the same time) after a matching slave address has been acknowledged. When this bit is set, clock stretching always occurs after an ACK of the address byte until firmware clears the I2C0INT bit. When this bit is cleared, the I2C0INT bit won't be set by the address ACK alone, but it may be set due to RD or WR.

This WR bit will only be set by hardware on the 9th SCL falling edge when the RX FIFO is full when the FACS bit is cleared. However, because the receiving byte will be read in time with the Receive FIFO Request (RFRQ) interrupt, the RX FIFO will never be full, and the WR interrupt won't be triggered.

The following figure describes the I2C Write sequence and firmware actions in each interrupt.

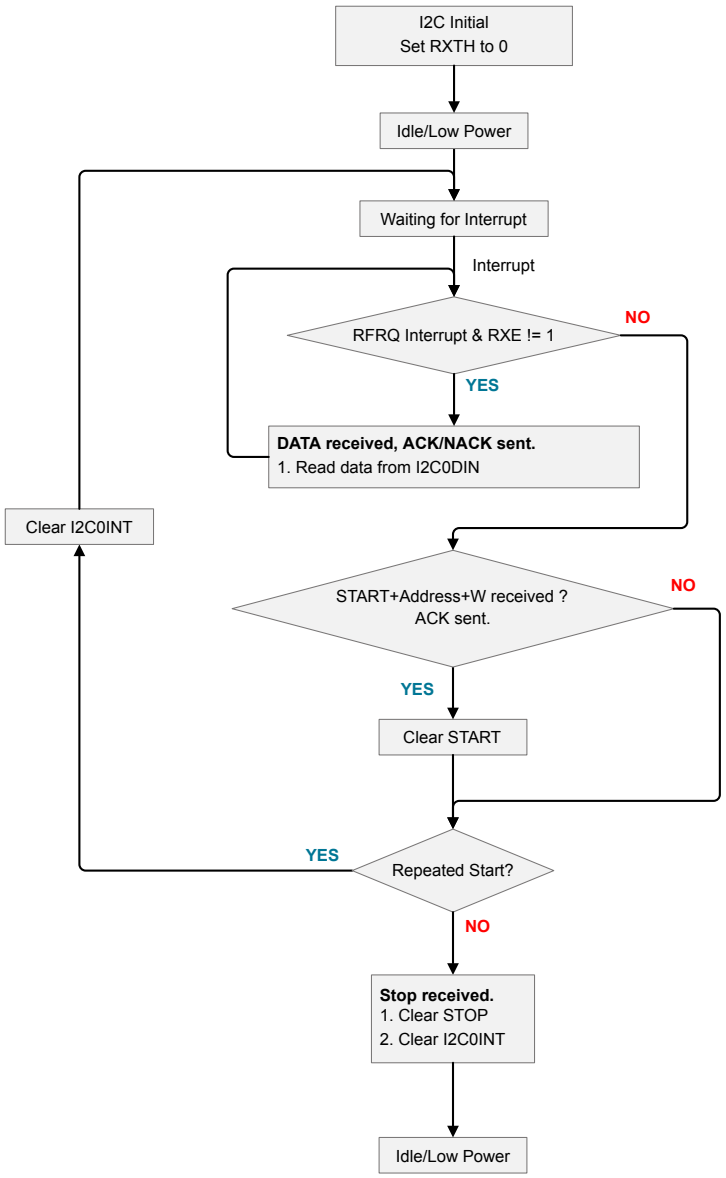


Figure 2.1. I2C Write Flow Diagram with the I2C Slave Peripheral

The following figures show the Fast Plus mode of EFM8LB1 with the I2C master JI-300. The SCL frequency of master JI-300 should be set to 1 Mbps, the duty cycle is 50%, and the default pull-up resistor value is 1 kΩ.

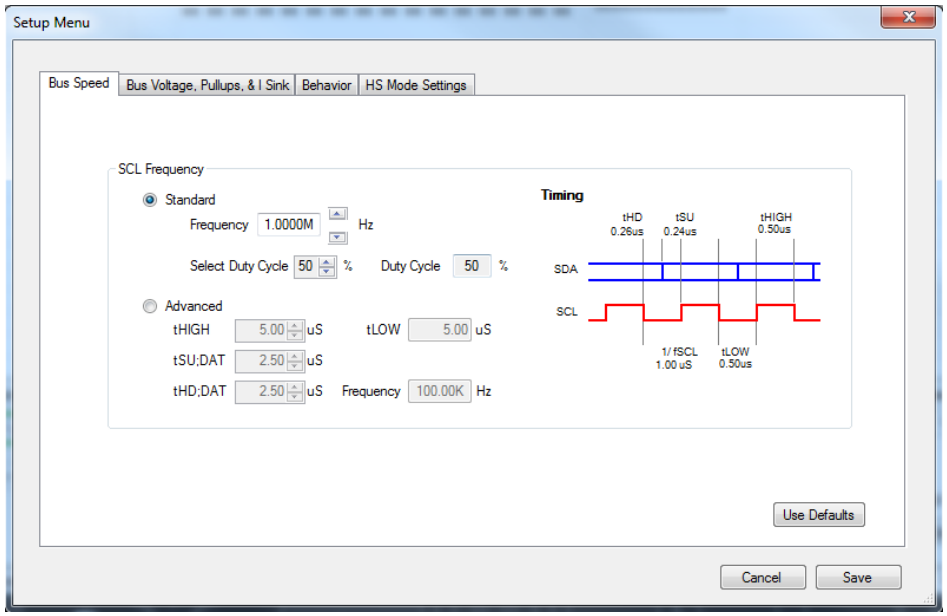


Figure 2.2. Bus Speed Setting of Master JI-300

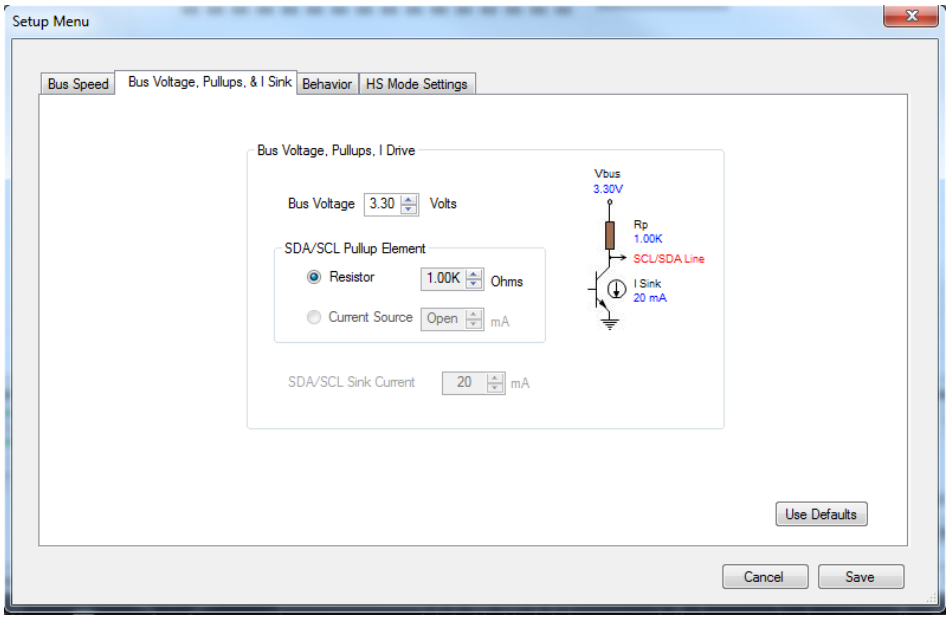


Figure 2.3. Pullup Resistor Setting of Master JI-300

After setting up the basic environment, a write message can be added to the message List to write 32 bytes to the EFM8LB1 continuously.

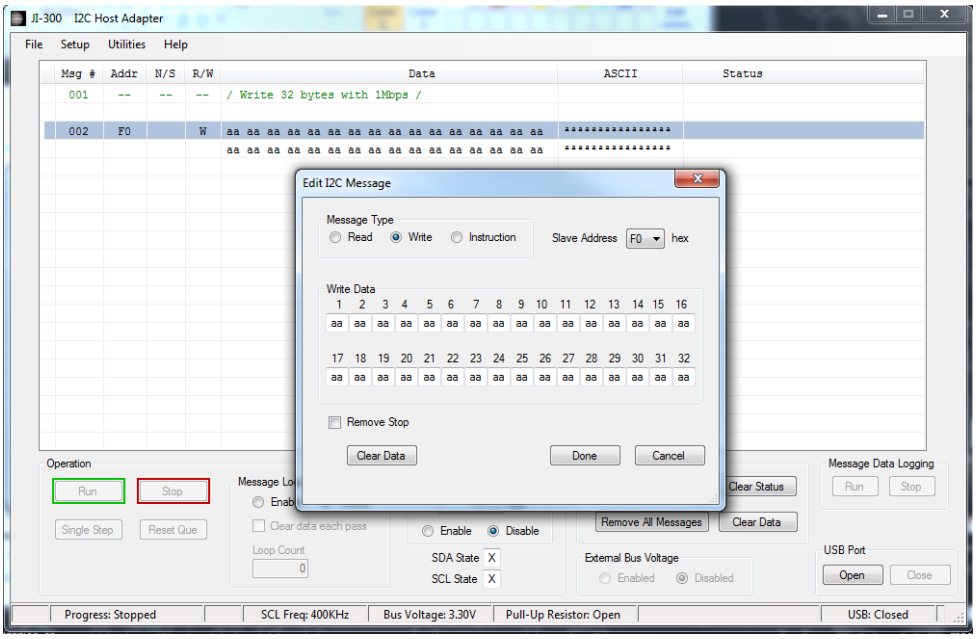


Figure 2.4. Write Message

These 32 bytes data will be received by the I2C SLAVE without clock stretching.

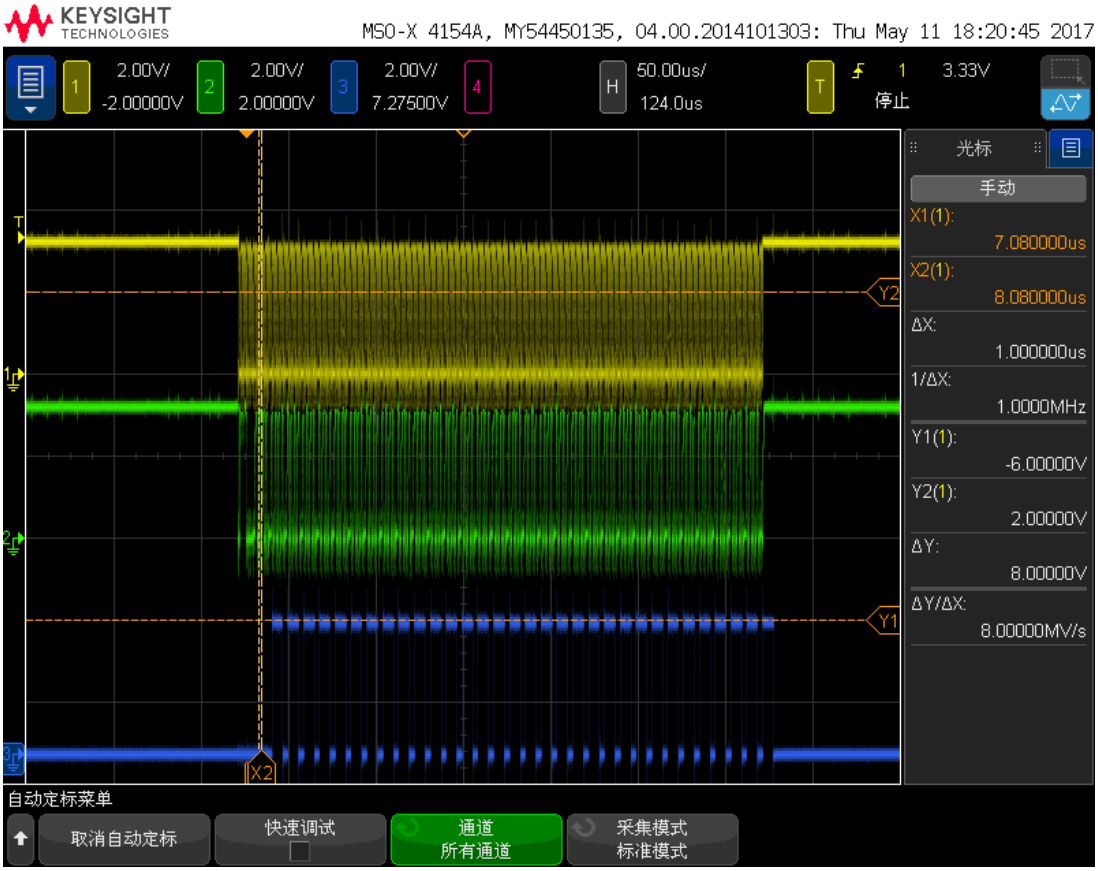


Figure 2.5. Write 32 Bytes Data without Clock Stretching

A GPIO signal can be used to indicate the interrupt processing flow (Blue waveform from the screenshot of oscilloscope). During operation, the Receive FIFO Request (RFRQ) interrupt will be triggered by each receiving byte following the START+ADDRESS+W. The I2C Slave peripheral still can receive another two bytes (1 byte in the FIFO and 1 byte in the shift register) during data processing since the RX FIFO is still not full.

To ensure the FIFO does not overflow, the maximum data processing time for each byte should be no longer than two byte receiving times (9 x 2 SCL clock) to avoid the hardware setting the WR bit , triggering an I2C0INT interrupt, and clock stretching.

If the RXTH field is set to ONE, the receive FIFO request interrupt will be triggered after receiving two bytes following the START+ADDRESS+W, and the I2C Slave peripheral still can receive another one byte during data processing since the RX shift register is still not full.

The advantage of bigger RXTH settings is that fewer I2C interrupts will be triggered; however, the firmware also has less time to process every two data bytes to avoid the clock stretching. From the experiment results, the recommended value of the RXTH field is ZERO to reserve more time to reserve more time to process received data.

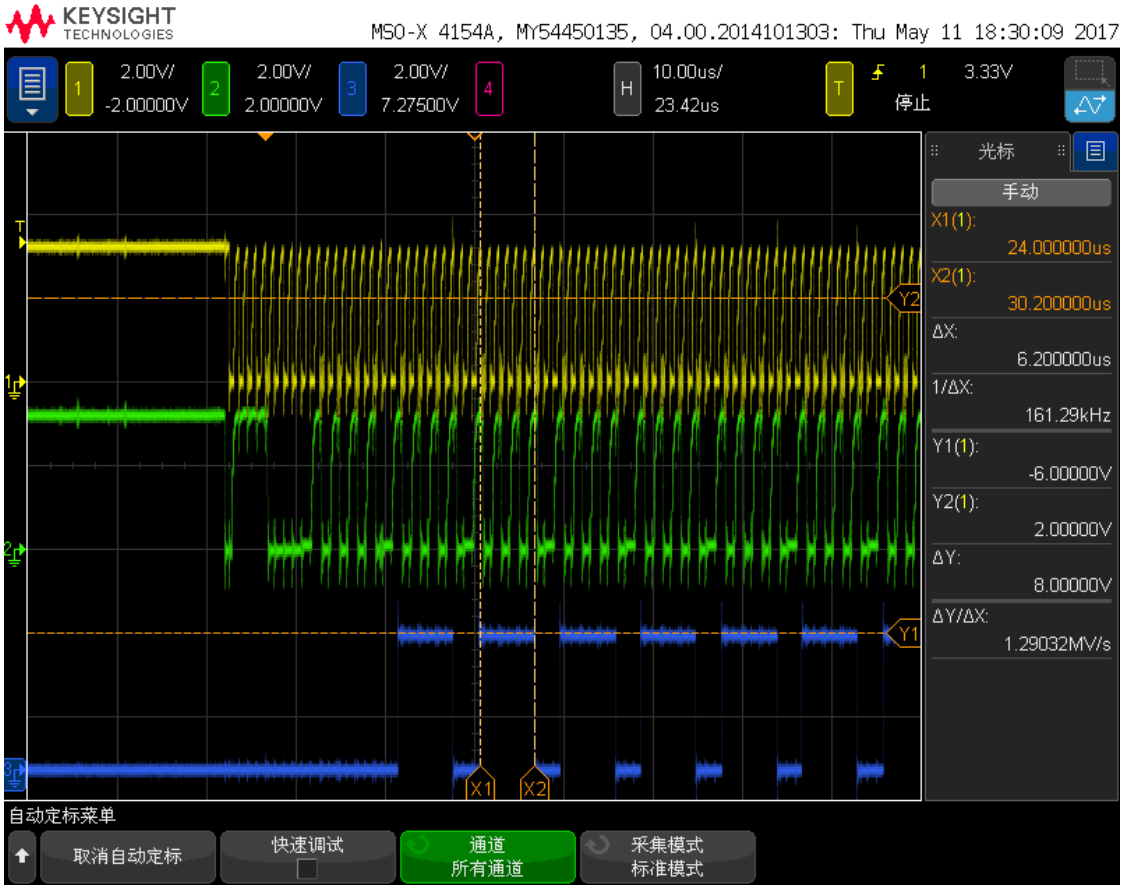


Figure 2.6. RFRQ Interrupt Processing

3. Avoiding the Clock Stretching Caused by RD Interrupts with TXTH

The I2C Read sequence with the I2C Slave peripheral consists of the following steps:

1. Initialize the I2C peripheral and set the TXTH to ONE, enabling the TFRQE bit at the same time. This ensures the Transmit FIFO Request (TFRQ) interrupt will be triggered whenever the number of bytes in the TX FIFO is equal to or less than ONE.
2. Firmware fills the TX FIFO on each Transmit FIFO Request (TFRQ) interrupt until the TX FIFO is full.
3. After receiving an incoming START + Address + R byte, the I2C peripheral will automatically ACK a matching address if BUSY is cleared to 0.
4. A Transmit FIFO Request (TFRQ) interrupt occurs after one byte is shifted from the TX FIFO to the shift register. The firmware will fill the TX FIFO until the TX FIFO is full or all of the ready data is written into the FIFO.
5. The master sends a NACK when the current data transfer completes and either a repeated START or STOP.
6. The master sends a STOP when the entire data transfer completes.
7. Firmware clears the STOP interrupt and reloads the TX FIFO, if necessary.

The following figure describes the I2C Read sequence and firmware actions in each interrupt.

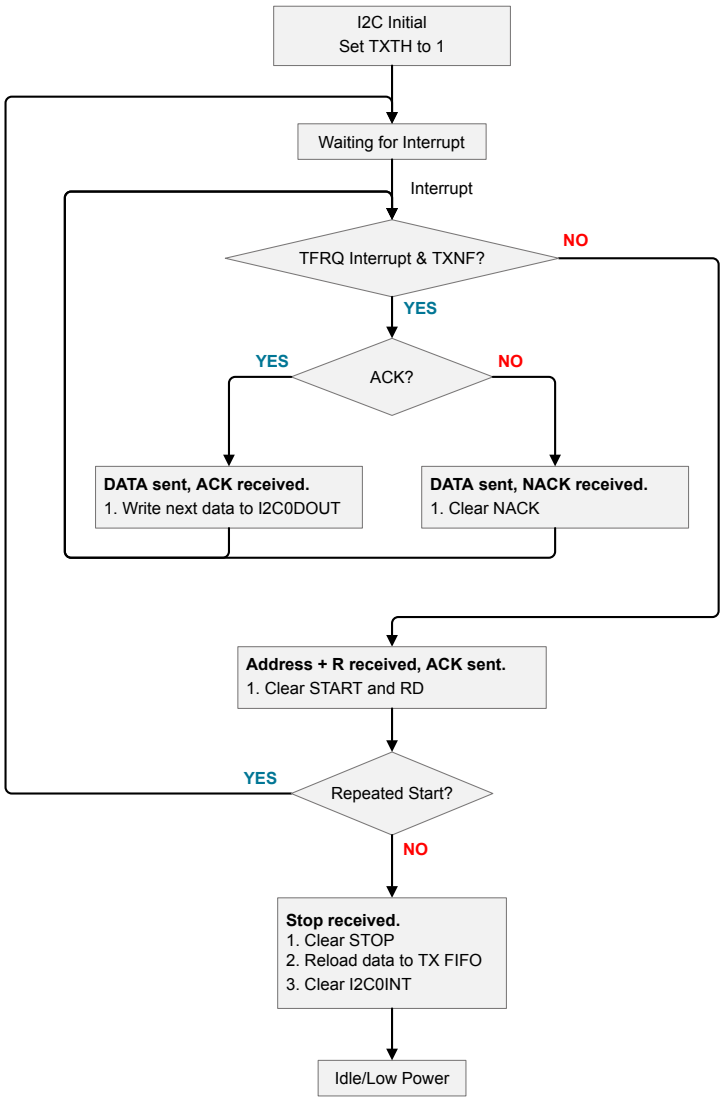


Figure 3.1. I2C Read Flow Diagram with the I2C Slave Peripheral

The following figure is the scope capture of 32 bytes of data read from the I2C SLAVE without clock stretching. After initializing the I2C Slave and setting the TXTH field to ONE with TFRQE enabled, the Transmit FIFO Request (TFRQ) interrupt will be triggered, and firmware will fill the TX FIFO with the available data until the FIFO is full.

After receiving START+ADDRESS+R, and the first byte in the TX FIFO is shifted to the shift register. When the number of bytes in the TX FIFO equals the value in TXTH, the Transmit FIFO Request (TFRQ) will be triggered again.

There is still one byte in the TX FIFO during firmware processing of the transmitted data, and the maximum processing time should no longer than the transmitting time of 2 bytes (1 remaining byte in the TX FIFO and 1 byte in the shift register).

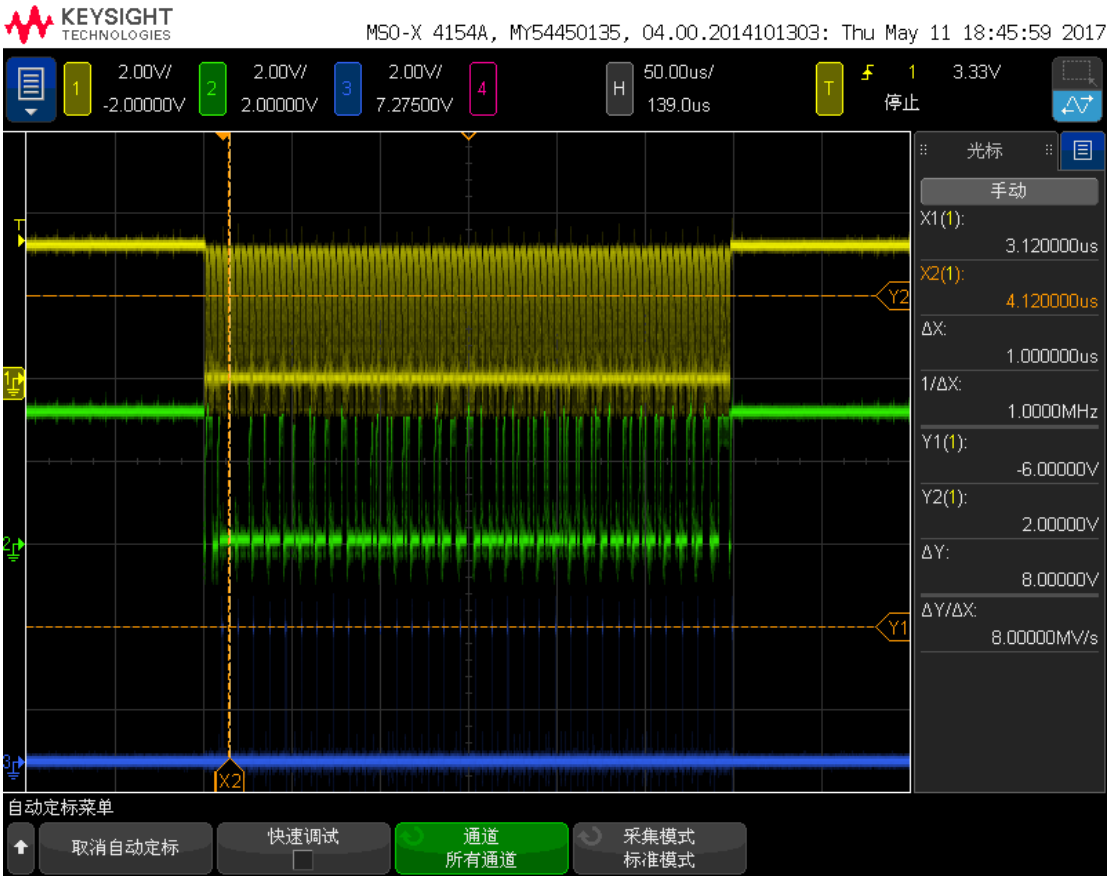


Figure 3.2. Read 32 Bytes of Data Without Clock Stretching

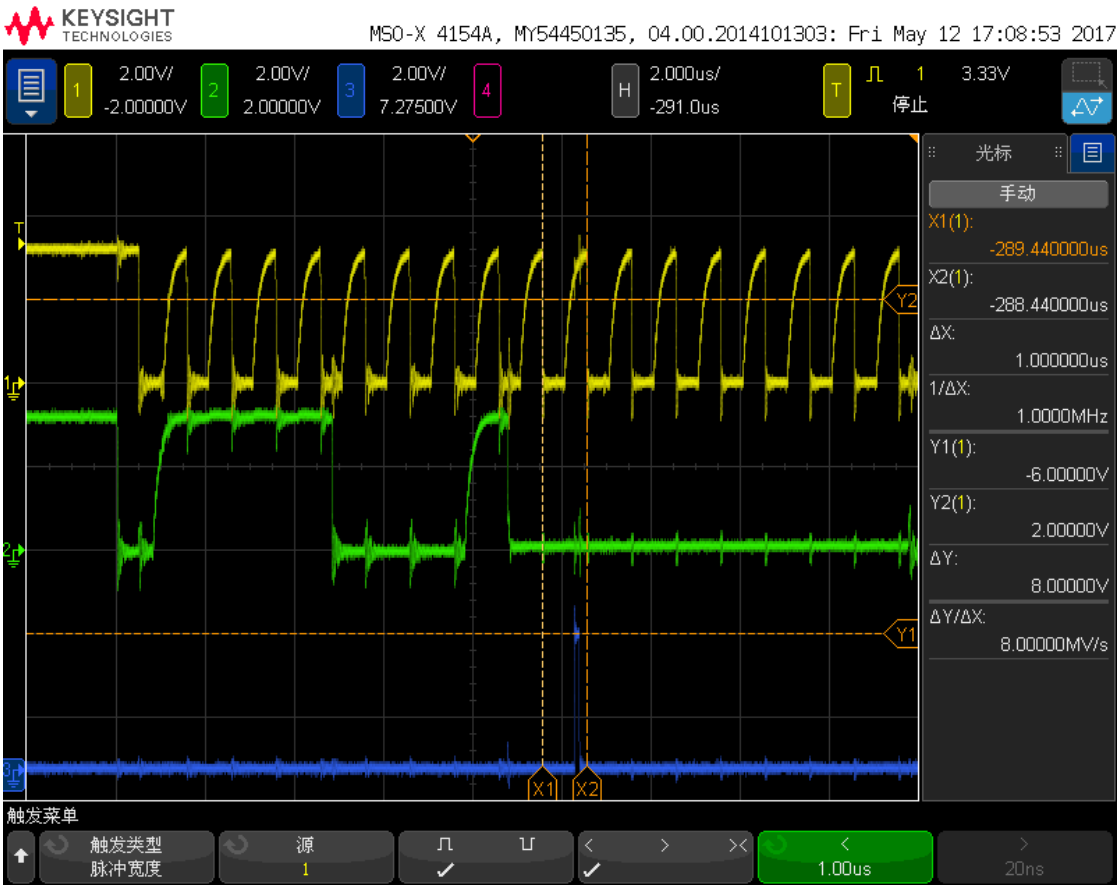


Figure 3.3. Read Data Without Clock Stretching

The master will send a STOP when the entire data transfer completes, the interrupt will be triggered, and firmware should reload the data into the TX FIFO on the STOP interrupt.

The following figure displays the TX FIFO structure of the I2C slave peripheral. This FIFO consists of 2 bytes and a 1-byte shift register. The data in the first FIFO byte will be shifted to the shift register when a read occurs. For example, if there are 31 bytes be transmitted from the I2C slave to the master successfully before STOP, and the 32nd byte has been shifted to the shift register, the TX FIFO can be filled with the 33rd and 34th bytes. The 32nd byte in the shift register will be flushed by the 33rd byte in the TX FIFO when the next read occurs, and 32nd byte will be lost during the data transfer.

Because of this, it is necessary to flush and reload the TX FIFO with the previous bytes after receiving the STOP interrupt.

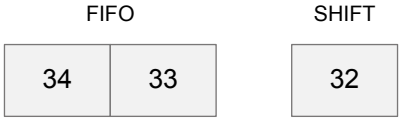


Figure 3.4. TX FIFO & SHIFT Register

After flushing and reloading, the 32nd and 33rd bytes will exist in the TX FIFO to avoid losing one byte.

Figure 3.5. Flush and Reload TX FIFO

The following figure demonstrates this result of continuous reading and writing without losing data.

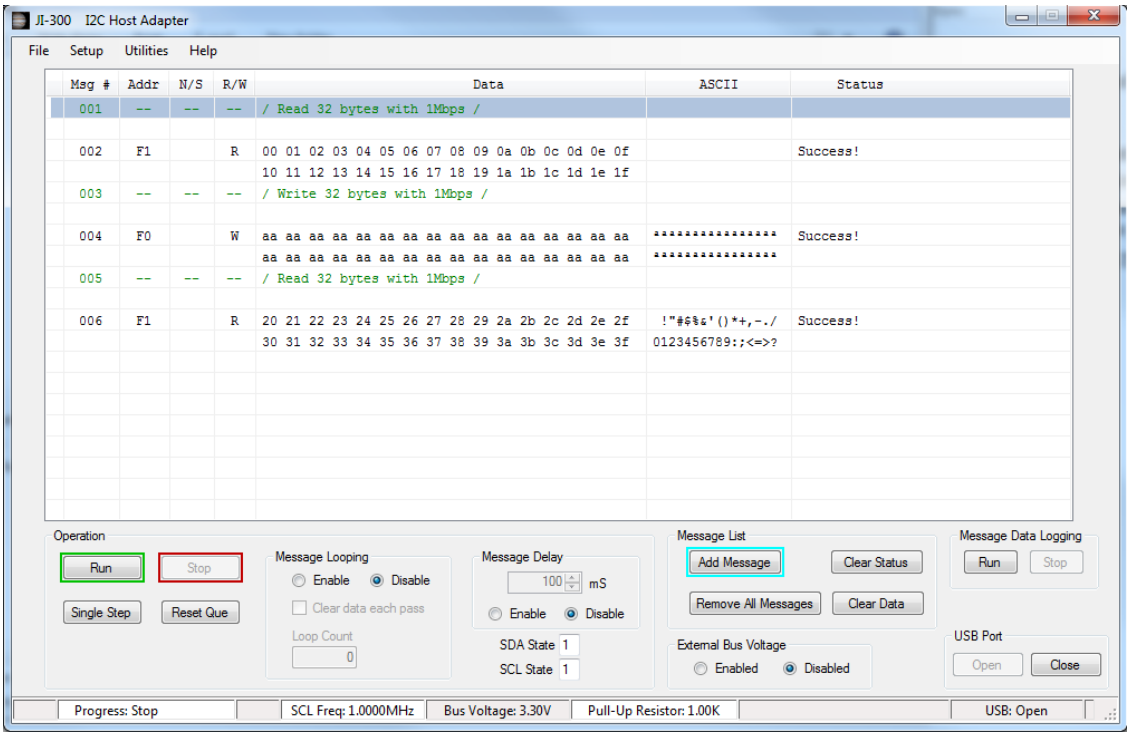
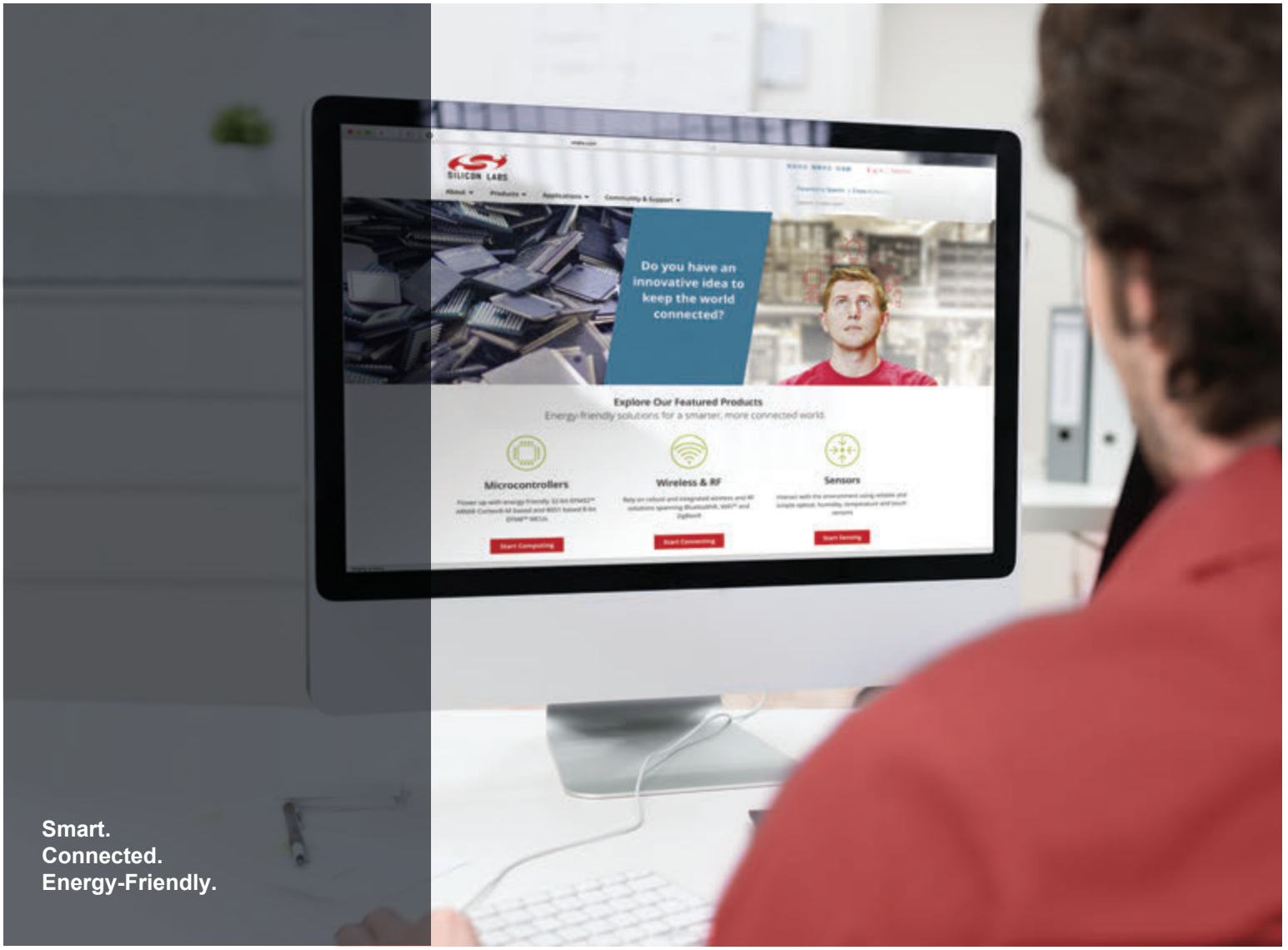


Figure 3.6. Continuous Reading and Writing Without Data Loss

4. Software Example

An I2C slave example called [**EFM8LB1_I2C_Slave_FIFO**] is included in the 8-bit SDK that demonstrates using the RXTH and TXTH fields to avoid clock stretching. This example can be found in Simplicity Studio (www.silabs.com/simplicity).



Products
www.silabs.com/products

Quality
www.silabs.com/quality

Support and Community
community.silabs.com

SILICON LABS®

Silicon Laboratories Inc.
400 West Cesar Chavez
Austin, TX 78701
USA

http://www.silabs.com