Application Note

BQ79718, BQ78706, and BQ79731 Application Guide for I2C Communication With EEPROM

Harrison Yang
Yang Wu

ABSTRACT

This document was translated from a simplified Chinese source. (ZHCAGA5)

The GPIOs of TI stackable battery monitoring and balancing chips (referred to as AFEs) BQ797xx and BQ787xx support configuration as I2C controllers, which can be used for EEPROMs, temperature and humidity sensors, and other I2C communication ICs. Connecting an AFE to an EEPROM enables storing battery information locally within the battery pack, which can be used for battery passport purposes or to better understand battery status during second-life battery utilization. Connecting the BQ79731 (referred to as BJB) to an EEPROM allows storing parameters such as shunt resistor value and temperature drift, which the MCU can utilize to improve current sampling accuracy.

This application note introduces the EEPROM read and write protocols and reminds users to pay attention to application issues such as EEPROM address roll-over, self-timed write cycle, and the AFE requirement that the interval between I2C read commands must be <20ms. In addition, this application note provides hardware modification schemes for connecting the BQ79718 EVM to an EEPROM, as well as software commands for I2C communication via the EVM host GUI, enabling users to quickly master and debug the I2C communication features of the AFE and BJB.

Table of Contents

List of Figures

List of Tables

1 Introduction to I2C Features for AFE and BJB

Figure 1-1 shows the I2C application block diagram from the BQ79718 datasheet; this architecture is also applicable to BQ78706 and BQ79731. Each AFE in the daisy chain can connect to multiple I2C slave nodes.

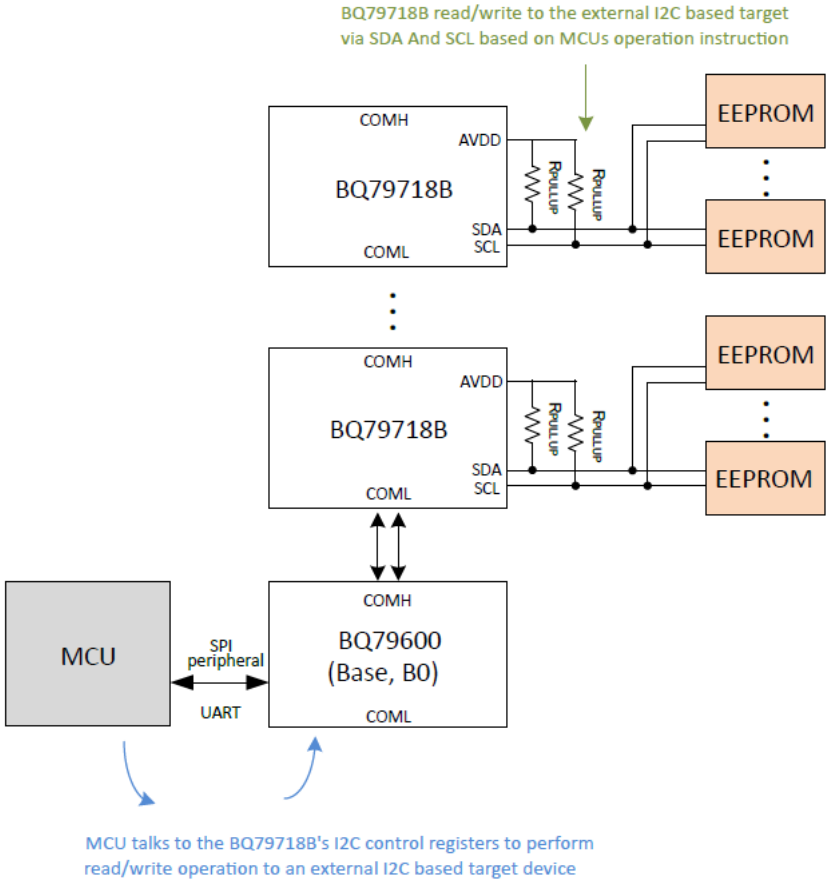

Figure 6-40. I²C Controller Function

Figure 1-1. BQ797xx I2C Application Diagram

I2C data transmission and reception mainly rely on the I2C_CTRL, I2C_WR_DATA, and I2C_RD_DATA registers, prior to which GPIO functional configuration (I2C enable) must be completed. The MCU writes data into the AFE register I2C_WR_DATA via the daisy chain, and the read-back data is stored in I2C_RD_DATA. I2C_CTRL configures SEND, RECEIVE, START, STOP, NACK, and initiates I2C communication.

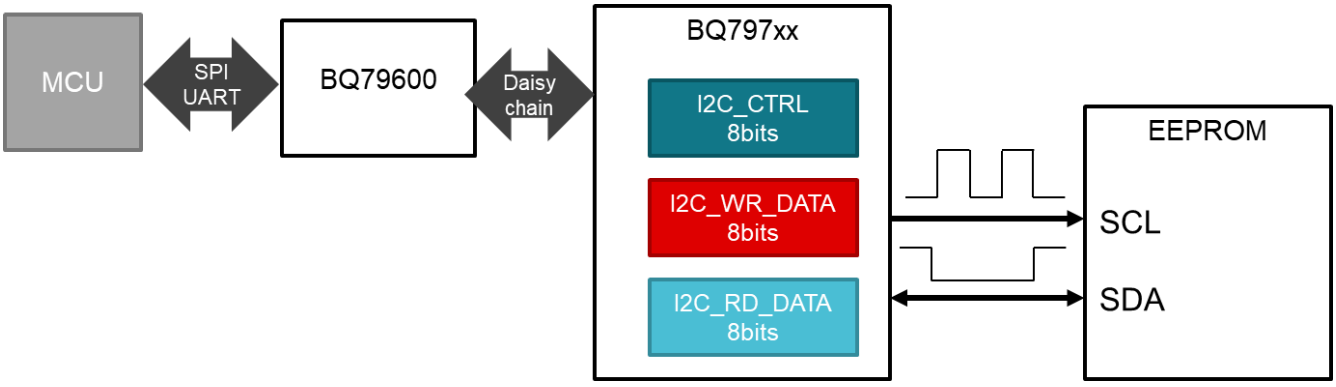

Figure 1-2. AFE and BJB I2C Related Registers

I2C_WR_DATA: Stores data to be written into the EEPROM.

I2C_RD_DATA: Stores data read from the EEPROM.

I2C_CTRL: START, NACK, and STOP before and after data in the I2C protocol are configured through this register. It also defines whether the current communication is transmit or receive, and initiates I2C communication.

NAME	ADDR	TYPE	RESET VALUE	DATA							
				B7	B6	B5	B4	B3	B2	B1	B0
I2C_CTRL	0x371	R/W-AC	0x00	RSVD	RSVD	SEND	RECEIVE	START	STOP	NACK	I2C_GO

Figure 1-3. I2C_CTRL Register Definition

The host node (AFE or BJB) controls START and STOP to inform the EEPROM of communication start and end.

After receiving 8bit data, the slave node (EEPROM) pulls SDA low for 1bit (ACK) to inform the host that data reception is complete and the next byte can be sent.

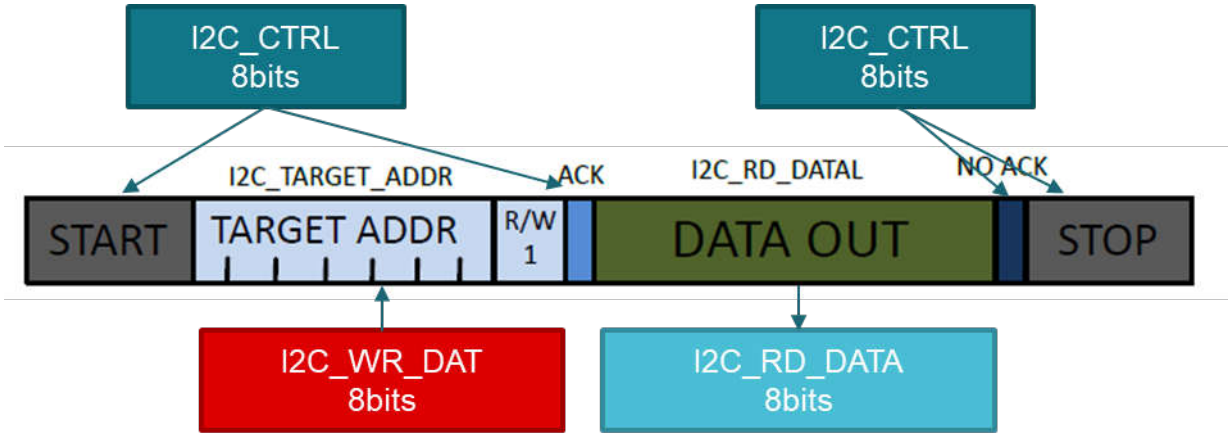

Figure 1-4. Mapping Between I2C Communication Protocol and Registers

For detailed I2C communication protocols, please refer to the application report *Understanding the I2C Bus*.

The EEPROM read communication process is divided into the following 7 steps:

- Step 1: The MCU places data into the AFE register I2C_WR_DATA via the daisy chain.
- Step 2: The MCU writes I2C_CTRL[I2C_GO] =1 to command the AFE to perform I2C communication.
- Step 3: Upon receiving the command, the AFE begins I2C communication with the EEPROM.
- Step 4: Send the I2C read command.
- Step 5: Data returned via I2C is stored in I2C_RD_DATA.
- Step 6: The MCU reads the data from the AFE register.
- Step 7: The MCU reads the FAULT_SYS register of the AFE to verify that no anomalies occurred during this communication.

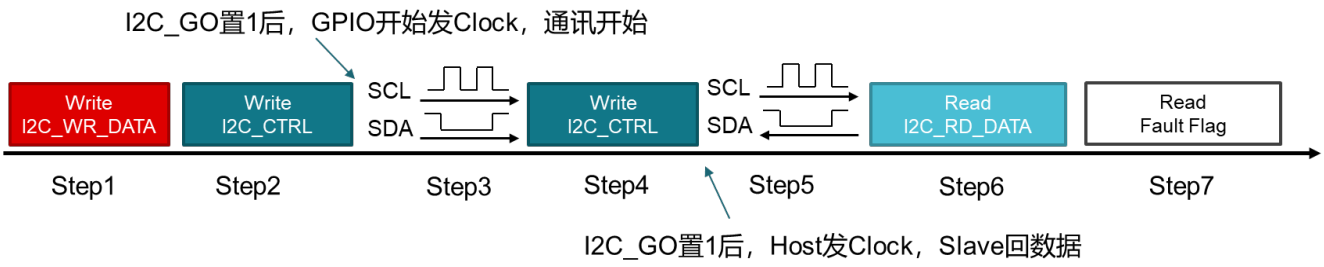

Figure 1-5. 7-Step Flowchart of I2C EEPROM Read Communication

2 I2C Hardware Overview for AFE and BJB

GPIO0 (SCL) and GPIO1 (SDA) of the AFE are used for I2C communication, while GPIO9 (SCL) and GPIO10 (SDA) of the BQ79731 are used for I2C communication. These two pins default to other functions and require writing CONTROL2[I2C_EN] =1 to enable the I2C function. SDA and SCL are open-drain outputs; smaller pullup resistors result in faster high-level rise time and higher communication frequency. The drawback of smaller pullup resistors is higher power consumption, requiring users to make a tradeoff between speed and power consumption. When I2C is idle, the GPIOs remain high, resulting in zero power consumption. The maximum I2C communication frequency for the AFE and BJB is 400kHz, and the minimum allowable value for the pullup resistor RPULLUP is 2kohm.

For pullup resistor calculation methods, refer to the application report *I2C Bus Pullup Resistor Calculation*.

Debugging I2C functions using the BQ79718 EVM and TI host GUI requires the following modifications:

Solder U14 EEPROM, verifying whether the pinout matches the schematic.

Solder U14 power supply capacitor C37.

Solder R94 and R184 0R jumpers.

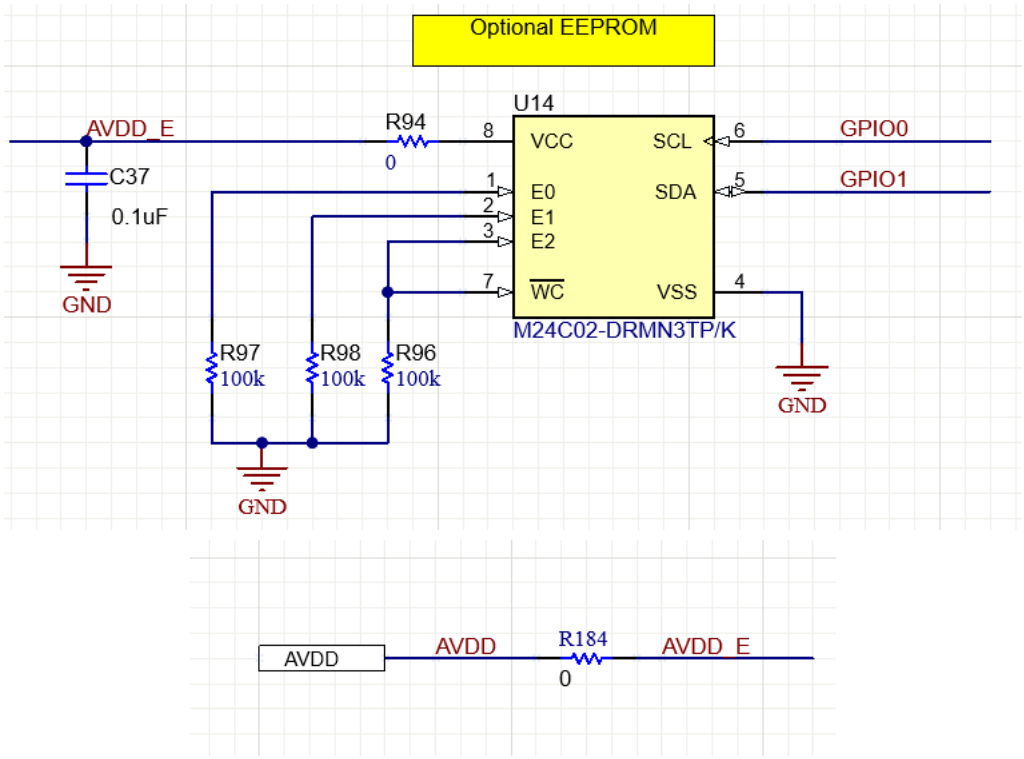

Figure 2-1. EEPROM Section Schematic of the BQ79718 EVM

Fly-wire GPIO0/1 with 2K pullup resistors connected to AVDD. No footprints for pullup resistors are reserved on the EVM.

Remove jumpers R62 and R64, and solder jumpers R61 and R63.

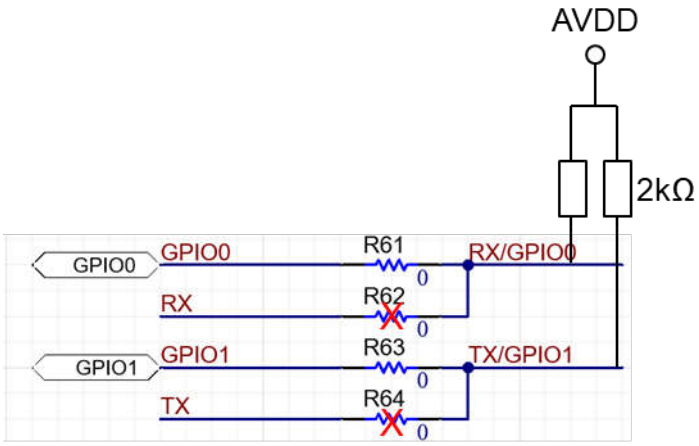

Figure 2-2. BQ79718 EVM EEPROM Schematic

3 Hands-On Practice with I2C Communication Commands

3.1 EEPROM Write Operation

Case 1: Write 0x00, 0x01, 0x02, 0x03, 0x04, and 0x05 into the address range 0x0000~0x0004 of the EEPROM with device ID 000.

The write command consists of three parts: Device ID, Register Start Address, and Data to be written.

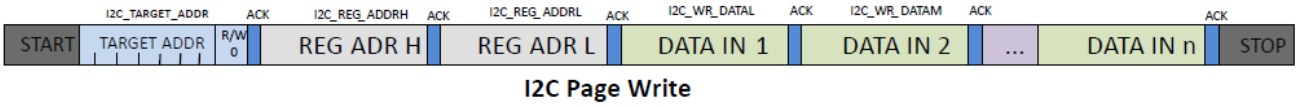

Figure 3-1. EEPROM Write Command Protocol

Device ID: TARGET ADDR

According to address definitions in EEPROM datasheets, the vast majority of EEPROM address definitions are as follows:

The device type is fixed at 0b1010. The device address is determined by the pin logic states of A2~A0. Bit0 is 0 for write operations and 1 for read operations.

Table 3-1. EEPROM Device ID Definition

Device Type Identifier				Device Address Bits			R/$\overline{W}$
Bit 7	**Bit 6**	**Bit 5**	**Bit 4**	**Bit 3**	**Bit 2**	**Bit 1**	**Bit 0**
1	0	1	0	A2	A1	A0	R/$\overline{W}$

Take the following figure as an example:

For write operations, ROM0 address is 0xA0 (0b1010 000 0), and ROM1 address is 0xA2 (0b1010 001 0)

For read operations, ROM0 address is 0xA1 (0b1010 000 1), and ROM1 address is 0xA3 (0b1010 001 1)

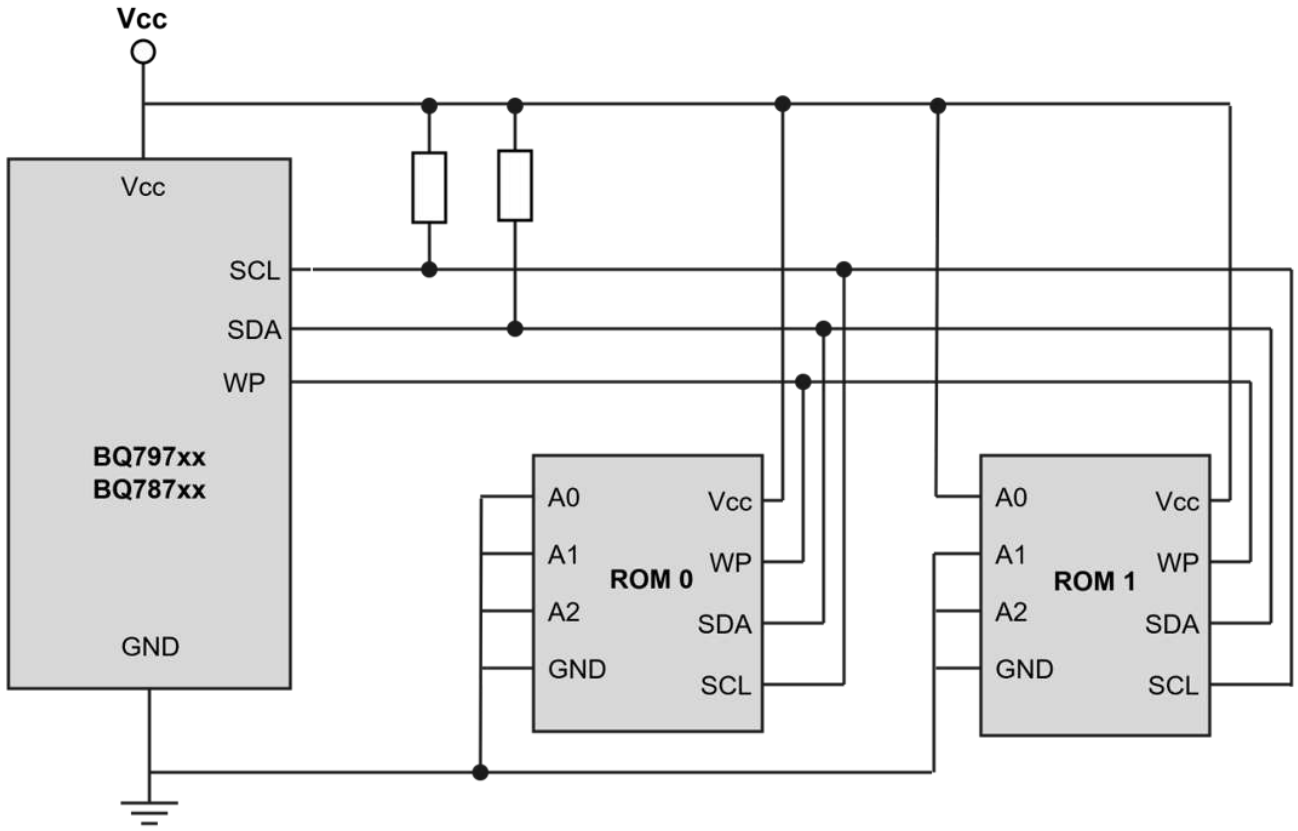

Figure 3-2. EEPROM Address Diagram

Register start address REG ADR H/L:

Writing can start from any address, **but consecutively written bytes cannot exceed the length of one Page**. After knowing the start address, the EEPROM automatically increments the address by 1 for each byte written before writing the next byte. However, the address does not accumulate across pages; otherwise, an address roll-over issue will occur. If the number of written bytes exceeds the remaining space of the current page, the address counter may automatically "roll over" to the beginning of that page, accidentally overwriting previously written data.

Page sizes vary across different ICs; searching for "byte page" in the datasheet reveals the page size. For example, with a 32byte-page, starting from the page start location 0x00 allows writing up to 32bytes consecutively. If writing starts from the page end location 0x1F, only one byte can be written.

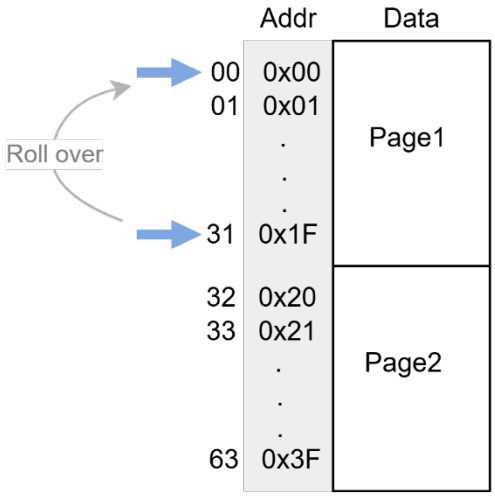

Figure 3-3. Register address automatically increments to the bottom of the page and automatically rolls over to the page start location

When writing large amounts of data across page boundaries, the data must be split into multiple write commands. **An interval of >5ms is required between write commands**, as specified in the EEPROM datasheet: Self-Timed Write Cycle within 5ms Maximum. This time is dedicated to a complex internal routine involving EEPROM erasure, charge pump boosting, data writing, and verification.

Below are the commands for writing 6 consecutive bytes

```
# 使能 I2C，将 GPIO0 和 GPIO1 设置为 SCL 和 SDA.
B0 03 0A 05 10 47
# I2C_WR_DATA 写设备 ID 0xA0 。I2C_CTRL 写 0x29，SEND-START-I2C_GO
B1 03 70 A0 29 E0 5B
# ADDH=00.  SEND-I2C_GO
B1 03 70 00 21 99 9D
# ADDL=00. SEND-I2C_GO
B1 03 70 00 21 99 9D
#写 1~5 个字节
B1 03 70 00 21 99 9D
B1 03 70 01 21 98 0D
B1 03 70 02 21 98 FD
B1 03 70 03 21 99 6D
B1 03 70 04 21 9B 5D
# 写最后一个字节， SEND-STOP-I2C_GO
B1 03 70 05 25 9B 0E
# 读故障寄存器，检查 I2C_LOW 和 I2C_NACK 是否有故障。
A0 05 3D 00 22 B5
```

Inputting the above commands into the TI EVM host GUI executes an I2C communication; the figure below shows the communication data recorded by a logic analyzer.

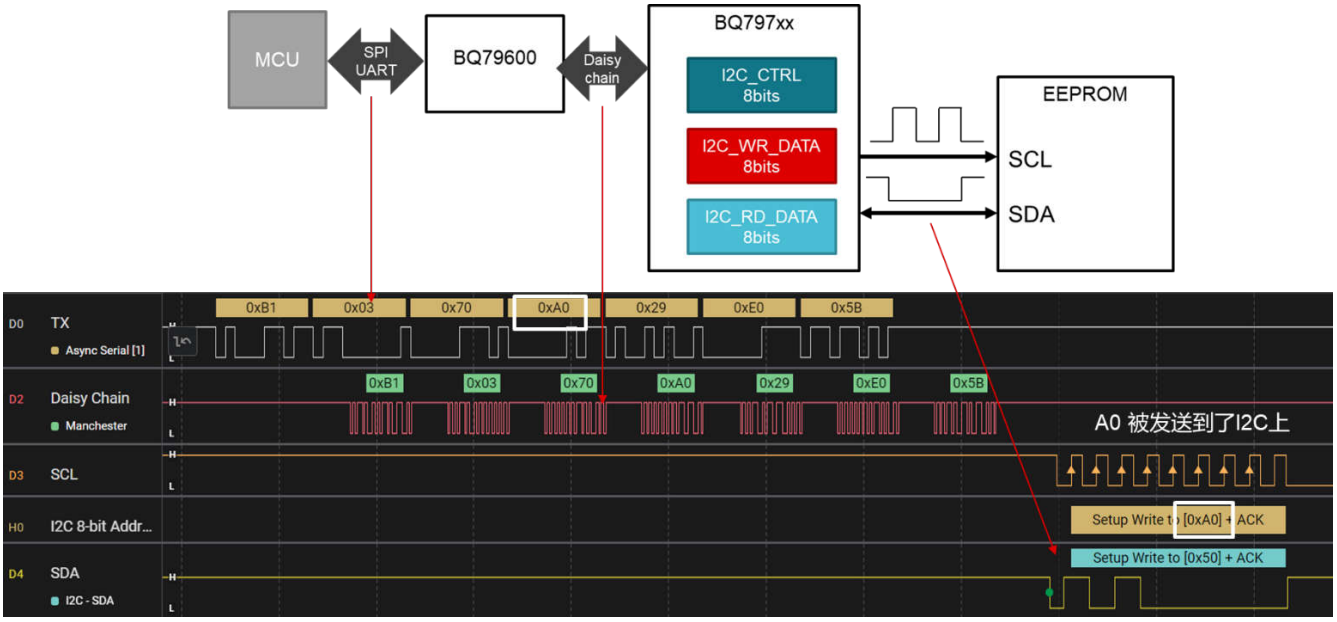

Figure 3-4. Logic Analyzer Data Analysis for I2C Transmission of A0

3.2 EEPROM Read Operation

3.2.1 Reading One Byte from Any Register Address

Case 2: Read data from address 0x0000 of the EEPROM with device ID 000

Write command workflow: First configure device ID 0xA0 and register address 0x0000, then write device ID 0xA1 and read data.

Figure 3-5. Protocol Definition for Reading a Single Byte

Below is the command for reading a single byte

```
# 使能 I2C，将 GPIO0 和 GPIO1 设置为 SCL 和 SDA.
B0 03 0A 05 10 47
# I2C_WR_DATA 写设备 ID 0xA0 。I2C_CTRL 写 0x29，SEND-START-I2C_GO
B1 03 70 A0 29 E0 5B
# ADDH=00.  SEND-I2C_GO
B1 03 70 00 21 99 9D
# ADDL=00. SEND-I2C_GO
B1 03 70 00 21 99 9D
#设备 ID A1; SEND-START-GO
B1 03 70 A1 29 E1 CB
# RECEIVE-STOP-NACK-I2C_GO
B0 03 71 17 B2 BA
# I2C 通讯已经结束，读取 I2C 传出来的数据。
A0 06 10 00 CF E5
# 读故障寄存器，检查 I2C_LOW 和 I2C_NACK 是否有故障。
A0 05 3D 00 22 B5
```

The figure below shows the GUI operation result, where the read result for register 0x610 is 0x00. 0x00 is the value written in Case 1, whereas the default stored value in EEPROM is 0xFF, indicating that both the write operation in Case 1 and the read operation in Case 2 were successful.

The 0x53D result only reported a DRST fault, indicating that the IC underwent a reset; this is a fault that will inevitably occur after device wakeup if not cleared during initialization. Therefore, this I2C communication succeeded without any anomalies.

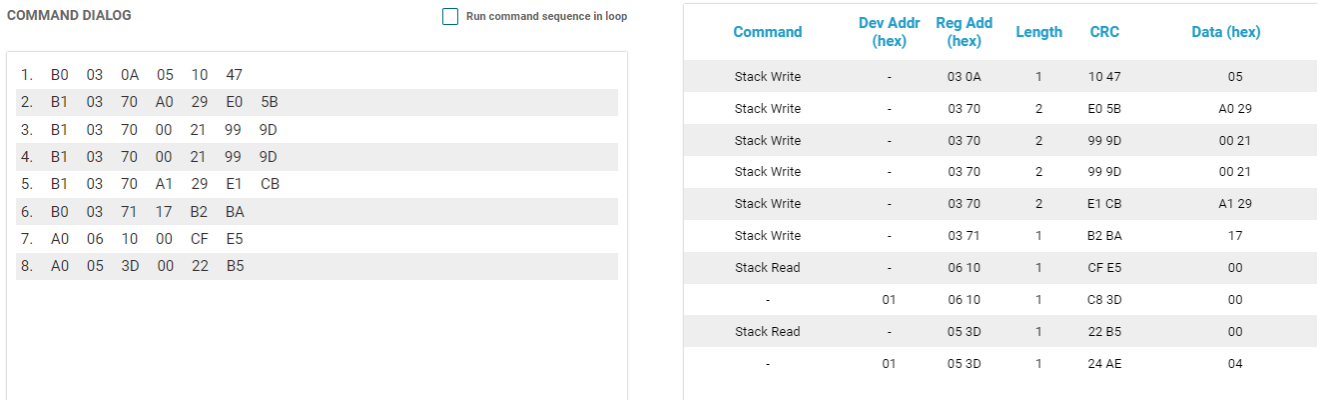

Figure 3-6. The left side is the command editing area of the TI host GUI, and the right side displays the actually transmitted and received data

3.2.2 Continuously Reading Multiple Bytes from Any Address

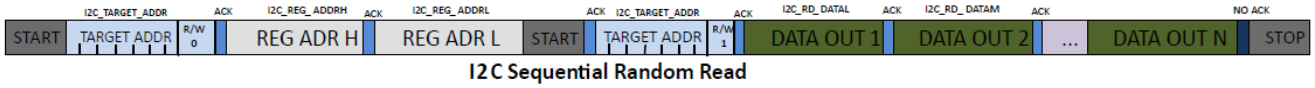

Figure 3-7. Multi-Byte Read Protocol

The difference between multi-byte read and single-byte read is that after reading one byte, no Stop condition is set; reading continues to the next byte until the last byte is read, at which point Stop is set. Below is the example code for writing 5 consecutive bytes and reading 16 consecutive bytes.

```
# I2C Enable
B0 03 0A 05 10 47
######### 写操作指定 ID 和寄存器起始地址#########
# 写设备 ID A0
B1 03 70 A0 29 E0 5B
# Send, ADDH=00, ADDL=00, I2C GO
B1 03 70 00 21 99 9D
B1 03 70 00 21 99 9D
######### 连续写 5 个字节 #########
B1 03 70 00 21 99 9D
B1 03 70 01 21 98 0D
B1 03 70 02 21 98 FD
B1 03 70 03 21 99 6D
B1 03 70 04 21 9B 5D
# Write Data SEND-STOP-I2C GO
B1 03 70 05 25 9B 0E
# Read Fault
A0 05 3D 00 22 B5
######### 读操作指定 ID 和寄存器起始地址##########
# 设备 ID A0; SEND-START-GO
B1 03 70 A0 29 E0 5B
# Send, ADDH=00, ADDL=00, SEND-GO
B1 03 70 00 21 99 9D
B1 03 70 00 21 99 9D
## 读设备 ID A1; SEND-START-GO
B1 03 70 A1 29 E1 CB
######### 开始读 16 个字节 RECEIVE-GO #########
B0 03 71 11 32 B8 A0 06 10 00 CF E5
B0 03 71 11 32 B8 A0 06 10 00 CF E5
B0 03 71 11 32 B8 A0 06 10 00 CF E5
B0 03 71 11 32 B8 A0 06 10 00 CF E5
B0 03 71 11 32 B8 A0 06 10 00 CF E5
B0 03 71 11 32 B8 A0 06 10 00 CF E5
B0 03 71 11 32 B8 A0 06 10 00 CF E5
B0 03 71 11 32 B8 A0 06 10 00 CF E5
```

```
B0 03 71 11 32 B8 A0 06 10 00 CF E5
B0 03 71 11 32 B8 A0 06 10 00 CF E5
B0 03 71 11 32 B8 A0 06 10 00 CF E5
B0 03 71 11 32 B8 A0 06 10 00 CF E5
B0 03 71 11 32 B8 A0 06 10 00 CF E5
B0 03 71 11 32 B8 A0 06 10 00 CF E5
B0 03 71 11 32 B8 A0 06 10 00 CF E5
B0 03 71 11 32 B8 A0 06 10 00 CF E5
## I2C 读最后一个字节
B0 03 71 17 B2 BA
A0 06 10 00 CF E5
A0 05 3D 00 22 B5
```

The code above includes a special treatment: combining two commands and sending them together.

Standard commands are as follows

B0 03 71 11 32 B8 # I2C Start Reading Data 1

A0 06 10 00 CF E5 # Daisy Chain Read Back Returned Data 1

B0 03 71 11 32 B8 # I2C Start Reading Data 2

A0 06 10 00 CF E5 # Daisy Chain Read Back Returned Data 2

Specially treated commands are as follows

B0 03 71 11 32 B8 A0 06 10 00 CF E5

B0 03 71 11 32 B8 A0 06 10 00 CF E5

The reason for this handling is that the host GUI command interval is too long, which, combined with the reset mechanism of the AFE, would cause the AFE to enter Shutdown Mode without special treatment.

After the execution of the blue Read Data 1 command, CLK/GPIO0 will remain low. CLK/GPIO0 will not be pulled high until the next green Read Data 2 command. If the GPIO0 low level duration exceeds 20ms, it will force the AFE into Shutdown Mode. All I2C and daisy chain communications are interrupted, requiring re-wakeup and re-initialization to function properly.

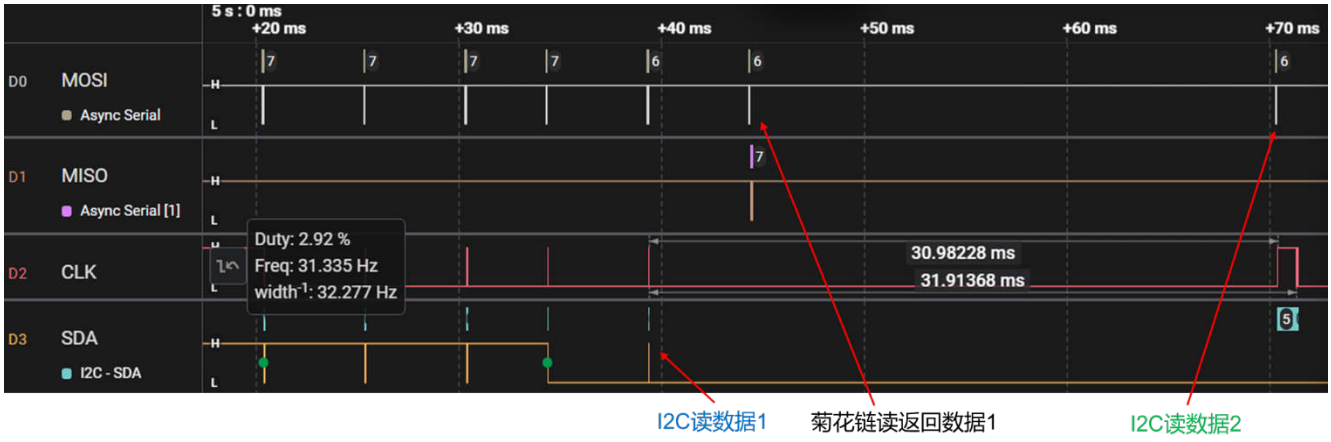

Figure 3-8. A 31ms Interval Between Two I2C Read Commands Causes the AFE to Falsely Identify a Hardware Reset

The multiplexed function of GPIO0 on the AFE is hardware resetting (Hardware Reset) the AFE. When the AFE uses UART to communicate with the MCU, GPIO0 is multiplexed as the RX pin to receive commands sent from the MCU. If GPIO0 is pulled low for more than 20ms, the AFE considers it a hardware reset command sent from the MCU, and enters Shutdown Mode after completing the reset process.

Note

The [I2C_EN] setting is ignored in the base device (device woken up by PING) and device UART communication continues to work even if [I2C_EN=1] in the base device. Stack devices (woken up by TONE) block UART comm clear/ Sleep2Act ping/WAKE ping ONLY if the I^2C feature is enabled ([I2C_EN] =1). When GPIO0 is used as I^2C SCL, SCL must not be low for more than 20 ms. Otherwise, it may mis-trigger SD_HW_RESET ping.

Figure 3-9. I2C Communication Considerations in the BQ79718 Datasheet

To avoid this issue, when the MCU reads I2C data continuously, the interval between two read commands must be less than 20ms, with <15ms recommended to leave more margin. As shown in the Note section of the figure above from the AFE datasheet, users are also reminded that the SCL low time cannot exceed 20ms; otherwise, it may be falsely recognized as a hardware reset signal.

When continuously reading data via GUI, the two commands must be sent combined together as in the special treatment example code above, so that the command interval between Read Data 1 and Data 2 is about 5ms, which will not be recognized as a hardware reset signal. The drawback is that the host GUI does not treat special commands as read commands, so the returned data will not be displayed in the LOG box on the right. A logic analyzer is required to decode the returned data on the I2C bus to verify whether the operation was successful.

4 Conclusion

This paper explains the key points and considerations of I2C communication. Host GUI command codes for EEPROM writing and reading are provided so that users can run examples on the TI host GUI to intuitively understand this communication process, allowing them to quickly implement this feature in their own software once they master the workflow.

5 References

1. Understanding the I2C Bus
2. I2C Bus Pullup Resistor Calculation

IMPORTANT NOTICE AND DISCLAIMER

TI PROVIDES TECHNICAL AND RELIABILITY DATA (INCLUDING DATASHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES “AS IS” AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS AND IMPLIED, INCLUDING WITHOUT LIMITATION ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT OF THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for skilled developers designing with TI products. You are solely responsible for (1) selecting the appropriate TI products for your application, (2) designing, validating and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, regulatory or other requirements.

These resources are subject to change without notice. TI grants you permission to use these resources only for development of an application that uses the TI products described in the resource. Other reproduction and display of these resources is prohibited. No license is granted to any other TI intellectual property right or to any third party intellectual property right. TI disclaims responsibility for, and you fully indemnify TI and its representatives against any claims, damages, costs, losses, and liabilities arising out of your use of these resources.

TI’s products are provided subject to TI’s Terms of Sale, TI’s General Quality Guidelines, or other applicable terms available either on ti.com or provided in conjunction with such TI products. TI’s provision of these resources does not expand or otherwise alter TI’s applicable warranties or warranty disclaimers for TI products. Unless TI explicitly designates a product as custom or customer-specified, TI products are standard, catalog, general purpose devices.

TI objects to and rejects any additional or different terms you may propose.

Last updated 10/2025