Subsystem Design

I2C Expander Through UART Bridge

1 Description

Figure 1-1 shows how to transfer data or commands from a universal asynchronous receiver-transmitter (UART) interface to several target I2C controllers using the MSPM0 as an I2C expander. Incoming UART packets are specifically formatted to facilitate the transition to I2C communication. Figure 1-1 also illustrates how errors can be communicated back to the host device. Code for this example is found in the MSPM0 SDK.

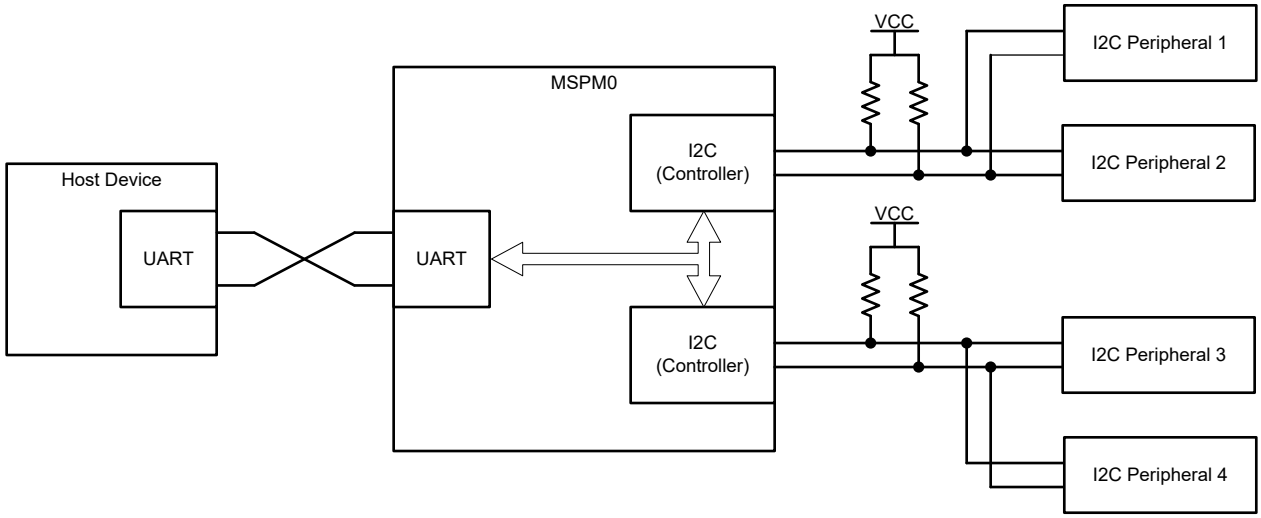

Figure 1-1. Subsystem Functional Block Diagram

2 Required Peripherals

This application requires a UART and I2C peripherals.

Table 2-1. Required Peripherals

Sub-block Functionality	Peripheral Use	Notes
UART TX-RX Interface	(1 ×) UART	Called UART_BRIDGE_INST in code
I2C Controllers	(2 ×) I2C	Called I2C_BRIDGE_INST and I2C_BRIDGE2_INST in code

3 Design Steps

1. Set UART peripheral instance, I2C peripheral instance, and pin out to desired device pins in SysConfig.
2. Set UART baud rate in SysConfig. Default is 9600baud.
3. Set I2C clock speed in SysConfig. Default is 100kHz.
4. Define the maximum I2C packet size the bridge handles.
5. Define key UART header values (optional).
6. Customize error handling (optional).

4 Design Considerations

- **Communication Speeds**: Increasing speeds increases data throughput and decreases chances of collision. Adjusting the external pullup resistors according to I2C specifications is necessary to allow for communication if I2C speeds are increased. Optimizations include higher device operating speeds, multiple transfer buffers, header size reduction, or state machine simplification.

- **UART Header**: The UART packer header and start byte are customizable for the application. Texas Instruments recommends assigning values that are less likely to occur during the start of typical data transfers.
- **Error Handling**: Correspond the error values to ASCII numerical values if monitoring UART bus with a computer terminal. Make sure the host UART device can read error values and know the associated meanings so appropriate action can be taken by the host. Add additional error types by modifying the ErrorFlags structure type and add additional error detection code within the Uart_Bridge(). The current implementation detects limited errors and reports back the corresponding code on the UART interface. The application code then breaks from the current communication state machine. Users can add additional error handling code to change the behavior of the bridge when an error occurs. For example, re-sending an I2C packet after a NACK occurs.

5 Software Flow Charts

Figure 5-1, Figure 5-2, and Figure 5-3 show the code flow diagrams for Main UART Bridge functionality, Main() plus UART ISR, and I2C ISR, respectively, for Figure 1-1.

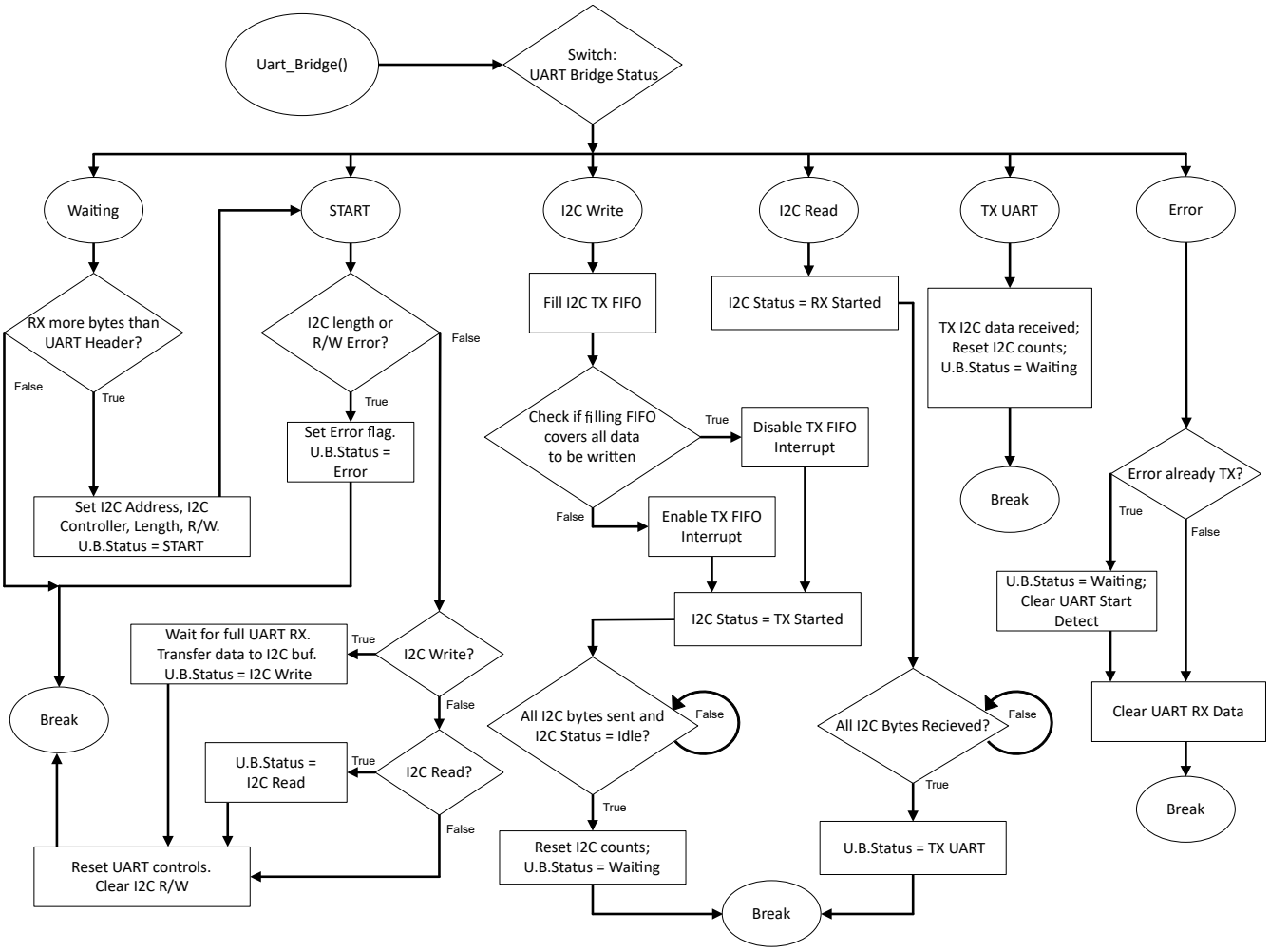

Figure 5-1. Software Flow Diagram for UART_Bridge()

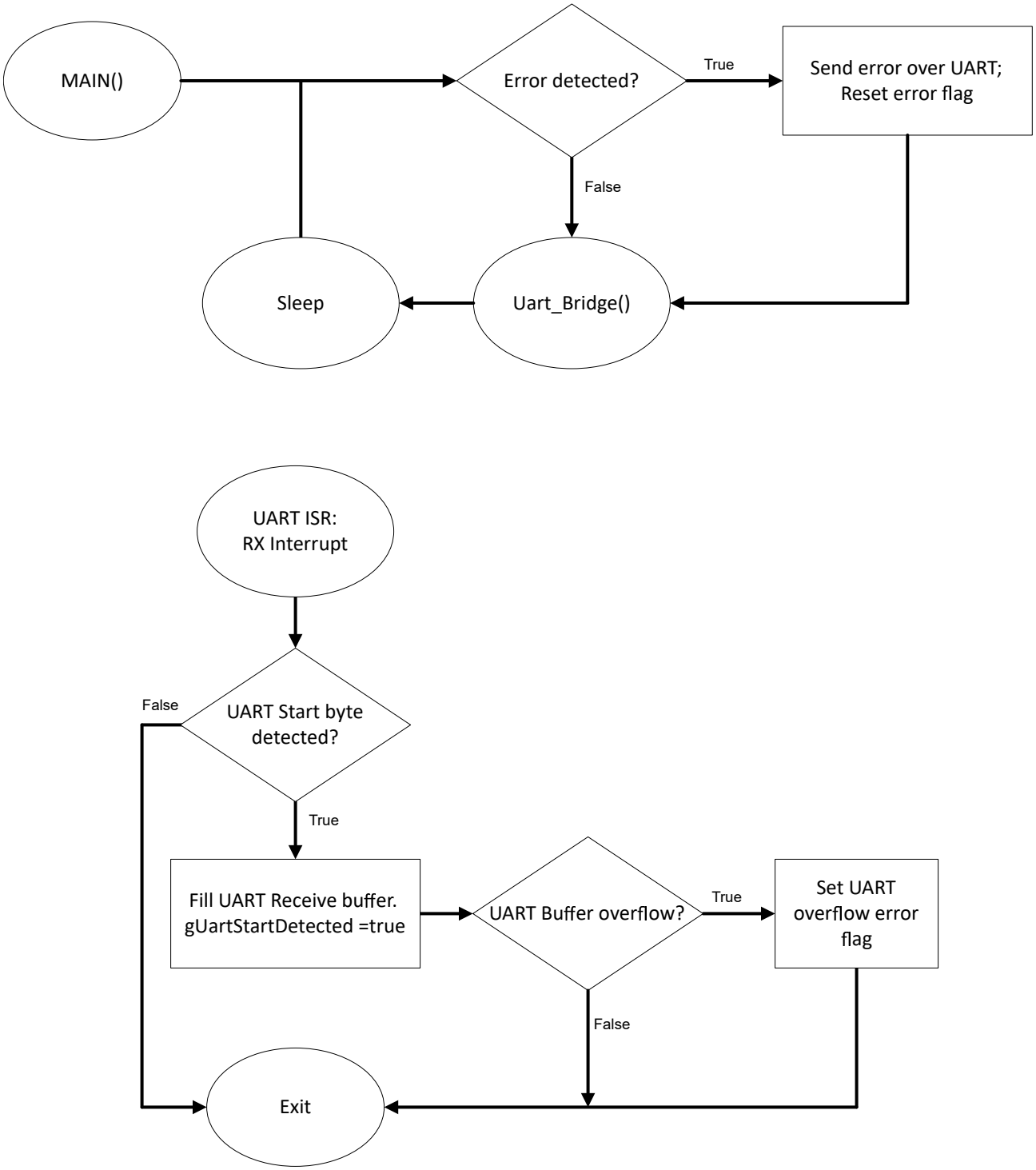

Figure 5-2. Software Flow Diagrams for MAIN Loop and UART ISR

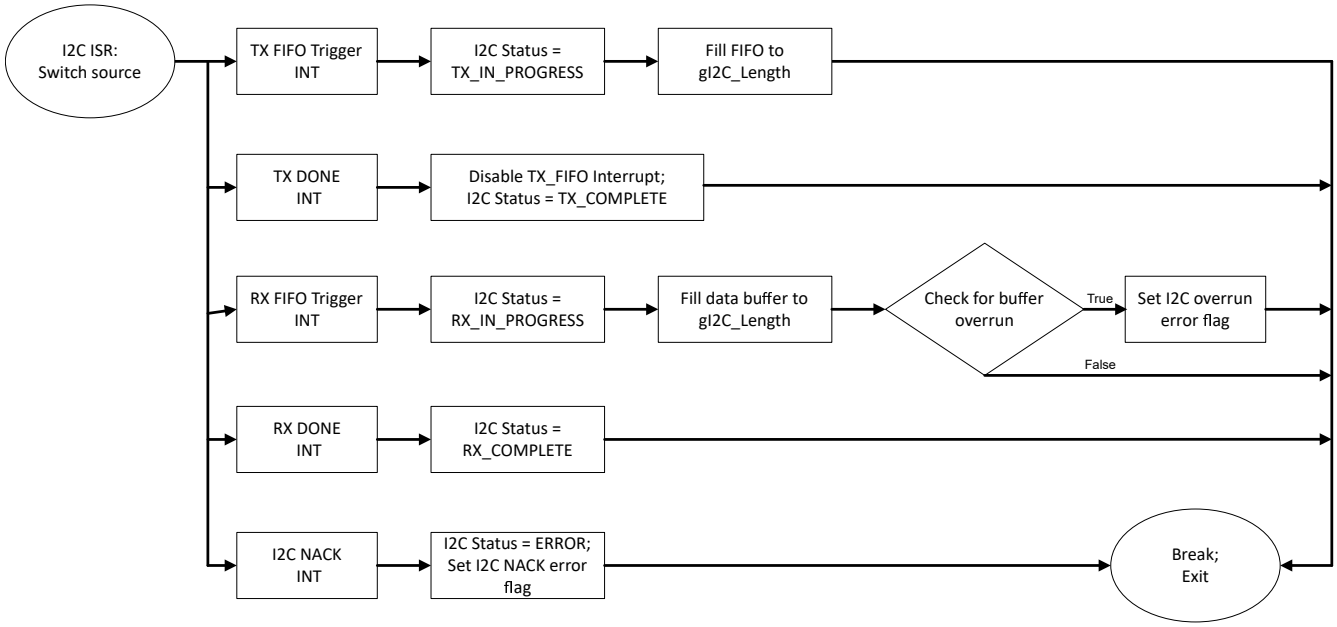

Figure 5-3. Software Flow Diagram for I2C ISR

6 Required UART Packet

Figure 6-1 shows the required UART packet for proper bridging to the I2C interface. The values shown are the default header values defined within Figure 1-1.

- *Start Byte*: The value used by the bridge to indicate a new transaction is starting. UART transmissions are ignored until this value is acknowledged by the bridge.
- *I2C Address*: The address of the I2C target the host communicates with.
- *I2C Read or Write Indicator*: The value that functions the bridge to read or write from the target I2C device.
- *Message Length N*: The length of data transferred in bytes. This value cannot be larger than the defined I2C maximum packet length.
- *Bridge Index*: The I2C controller that the host communicates on.
- *D0, D1...., Dn*: The data transferred within the bridge.

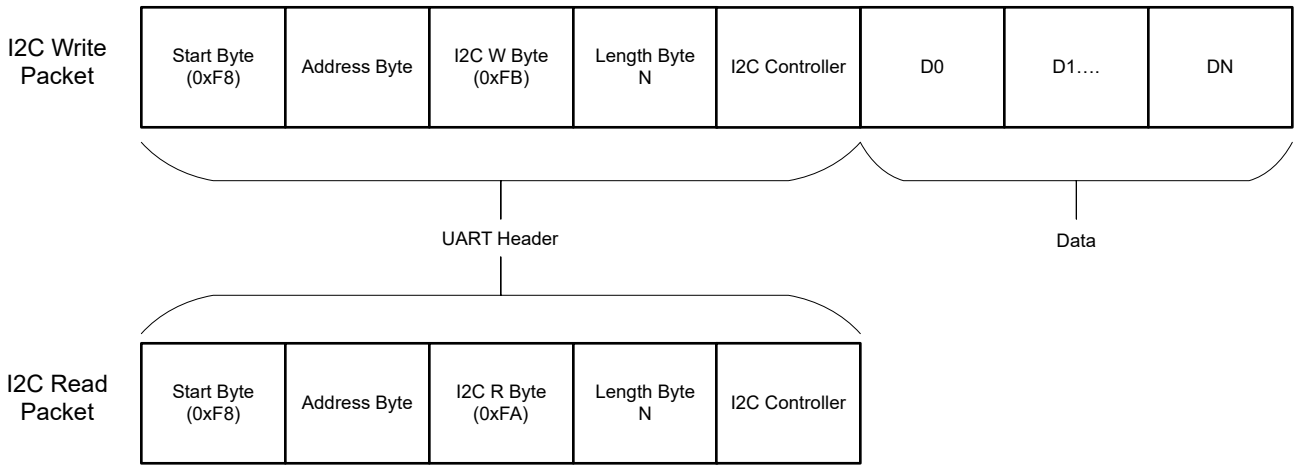

Figure 6-1. UART Bridge Packet Description

7 Device Configuration

This application makes use of TI System Configuration Tool (SysConfig) graphical interface to generate the configuration code for the COMP and two TIMER modules. Using a graphical interface to configure the device peripherals streamlines the application prototyping process.

8 Application Code

To change the specific values used by the UART packet or the maximum I2C packet size, modify the following #defines in the beginning of the code example, as demonstrated in the following code block:

```
/* Define UART Header and Start Byte*/
#define UART_HEADER_LENGTH 0x04
#define UART_START_BYTE 0xF8
#define UART_READ_I2C_BYTE 0xFA
#define UART_WRITE_I2C_BYTE 0xFB
#define ADDRESS_INDEX   0x00
#define RW_INDEX        0x01
#define LENGTH_INDEX    0x02
#define BRIDGE_INDEX    0x03

/*Define max packet sizes*/
#define I2C_MAX_PACKET_SIZE 16
#define UART_MAX_PACKET_SIZE (I2C_MAX_PACKET_SIZE + UART_HEADER_LENGTH)
```

9 Additional Resources

- Texas Instruments, I2C Expander Sub-System Code
- Texas Instruments, Download the MSPM0 SDK
- Texas Instruments, Learn more about SysConfig
- Texas Instruments, MSPM0L LaunchPad™
- Texas Instruments, MSPM0G LaunchPad™
- Texas Instruments, MSPM0 UART Academy
- Texas Instruments, MSPM0 I2C Academy

10 E2E

See TI's E2E™ support forums to view discussions and post new threads to get technical support for utilizing MSPM0 devices in designs.

11 Revision History

NOTE: Page numbers for previous revisions may differ from page numbers in the current version.

Changes from Revision * (September 2024) to Revision A (August 2025)	Page
• Removed compatible devices section	1

12 Trademarks

LaunchPad™ and E2E™ are trademarks of Texas Instruments.
All trademarks are the property of their respective owners.

IMPORTANT NOTICE AND DISCLAIMER

TI PROVIDES TECHNICAL AND RELIABILITY DATA (INCLUDING DATA SHEETS), 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 will 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 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.

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

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265