

# Subsystem Design

## UART to I2C Bridge



### Description

Figure 1-1 demonstrates how to transfer data or commands from a UART interface to several target I2C peripherals using the MSPM0 as an I2C controller. Incoming UART packets are specifically formatted to facilitate the transition to I2C communication. Figure 1-1 can communicate errors in communication back to the host device. Code for this example is found at [UART to I2C Bridge Sub-System Code](#).

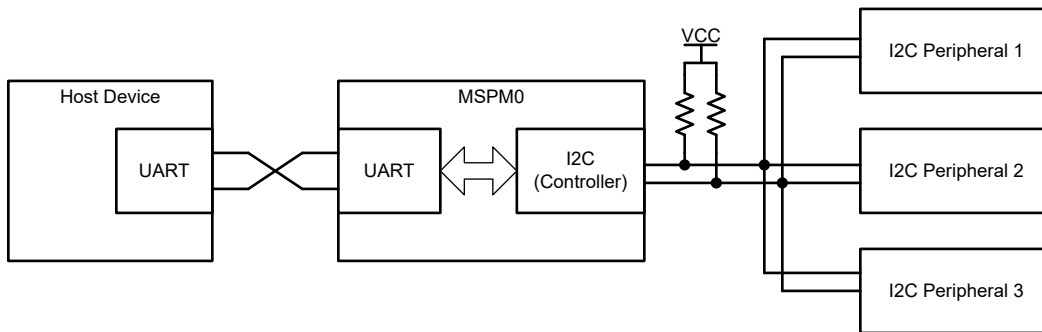


Figure 1-1. Sub-System Functional Block Diagram

### Requirements

Applying this application requires a UART and I2C peripheral.

Table 1-1. Required Peripherals

Sub-Block Functionality	Peripheral Use	Notes
UART TX/RX Interface	UART	Called UART_Bridge_INST in code. Default 9600 baud rate.
I2C Controller	I2C	Called I2C_Bridge_INST in code. Default 100 kHz transmission rate.

## 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 9600 baud.
3. Set I2C clock speed in [Sysconfig](#). Default is 100 kHz.
4. Define the max-I2C packet size the bridge handles.
5. Define key UART header values (optional).
6. Customize error handling (optional).

## Design Considerations

1. Communication speed.
  - a. Increasing both interface speeds increases data throughput and decreases chances of data collisions.
  - b. Adjusting external pull-up resistors according to I2C specifications is necessary to allow for communication if I2C speeds are increased.
  - c. Repeated, large data packets at higher speeds can impact overall system performance. Additional optimization of this code can be necessary to meet increased bridge utilization. Additional optimizations include higher device operating speeds, multiple transfer buffers, header size reduction or state machine simplification.

---

### Note

[Figure 1-1](#) example was only tested with default speeds of 9600 baud (UART) and 100 kHz (I2C) speeds.

---

2. UART header.
  - a. The UART packet header and start byte values are customizable for your application. Texas Instruments recommends assigning values that are less likely to occur during the start of typical data transfers.
3. Error handling.
  - a. Correspond the error values to ASCII numerical values if monitoring UART bus with a computer terminal.
  - b. Make sure the host UART device can read error values and know the associated meanings so appropriate action can be taken by the host.
  - c. Add additional error types by modifying the *ErrorFlags* structure type and add additional error detection code within the *Uart\_Bridge()*.
  - d. 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.

---

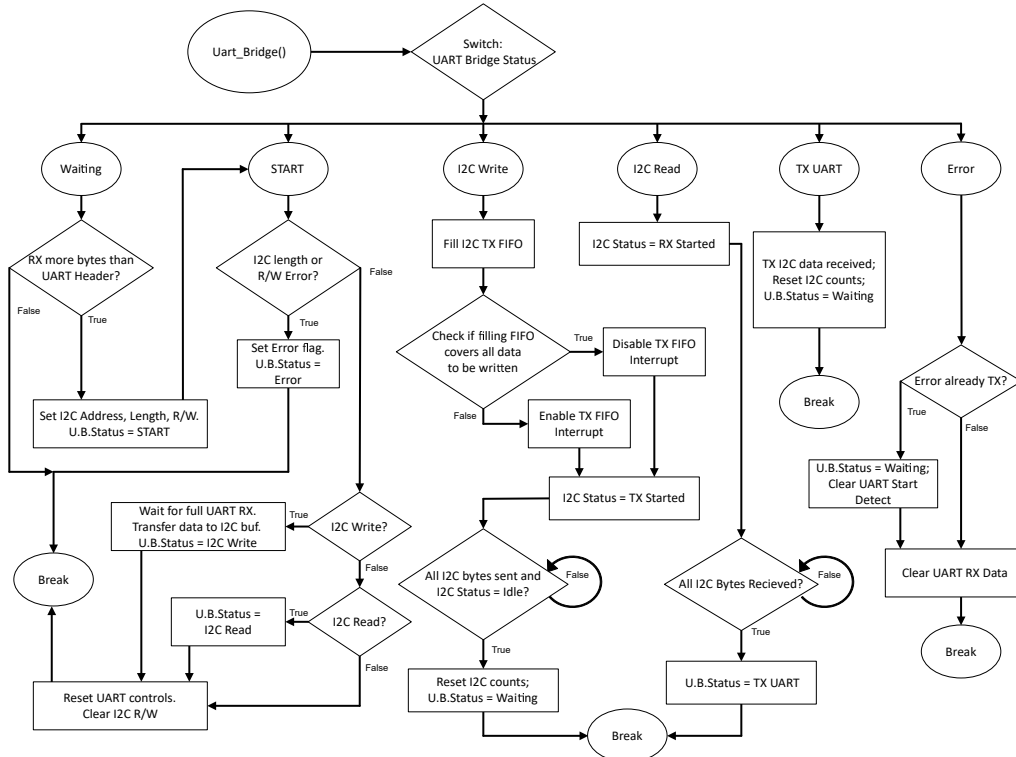
### Note

[Figure 1-1](#) currently flags common errors and assigns them numerical values, as defined in the *ErrorFlags* structure type.

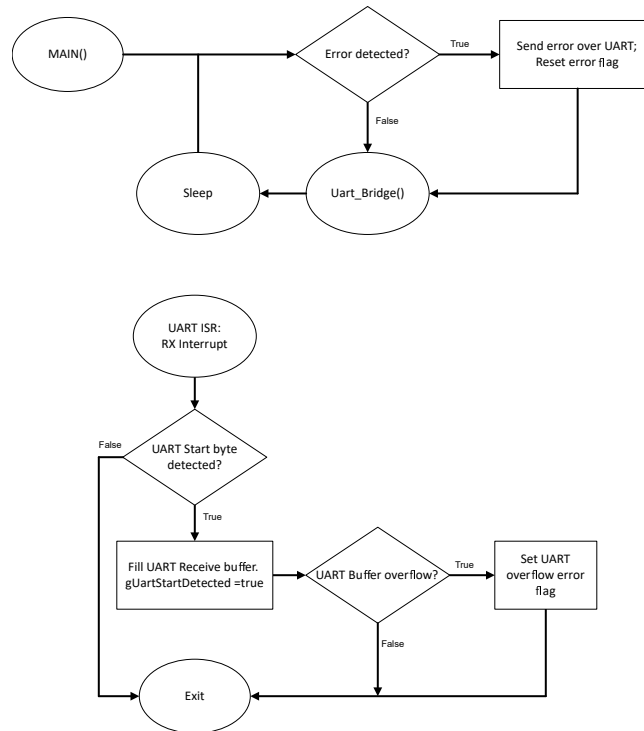
---

**Software Flowcharts**

Figure 1-2, Figure 1-3, and Figure 1-4 show the code flow diagrams for *Main UART Bridge functionality*, *Main() plus UART ISR*, and *I2C ISR*, respectively, for Figure 1-1.



**Figure 1-2. Software Flow Diagram for UART\_Bridge()**



**Figure 1-3. Software Flow Diagrams for MAIN Loop and UART ISR**

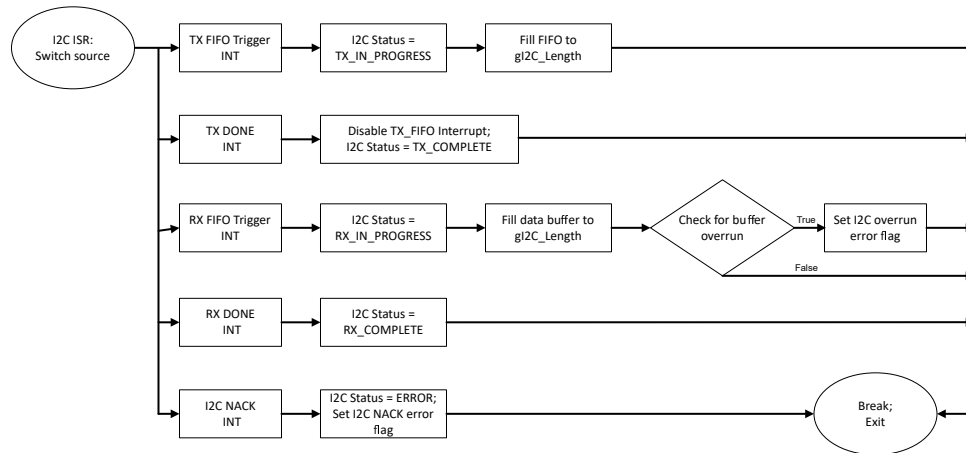


Figure 1-4. Software Flow Diagram for I2C ISR

### Required UART Packet

Figure 1-5 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 max packet length.
- **D0, D1...., Dn:** The data transferred within the bridge.

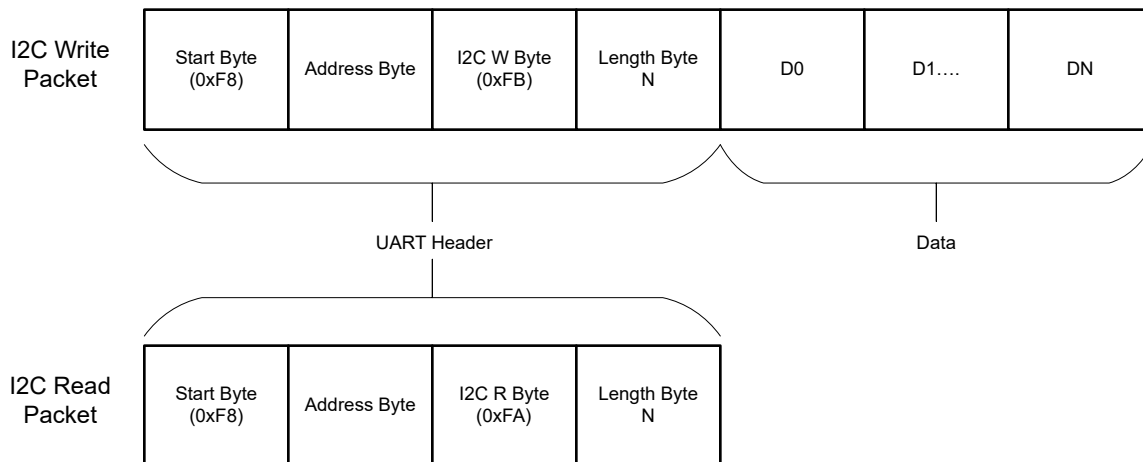


Figure 1-5. UART Bridge Packet Description

## Device Configuration

Figure 1-1 application uses the [TI System Configuration Tool \(SysConfig\)](#) graphical interface to generate the configuration code of the device peripherals. Using a graphical interface to configure the device peripherals streamlines the application prototyping process.

## Application Code

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

```
/* Define UART Header and Start Byte*/
#define UART_HEADER_LENGTH 0x03
#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 max packet sizes*/
#define I2C_MAX_PACKET_SIZE 16
#define UART_MAX_PACKET_SIZE (I2C_MAX_PACKET_SIZE + UART_HEADER_LENGTH)
```

Several points in the code are comments around error detection. A user can add custom error handling and additional error reporting at these points in the code. For brevity, not all error handling code intersections are included here. In practice, search for comments in the code similar to what is demonstrated in the following code block:

```
while (DL_I2C_isControllerRXFIFOEmpty(I2C_BRIDGE_INST) != true) {
    if (gI2C_Count < gI2C_Length) {
        gI2C_Data[gI2C_Count++] =
            DL_I2C_receiveControllerData(I2C_BRIDGE_INST);
    } else {
        /*
         * Ignore and remove from FIFO if the buffer is full
         * Optionally add error flag update
         */
        DL_I2C_receiveControllerData(I2C_BRIDGE_INST);
        gError = ERROR_I2C_OVERUN;
    }
}
```

## Additional Resources

- Texas Instruments, [UART to I2C Bridge Sub-System Code](#)
- Texas Instruments, [Learn More About TI Sysconfig](#), tool
- Texas Instruments, [MSPM0 Support Development Kit](#), tool
- Texas Instruments, [MSPM0 Academy: UART](#), training
- Texas Instruments, [MSPM0 Academy: I2C](#), training

## 1 Revision History

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

Changes from Revision * (December 2023) to Revision A (August 2025)	Page
• Removed compatible devices section.....	1

## Trademarks

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](https://www.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

Copyright © 2025, Texas Instruments Incorporated