Subsystem Design

UART to I2C Bridge

Jace Hall

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.

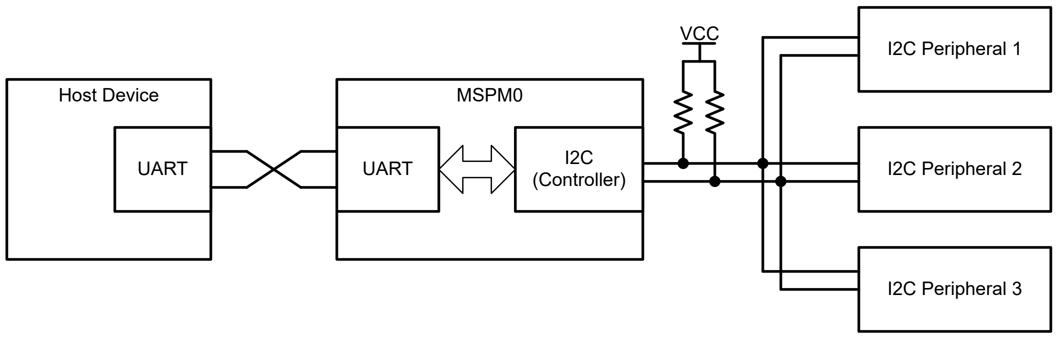

Figure 1-1. Sub-System Functional Block Diagram

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.

Compatible Devices

Compatible devices are listed in Table 1-2 with corresponding EVMs based on the requirements in Table 1-1. Using other MSPM0 devices and corresponding EVMs is possible if the requirements in Table 1-1 are met.

Table 1-2. Compatible Devices

Compatible Devices	EVM
MSPM0Lxxxx	LP-MSPM0L1306
MSPM0Gxxxx	LP-MSPM0G3507

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.

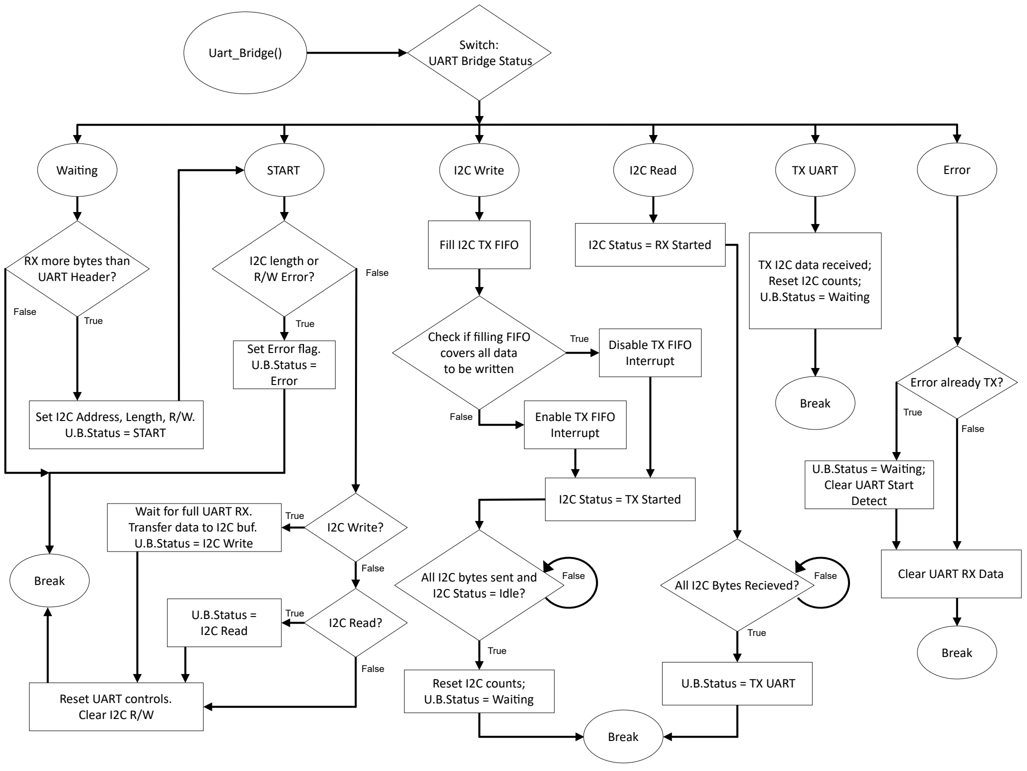
Uart_Bridge()
Switch: UART Bridge Status
Waiting
START
I2C Write
I2C Read
TX UART
Error
RX more bytes than UART Header?
False
True
Set I2C Address, Length, R/W. U.B.Status = START
I2C length or R/W Error?
False
True
Set Error flag. U.B.Status = Error
Fill I2C TX FIFO
Check if filling FIFO covers all data to be written
True
Disable TX FIFO Interrupt
False
Enable TX FIFO Interrupt
I2C Status = TX Started
I2C Status = RX Started
TX I2C data received; Reset I2C counts; U.B.Status = Waiting
Break
Error already TX?
True
False
U.B.Status = Waiting; Clear UART Start Detect
Clear UART RX Data
Break
Break
Wait for full UART RX. Transfer data to I2C buf. U.B.Status = I2C Write
True
I2C Write?
False
U.B.Status = I2C Read
True
I2C Read?
False
Reset UART controls. Clear I2C R/W
All I2C bytes sent and I2C Status = Idle?
False
True
Reset I2C counts; U.B.Status = Waiting
All I2C Bytes Recieved?
False
True
U.B.Status = TX UART
Break

Figure 1-2. Software Flow Diagram for UART_Bridge()

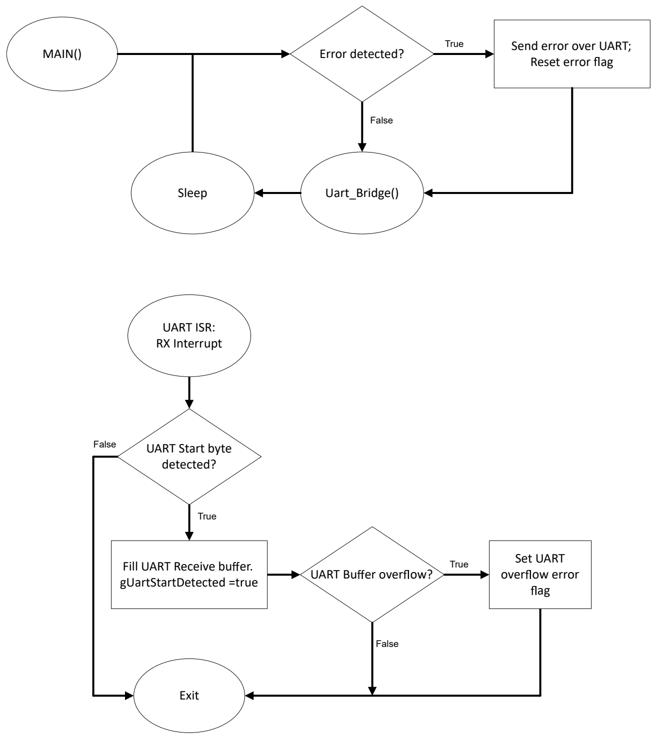
MAIN()
Error detected?
True
Send error over UART; Reset error flag
False
Sleep
Uart_Bridge()
UART ISR: RX Interrupt
UART Start byte detected?
False
True
Fill UART Receive buffer. gUartStartDetected =true
UART Buffer overflow?
True
Set UART overflow error flag
False
Exit

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

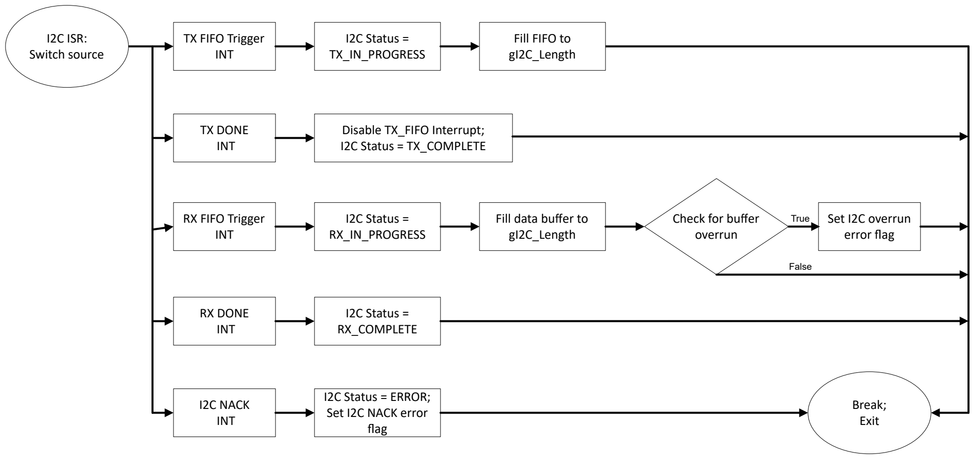
I2C ISR:
Switch source
TX FIFO Trigger INT
I2C Status = TX_IN_PROGRESS
Fill FIFO to gI2C_Length
TX DONE INT
Disable TX_FIFO Interrupt; I2C Status = TX_COMPLETE
RX FIFO Trigger INT
I2C Status = RX_IN_PROGRESS
Fill data buffer to gI2C_Length
Check for buffer overrun
True
Set I2C overrun error flag
False
RX DONE INT
I2C Status = RX_COMPLETE
I2C NACK INT
I2C Status = ERROR; Set I2C NACK error flag
Break; Exit

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.

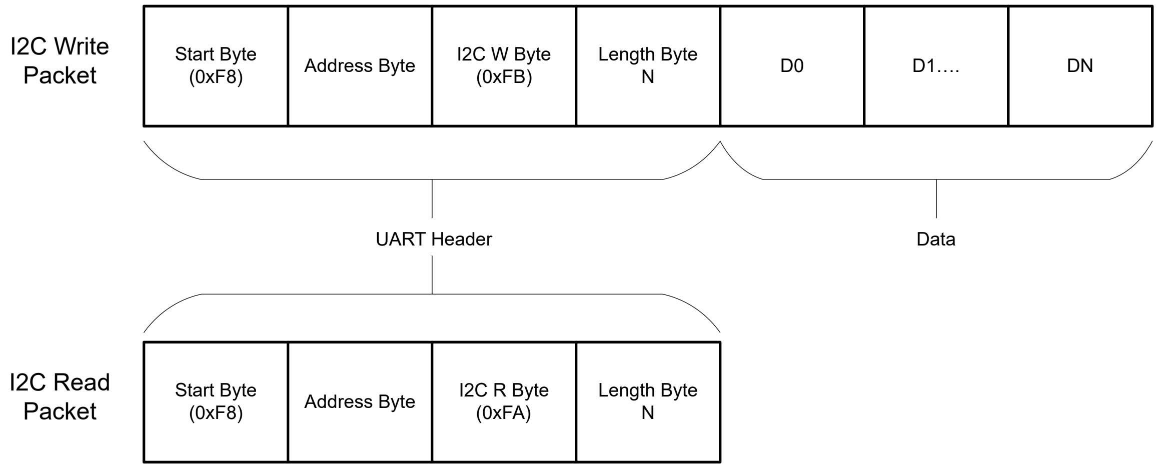

Figure 1-5. UART Bridge Packet Description

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