Subsystem Design

MSPM0 Cookbook UART to SPI

1 Description

This subsystem demonstrates how to implement the MSPM0 device as a UART to SPI bridge. Incoming UART packets are expected to be in a specific format in order to facilitate SPI communication. This example also has the ability to determine error conditions and communicate them back to the UART device. The code for this example can be found in the MSPM0 SDK.

2 Required Peripherals

Peripheral Used	Notes
UART	Called UART_BRIDGE_INST in code
SPI	Called SPI_0_INST in code

3 Compatible Devices

Based on the requirements in required peripherals, this example is compatible with the devices shown in the below table. Generally, any device with the capabilities listed in the required peripherals table can support this example.

Compatible Devices	EVM
MSPM0Lxxxx	LP-MSPM0L1306
MSPM0Gxxxx	LP-MSPM0G3507

4 Design Steps

1. Set up the SPI module in Sysconfig. Put the device in controller mode, and leave the rest of the settings on default. In the Advanced Configuration tab, make sure that the RX FIFO Threshold level is set to "RX FIFO contains ≥1 entry. Make sure that the TX FIFO Threshold level is set to "TX FIFO contains ≤ 2 entries." Now navigate to the Interrupt configuration tab, and enable the Receive, Transmit, RX Timeout, Parity Error, Receive FIFO Overflow, Receive FIFO Full, and Transmit FIFO Underflow interrupts.
2. Set up the UART module in Sysconfig. Set the baud rate to 9600. Enable the Receive interrupt.

5 Design Considerations

1. In the application code, make sure you checked the SPI and UART maximum packet sizes against the requirements of your application.
2. To increase the UART baud rate, adjust the value in the SysConfig UART tab labeled *Target Baud Rate*. Below this, observe the calculated baud rate change to reflect the target baud rate. This is calculated using the available clocks and dividers.
3. Check error flags and handle them appropriately. The UART and I^2C peripherals are both capable of throwing informative error interrupts. For easy debugging, this subsystem uses an enum and a global variable to save error codes when error codes are thrown. In real-world applications, handle errors in the code so the errors do not break down the project.
4. The current form of the project defines all of the formatted parts of the packet, such as UART_START_BYTE, UART_READ_SPI_BYTE, and UART_WRITE_SPI_BYTE. These are accompanied by definitions to specify where in the packet header these commands are found. In your implementation you

may want to change some of these values. Ensure that the UART start and read/write bytes are bytes that you would not expect to see in your application.

6 Software Flowchart

Figure 6-1 shows the code flow diagram for this example and explains the different UART Bridge wait states and the actions the device takes in each state. The flowchart also shows the Interrupt Service Routines for UART and SPI.

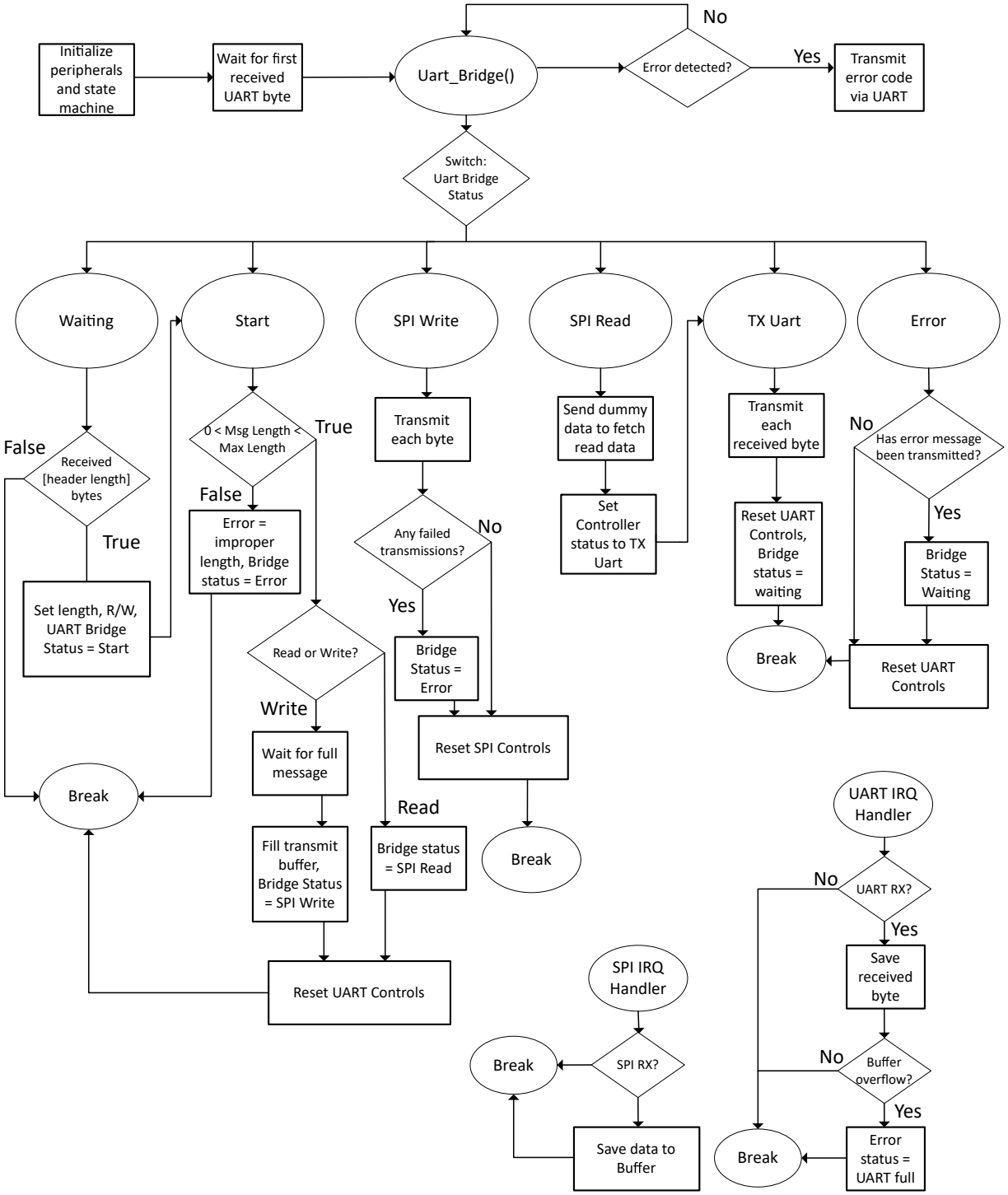

Figure 6-1. Software Flowchart

Device Configuration

This application makes use of 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.

The code described in the software flowchart is found in the *uart_to_spi_bridge.c* file.

Required UART Packets

The figure below shows the required UART packet for performing reads and writes with the SPI. The values shown are the default header values defined in the example.

- Start Byte: The value used by the bridge to indicate a new transaction in starting. UART transmissions are ignored until this value is seen by the bridge.
- SPI Read or Write Indicator: This value tells the bridge whether to perform a read from or a write to the SPI peripheral device.
- Message Length N: The length of the data being transferred in bytes.
- D0, D1, ..., D(N-1): Data being transferred to the bridge

Note

the Read packet includes only the header. When conducting a read, there is no need to send data after the packet. The bridge device automatically sends the correct amount of dummy data to the SPI peripheral to fetch the read data.

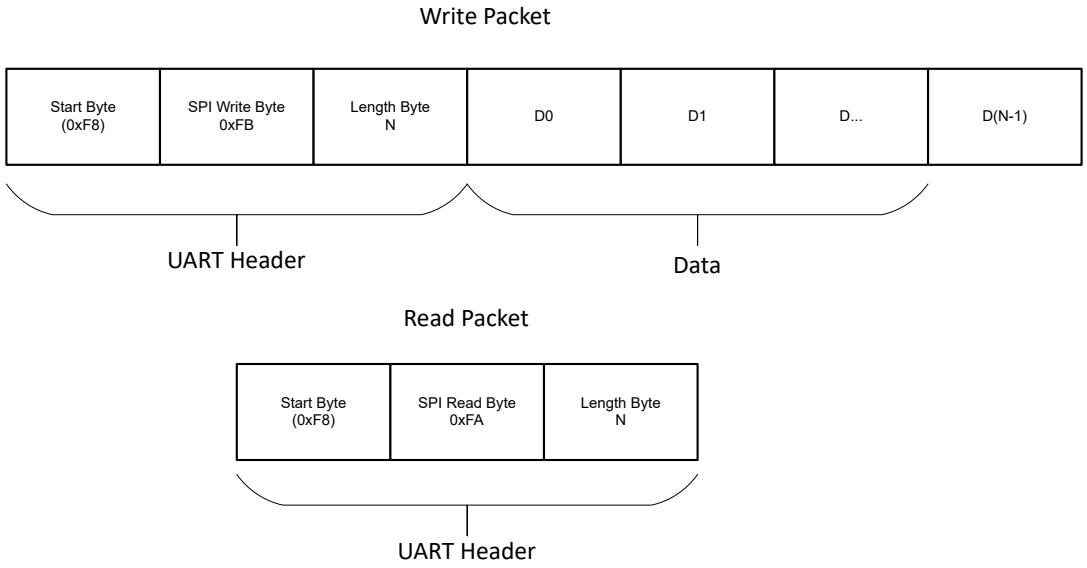

Figure 6-2. UART Write and Read Packet Format

7 Application Code

Some users may want to change the specific values that are used by the UART packet header, or the maximum packet size. This can be done by modifying the #define values found in the beginning of the uart_to_spi_bridge.c file as shown below.

```
/* Define UART Header and Start Byte*/
#define UART_HEADER_LENGTH  0x02
#define UART_START_BYTE     0xF8
#define UART_READ_SPI_BYTE  0xFA
#define UART_WRITE_SPI_BYTE 0xFB
#define RW_INDEX            0x00
#define LENGTH_INDEX        0x01

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

Many portions of the code are intended to be used for error detection and handling. At these points in the code, the user may want to use additional error handling or reporting for a more robust application. For example, the code segment shown below demonstrates a way to check for errors in SPI transmissions, and sets and error flag in the event of an error. The user may want to quit sending and change the UART Bridge Status here to reflect the error. This and many other areas in the code have options for error consideration.

```
for(int i = 0; i < gMsgLength; i++){
                if(!DL_SPI_transmitDataCheck8(SPI_0_INST, gSPIData[i])){
                    gError = ERROR_SPI_WRITE_FAILED;
        }
}
```

8 Additional Resources

1. Texas Instruments, Download the MSPM0 SDK
2. Texas Instruments, Learn more about SysConfig
3. Texas Instruments, MSPM0L LaunchPad™
4. Texas Instruments, MSPM0G LaunchPad™
5. Texas Instruments, MSPM0 SPI Academy
6. Texas Instruments, MSPM0 UART Academy

9 E2E

Please visit TI's E2E website to view discussions and post new threads in order to get technical support for utilizing MSPM0 devices in your design.

10 Revision History

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

DATE	REVISION	NOTES
January 2024	*	Initial Release

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