Subsystem Design

I2C IO Expander

1 Description

This subsystem example demonstrates how to configure an IO expander with MSPM0 along with a controller. The configuration procedure sets PIN direction, state, and reads state.

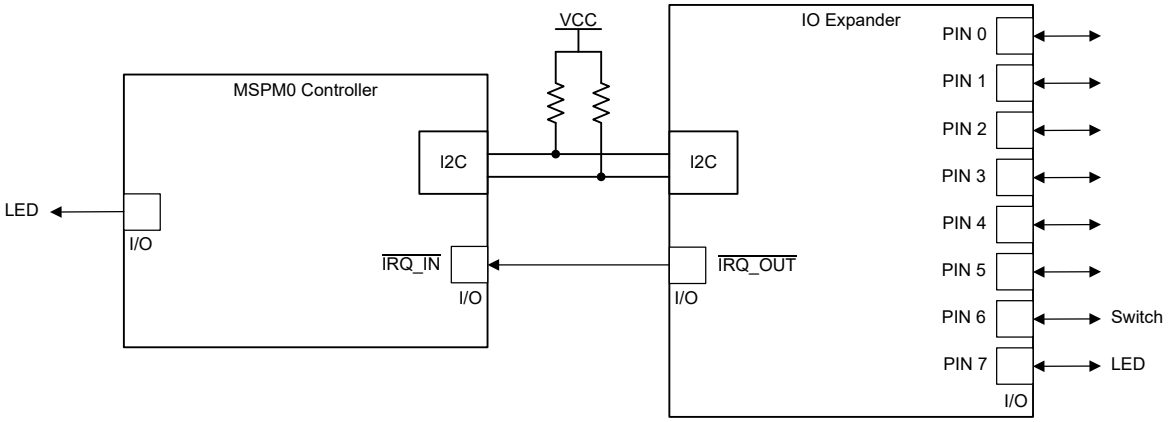

Figure 1-1. Subsystem Functional Block Diagram

2 Required Peripherals

Table 2-1 and Table 2-2 describe the *required* integrated peripherals.

Table 2-1. Required Peripherals for Controller

Subblock Functionality	Peripheral Use	Notes
LED and Interrupt Pin	(2 ×) GPIO	Shown as *LED* and *IRQ_IN* in code
I2C Controller	(1 ×) I2C	Shown as *I2C* in code

Table 2-2. Required Peripherals for IO Expander

Subblock Functionality	Peripheral Use	Notes
GPIO and Interrupt Pin	(9 ×) GPIO	Shown as *PIN_0*, *PIN_1*, *PIN_2*, *PIN_3*, *PIN_4*, *PIN_5*, *PIN_6*, *PIN_7*, and *IRQ_OUT* in code
I2C Target	(1 ×) I2C	Shown as *I2C* in code

3 Design Steps

1. Configure the controller and peripheral I2C instance, GPIO pins, GPIO switches, and GPIO LEDs in SysConfig.
2. Set I2C clock speed in SysConfig. Default is 400kHz for LaunchPad™ development kits with external pullups.
3. Define the required I2C packet for proper communication.
4. Create a demonstration that toggles IO expander outputs and an input from a switch that toggles an LED on the controller.

4 Design Considerations

1. **Detecting pin state changes**: The IO expander utilizes an interrupt to notify the controller to request a read if an input pin changes state. The IO expander also updates the locally saved pin state any time a GPIO pin observes a rising or falling edge.
2. **Changing GPIO direction**: Because MSPM0 can enable GPIO input and output at the same time, GPIO input is always enabled. Direction bytes are set to enable or disable GPIO output. The bit position within each byte determines output control. This allows for all pins to use the rising and falling edge interrupt and the ability to read all pin states.

5 Software Flow Chart

Figure 5-1 shows the main function code for the IO expander. The main function initializes the peripherals and then enters a loop to handle received I2C communication.

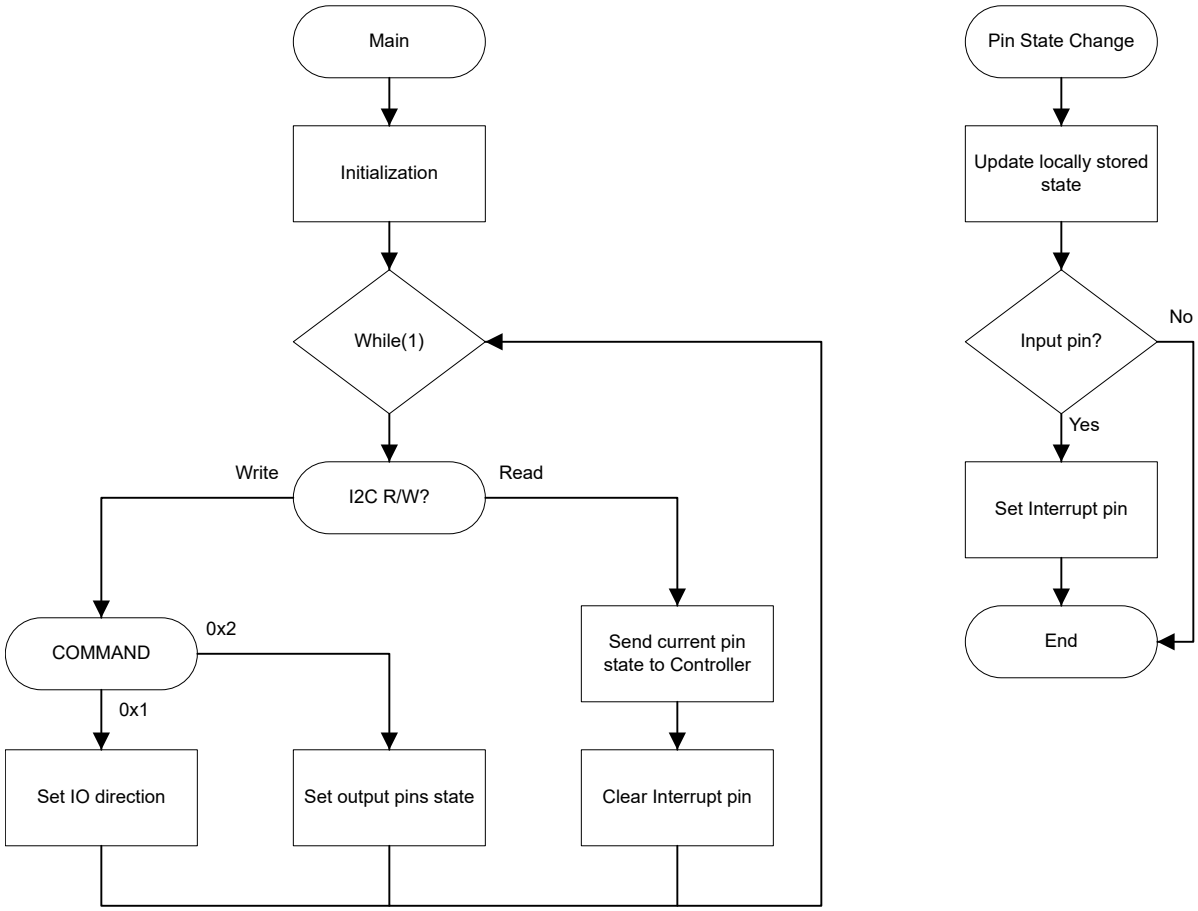

Figure 5-1. IO Expander Software Flow Chart

Figure 5-2 shows the main function code for the controller. The main function initializes the peripherals, sends the I2C command to set the IO direction, and enters a loop that toggles the output pins on the IO expander.

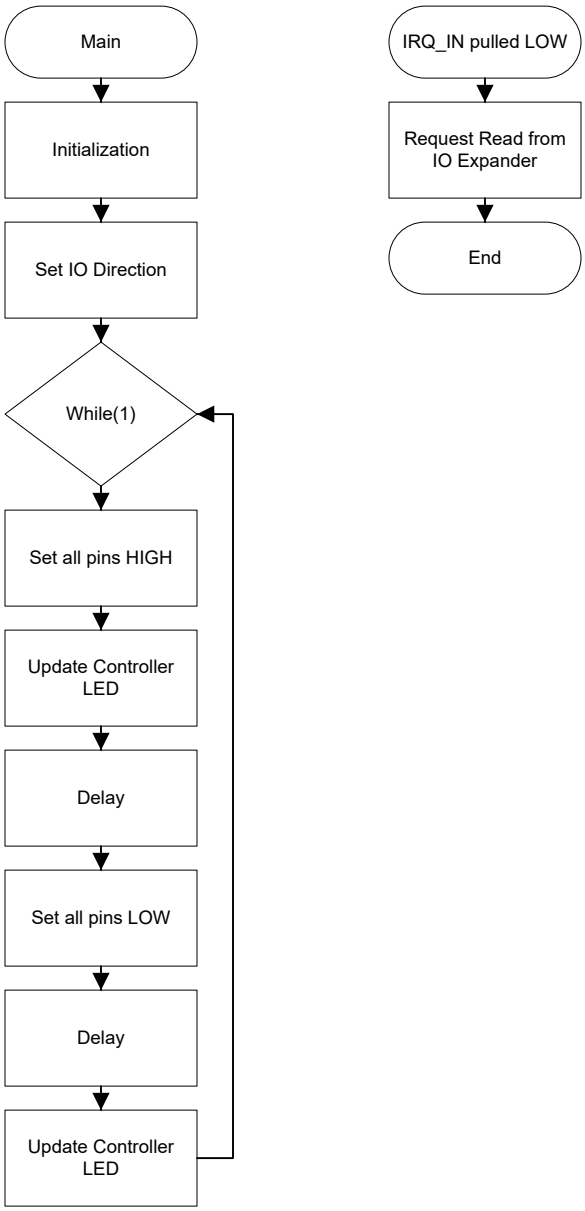

Figure 5-2. Controller Software Flowchart

6 Required I2C Packet

Figure 6-1 and Figure 6-2 show the required I2C packet for proper communication using the I2C interface.

- **COMMAND**: byte containing command to either change IO direction or set output pin state.
  - **0x1**: set GPIO direction
  - **0x2**: set output pin state
- **DATA**: byte containing pin configuration
  - **COMMAND = 0x1**: 1 represents output, 0 represents input
  - **COMMAND = 0x2**: 1 represents output HIGH, 0 represents output LOW

Table 6-1. I2C Packet Breakdown

Function	COMMAND	DATA
Set GPIO direction	0x1	1: Output
		0: Input

Table 6-1. I2C Packet Breakdown (continued)

Function	COMMAND	DATA
Set output pin state	0x2	1: High
		0: Low

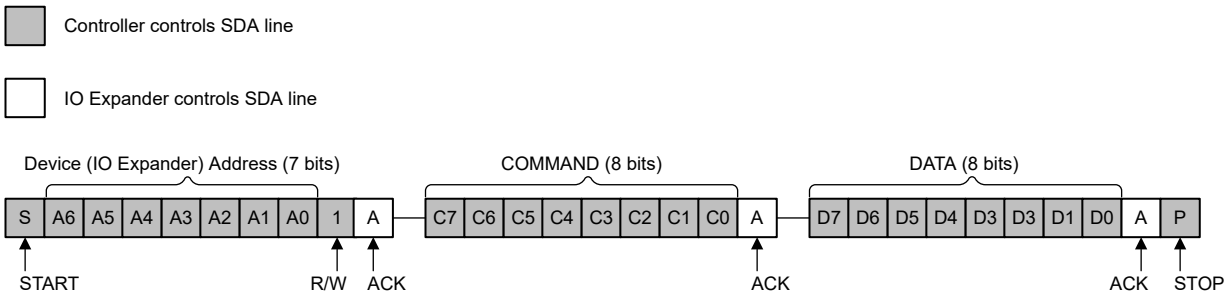

Figure 6-1. I2C Write Packet

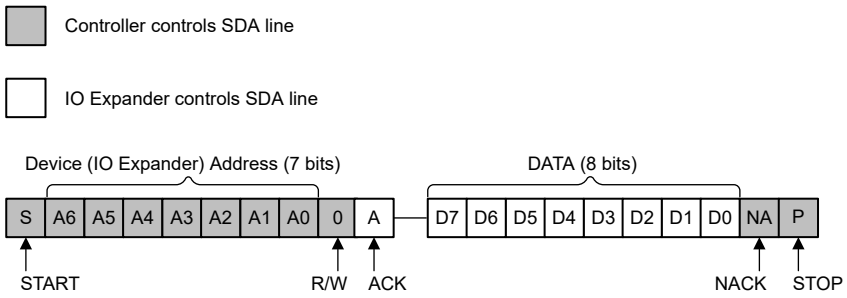

Figure 6-2. I2C Read Packet

7 Application Code

The code example updates the stored GPIO state any time a pin observes a rising or falling edge. The IRQ_OUT pin is only pulled low when an input pin observes a rising or falling edge, and is pulled high once the controller requests a read. The TX FIFO is filled upon receiving an I2C Start and is flushed upon an I2C Stop:

```
//I2C INT
void I2C_INST_IRQHandler(void)
{
    switch (DL_I2C_getPendingInterrupt(I2C_INST)) {
        case DL_I2C_IIDX_TARGET_START:
            /* Fill TX FIFO with current pin state */
            DL_I2C_fillTargetTXFIFO(I2C_INST, &gGpioState, 1);
            break;
        case DL_I2C_IIDX_TARGET_RXFIFO_TRIGGER:
            /* Store received data in buffer */
            while (DL_I2C_isTargetRXFIFOEmpty(I2C_INST) != true) {
                receivePacket(DL_I2C_receiveTargetData(I2C_INST));
            }
            break;
        case DL_I2C_IIDX_TARGET_TX_DONE:
            /* Pull interrupt pin high when Controller reads from IO Expander */
            DL_GPIO_setPins(GPIO_GRP_0_PORT, GPIO_GRP_0_IRQ_OUT_PIN);
            break;
        case DL_I2C_IIDX_TARGET_STOP:
            /* Flush TX FIFO */
            DL_I2C_flushTargetTXFIFO(I2C_INST);
            break;
        default:
            break;
    }
}

void GPIOA_IRQHandler(void)
{
    /* Store the current pin state */
```

```
    gGpioState = (DL_GPIO_readPinStatus(GPIO_GRP_0_PIN_7_PIN) << 7) |
                 (DL_GPIO_readPinStatus(GPIO_GRP_0_PIN_6_PIN) << 6) |
                 (DL_GPIO_readPinStatus(GPIO_GRP_0_PIN_5_PIN) << 5) |
                 (DL_GPIO_readPinStatus(GPIO_GRP_0_PIN_4_PIN) << 4) |
                 (DL_GPIO_readPinStatus(GPIO_GRP_0_PIN_3_PIN) << 3) |
                 (DL_GPIO_readPinStatus(GPIO_GRP_0_PIN_2_PIN) << 2) |
                 (DL_GPIO_readPinStatus(GPIO_GRP_0_PIN_1_PIN) << 1) |
                 (DL_GPIO_readPinStatus(GPIO_GRP_0_PIN_0_PIN));

    /* Loop through all pins */
    for (int i = 0; i < 8; i++) {
        /* Check if the current pin state changed */
        if (((gGpioState >> i) & 0x1) != ((gLastGpioState >> i) & 0x1)) {
            /* If the pin is an Input */
            if (((gGpioDirection >> i) & 0x1) == 0) {
                /* Pull interrupt pin LOW when an input pin state changes */
                DL_GPIO_clearPins(GPIO_GRP_0_PORT, GPIO_GRP_0_IRQ_OUT_PIN);
            }
        }
    }
    gLastGpioState = gGpioState;
}
```

The controller IRQ_IN interrupt is triggered upon a detected falling edge, where a read to the IO expander is requested:

```
void GPIOA_IRQHandler(void)
{
    /* Set LED HIGH */
    DL_GPIO_clearPins(GROUP0_PORT, GROUP0_LED_PIN);

    /* Request a read from the IO Expander */
    gRxLen = I2C_RX_PACKET_SIZE;
    gRxCount = 0;

    /* Wait until the I2C bus is idle */
    while (!(DL_I2C_getControllerStatus(I2C_INST) & DL_I2C_CONTROLLER_STATUS_IDLE));

    /* Start a read operation */
    gI2cControllerStatus = I2C_STATUS_RX_STARTED;
    DL_I2C_startControllerTransfer(I2C_INST, I2C_IO_EXPANDER_ADDRESS,
                                   DL_I2C_CONTROLLER_DIRECTION_RX, gRxLen);
    /* Enable RX FIFO trigger interrupt */
    DL_I2C_enableInterrupt(I2C_INST, DL_I2C_INTERRUPT_CONTROLLER_RXFIFO_TRIGGER);
}
```

8 Additional Resources

- Texas Instruments, Download the MSPM0 SDK
- Texas Instruments, Learn more about SysConfig
- Texas Instruments, MSPM0C LaunchPad™ Development Kit
- Texas Instruments, MSPM0L LaunchPad™ Development Kit
- Texas Instruments, MSPM0G LaunchPad™ Development Kit
- Texas Instruments, MSPM0 Academy

9 E2E

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

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