I2C Slave Lab Assignment

Objective

Get hands on experience working with and understanding how I2C works by implementing an I2C slave device.

Assignment

- The I2C master driver is already implemented in SJSU-Dev2.
- Study the existing I2C code: **i2c.hpp** file and extend it to handler slave mode operation.
- Flashing the example project **CommandLine** on to the board (completely unmodified) and connect it to your 2nd Slave Board which will contain your i2c slave driver.
  - On your **master board**, you can just use the `i2c` terminal command to read and write to the I2C registers of a slave device.
  - See `help` for the help about the command

Part 0: Read the I2C Chapter

> IGNORE THE Software Implementation of the I2C chapter. It is not correct and does not match the proper information laid out in the state diagram and truth tables.

1. Read over the I2C chapter and the various registers it contains. You may ignore the DMA registers. In order to grasp the amount of information in the chapter, you may need to read it multiple times.
2. Draw out the state machine outlined in the chapter's state diagram and truth table for yourself.

Part 1: Get I2C Slave Interrupt to fire

In this ***Part 0*** of this assignment, your objective is to simply to initialize your I2C slave to get its first state interrupt to fire.

1. Add your `InitializeAsSlave()` method
2. Add the **slave address recognized** state into your I2C slave driver and print a message when you hit this state inside the ISR.
3. Connect this Master Board to the Slave Board by wire-wrapping the I2C SCL/SDA wires together.
4. To test that your slave driver initialize is working, invoke `i2c discover` on the master board. Your slave board's address will appear.

> When you connect the two boards to I2C There will be two of each sensor with the same address, which too is sort of okay. For example, the temperature sensor at the same address (from the 2 boards).

```
#include <cstdint>
#include <cstdio>
#include "L1_Drivers/i2c.hpp"
// Slave Board sample code reference
int main(void)
{
    // Create I2c object (defaults to I2C2)
    I2c i2c;
    // Pick any address other than an existing ones on the board. Use `i2c discover` to see what those
    const uint8_t slaveAddr = 0xC0;
    // Our devices read/write buffer (This is the memory your master board will read/write)
    volatile uint8_t buffer[256] = { 0 };
    // I2C is already initialized before main(), so you will have to add initSlave() to i2c base class
    i2c.InitializeSlave(slaveAddr, buffer, sizeof(buffer));
    // I2C interrupt will (should) modify our buffer.
    // So monitor the buffer, and print and/or light up LEDs
    // ie: If buffer[0] == 0, then LED ON, else LED OFF
    uint8_t prev = buffer[0];
    while(true)
    {
        if (prev != buffer[0])
        {
            printf("buffer[0] changed from 0x%08X to 0x%08X by the other Master Board\n", prev, buffer
            prev = buffer[0];
        }
    }
    return 0;}
```

Part 2: Implement I2C slave behavior in I2CHandler

1. Using the state machine diagrams in the datasheet before you begin, make sure you fully understand the slave states. The code you implement will need to follow the state machine guidelines exactly.
2. Extend I2C state machine to handle I2C slave operations.

> You may add printf statements to the I2C state machine code to identify what states you enter when the Master Board is trying to do an I2C transaction.

Requirements

See Canvas.

> **Extra Credit:**
>
> • Get multi byte read and write operation to work.
> • Do something creative with your slave board since you have essentially memory mapped the slave device over I2C. Maybe use buffer[0] to enable a blinking LED, and buffer[1] controls the blink frequency? You can do a lot more. Just blinking the LEDs is not enough.

?

Revision #3
Created 1 month ago by Khalil Estell
Updated 1 week ago by Khalil Estell