Lab Assignment: I2C Slave Driver

Assignment

- The `I2C#2` master driver is already implemented and used for the on-board SJ-One I2C devices.
- Study the existing I2C code: `i2c_base.cpp`. Please ask any questions if you have any, but the driver was implemented using the state machine diagrams as part of the I2C information at this website.
- You will use the sample project on one board (completely unmodified) and connect it to your 2nd Slave Board which will contain your `I2C2` Slave driver
    - On your `Master Board`, you can just use I2C terminal command to read or write an I2C register of a slave device
    - See `help i2c` for the help of this command (run this command on the `Master Board`)

Part 0: Get I2C Slave Interrupt to fire

Using the state machine diagrams you drew will be critical before you begin. Make sure you fully understand the slave states because you will write the code that matches exactly to your state machine diagram. In this ***Part 0*** of the assignment, your objective is simply to initialize your I2C slave to get its first state interrupt to fire, and then the rest will be easy.

- Start with the `Master Board` running the default, unmodified FreeRTOS sample project
- Familiarize yourself with the `Master Board's` terminal command: `help i2c`
- Connect this `Master Board` to the `Slave Board` by wire-wrapping the I2C SCL/SDA wires together.
    - Your pull-up resistor values will be halved but this is okay
    - 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) now appears twice on your I2C bus.
- Initialize the `Slave Board` slave address such that you will get an interrupt when your address is sent by the `Master Board`
- Add `printfs` to the I2C state machine code to identify what states you enter when the `Master Board` is trying to do an I2C transaction
- Use your I2C slave state machine diagrams to augment the existing code, particularly at the state machine function to complete the rest of the assignment

```
#include <stdint.h>
```

```
#include <stdio.h>
#include "i2c2.hpp"
// TODO: Modify the handleInterrupt at the I2C base class
// 1. Add your slave init method
// 2. Add the "slave address recognized" state in your I2C slave driver and print a msg when you hit tl
// 3. To test that your slave driver init is working, invoke "i2c discover" on the Master board
// Slave Board sample code reference
int main(void)
{
    I2C2& i2c = I2C2::getInstance(); // Get I2C driver instance
    const uint8_t slaveAddr = 0xC0;  // Pick any address other than an existing one at i2c2.hpp
    volatile uint8_t buffer[256] = { 0 }; // Our slave read/write buffer (This is the memory your othe
    // I2C is already initialized before main(), so you will have to add initSlave() to i2c base class
    i2c.initSlave(slaveAddr, &buffer[0], sizeof(buffer));
    // I2C interrupt will (should) modify our buffer.
    // So just monitor our 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(1)
    {
        if (prev != buffer[0]) {
            prev = buffer[0];
            printf("buffer[0] changed to %#x by the other Master Board\n", buffer[0]);
        }
    }
    return 0;}
```

Since the I2C state machine function is called from inside an interrupt, you may not be able to to use `printf()`, especially if you are running FreeRTOS. As an alternative, use the debug printf methods from the `printf_lib.h` file.

?

Guideline

Design your I2C slave interface by extending the existing I2C master driver. Use one person's board as `Master` and communicate with the second person's `Slave` board.

1. Study `i2c_base.cpp` , particularly the following methods:
    - `init()`
    - `i2cStateMachine()`
      Note that this function is called by the hardware interrupt asynchronously whenever I2C state changes.
      The other I2C master will simply "kick off" the START state, and this function carries the hardware through its states to carry out the transaction.
    - The functions you add to this base class are accessible by the I2C2 instance.

2. Add `initSlave()` method to the I2C class to initialize the slave operation.
    - Allow the user to supply a memory to be read or written by another master.

3. Extend the state machine for the I2C slave operation.
    - Use the diagrams you drew from the previous I2C assignment

Requirements

1. Demonstrate that you are able to read and write the slave memory from the `Master Board`
    - You can use the `i2c` terminal command for this.
    - On the slave device, you can print out when the data sent to you has changed.

2. Submit logic analyzer screenshots of a read and a write transaction occurring with your slave.

> **Extra Credit:**
>
> + For extra credit and bragging rights, get multi byte read and write operation to work and impress the ISA team.
>
> + 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, so do not just blink LEDs

Revision #10
Created 5 months ago by Admin
Updated 2 months ago by Preet Kang