Dylan Sandall

12/8/2022

CPE-333

333 Final Project - Hardware I2C Communication & LCD1602 Driver

Procedure

A useful addition for the RISC-V Otter would be the ability to easily communicate with external modules using the well-established I2C protocol. My project is to implement this communication protocol in hardware, as well as provide a basic software solution for interfacing with one such external device, the LCD1602 Character Display.

I began work on this project by researching the I2C protocol and identifying which version I would support. I decided on the base I2C, standard speed, single controller, with 7 bit addresses. None of the limitations of the basic protocol would be a problem for the relatively simple end goal I had. Starting with the basic multi-stage Otter provided on canvas, I wrote a support module (i2c_connect.sv) for the controller core (i2c_master.sv) from circuitden.com, which provides a way for the Otter to write to and read from the I2C bus. SDA/SCL lines, MMIO registers for I2C data, and a basic "busy" status flag were added to the wrapper (OTTER_wrapper.sv).

Once functionality of the simulated I2C device and Otter was verified, I moved onto adapting the LCD1602 library from https://github.com/bitbank2/LCD1602. This is necessary, as the base LCD1602 actually uses a unique parallel interface and is commonly adapted to I2C using a backpack board. I2C transmission data must provide the clock and control signals for this display, as well as the output data. The most user-friendly solution would be to adapt the library to work directly on the Otter using the provided C compiler, but I desired a lighter and hopefully more efficient codebase. I wrote a basic C program that uses the library to write a series of hex values to the console based on the provided string input (the desired display reading). This series of hex values handles initialization of the LCD1602, backlight control, and writing text. I then manually move these hex bytes into an assembly program I wrote that takes care of the MCU's share of the logic.

Engineering Hurdles

Ideally, the C translation library would dump these values to a file that could then be read by the Otter from memory with a few tweaks to my assembly program, but the current scheme works well enough as a proof of concept. Unfortunately, I could not get a working demo of the display in action. Simulations for write and read operations work (mostly) as expected and the program will run on the board with a simulated I2C device, but something is preventing communication with the display in practice. I suspect the problem is with the core I chose, as it is only capable of sending 2 sequential bytes without readdressing the I2C device. The other problem with the core I chose is an odd delay between the addressing and data transfer of every write command. It's as if it's sending a blank byte of data before every intended byte of data, cutting data transmission speed in half. Ideally, it would be able to send a single address packet followed by all the data bytes, and this may be causing a problem with the display.

Documentation/Usage Guide

An end user should follow the following procedure to use RISC-V Otter for I2C communication:

- Using MMIO
    - Write 7 bit device address to the "slave addr register"
    - If writing to device: write data byte to "MOSI", clear "R/W", and set "new message"
    - If reading from device: set "R/W" and set "new message"
    - Poll for "new message" bit, when cleared (by I2C module) transfer is complete
    - If reading from device: read data from "MISO"

This is also detailed in the provided assembly program, "I2Cdemo.asm", as well as a handy subroutine that handles R/W and new message flags, as well as polling for data transfer completion.

An end user should follow the following procedure to use the LCD1602 Translation Library for RISC-V Otter:

First of all, probably don't. It's not an efficient workflow to copy hex values and paste them into your assembly code. For actual use, adapt the code to write to a file that can be read by the Otter, or write something new. But, if you insist, open LCD1602_hexTranslation.c and change `lcd1602WriteString("A");` to whatever you wish to write to the 1602 display, run the file and copy hex values into I2CDemo.asm, utilizing the format and subroutine provided.

Sources and Provided Files

I2C Project Documentation - This file.

I2Cdemo.asm - Assembly file and basic framework for I2C interfacing.

LCD1602_hexTranslation.c - C program that handles initialization of LCD1602 and translation of text to necessary I2C data bytes. Based on https://github.com/bitbank2/LCD1602.

Annotated Timing Diagram.pdf - An annotated Vivado simulation waveform of the I2Cdemo.asm program. Highlighted is a breakdown of each segment of the I2C transmission, including addressing, data, and known bugs from the core.

(Subdirectory) vivadosource - Vivado source and constraint files used in this version of the Otter. **Files with significant changes include** - simTemplate-1.v, OTTER_wrapper.sv, i2c_connect.sv, otter_memory.mem, and Basys-3-Master.xdc. Files i2c_master, i2c_slave, and i2c_master_tb from https://github.com/0xArt/Tiny_But_Mighty_I2C_Master_Verilog.