1 Introduction

The objective of this lab is to read sensor data conveyed by a sensor (VD-004) to your simulated CPU using I2C master write transactions. Your simulated CPU will store each sensor value to a storage device (VD-005) using I2C master write operations.

This lab will use a “bit-banged” implementation of the I2C protocol.

Part	Due Date
Code	Apr. 28

Figure 1: Table of due dates for each part.

Contents

2 How to Submit

When your code is ready to turn in, please submit only your `main.cpp` file via Moodle. Note that you should write all code that you need within `main.cpp`. As before, your code must compile to receive any credit.

3 Simulated Devices

In this lab, you will interface with two virtual devices using I2C, a sensor and a flash memory.

Normally, all three devices should share common `scl` and `sda` lines, as shown below.

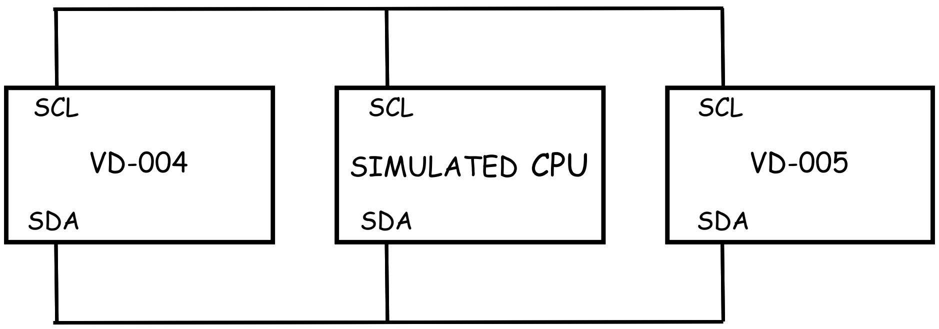

Figure 2: Topology of the simulated devices.

However, due to limitations of software emulation using Verilator, we adopt a slightly different approach, where the values of `scl` and `sda` will be automatically selected from the values of `scl` and `sda` being driven from each of the three devices. Only one device controls each line at one time.

This is relevant when viewing the waves generated from your code, in which you'll see traces for `sda`, `sda_i`, `scl`, and `scl_i`. The values for `sda` and `sda_i` will always be identical except during periods when VD-004 or VD-005 is actively driving the wire (the same applies to `scl` and `scl_i`). Note that you will still read and write to `IO_SDA` and `IO_SCL` and you don't need to do anything special to "yield" control of either line to one of the peripherals (they will claim these lines whenever needed).

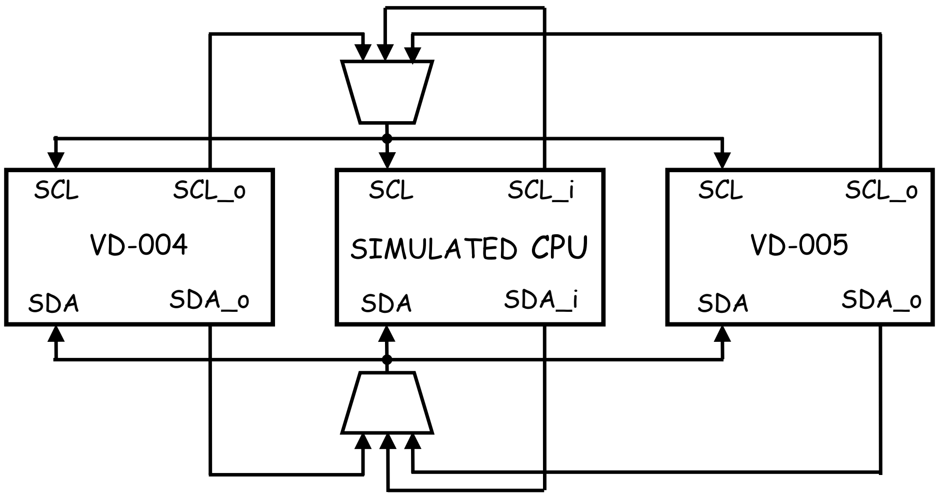

Figure 3: Topology of the simulated devices.

3.1 VD-004: Simulated Sensor

Read this section carefully as it has changed since the previous lab.

The VD-004 collects sensor readings that may contain between 1 and 128 bytes of data. It will begin collecting data once your design reads the NPAGE register from VD-005 (see below). After this, VD-004 will perform an I2C master write operations to the simulated CPU after every 200 clock cycles of idle time.

When VD-004 is ready to send a batch of sensor values, it will send $n + 1$ write transactions to the simulated CPU. The first of these will contain the number of sensor readings to follow. The next n transactions will contain the sensor values. Afterward it will repeat this sequence.

The I2C address of your simulated CPU is 0x7A. Your simulated CPU should only repond to the master write request if the requested address is correct.

3.2 VD-005: Simulated Flash Memory

The VD-005 simulates a flash memory storage device. In this lab it will use a simplified interface, in which consecutive writes to the DATA register will automatically be placed into consecutive locations on the selected page. This way, your code will no longer need to write the OFFSET register. Also, you no longer need to write the magic number.

The register map for VD-005 is shown in the table below.

Register Address	Name	Mode(s)	Description
`0x1D`	`NPAGE`	read	Returns the number of pages on the flash device.
`0x1B`	`PAGESEL`	write	Selects which page of memory to write.
`0x1C`	`WHO_AM_I`	read	Always contains the value `0x36`
`0x1F`	`DATA`	write	Specifies the value at the active address on the active page.

Figure 4: Table of VD-005 registers.

3.3 Program Requirements

Your program should first query the `WHO_AM_I` on VD-005 and verify it returns the correct value. If not, then your program should display an error message (on standard out) and exit.

Your program should also query the `NPAGE` register of the VD-005 to determine the number of pages of memory available. Note that reading this register will cause VD-004 to begin performing I2C master write transactions after every 200 cycles of idle time.

Finally, your program should enter an infinite loop, waiting for an I2C master write from VD-004 containing the number of sensor readings, writing the page number to VD-005, then alternating between waiting for each new sensor reading from VD-004 and writing it to VD-005.

This will give way to the following sequence:

- read VD-005.WHO_AM_I
- read VD-005.NPAGE
- receive VD-004.NREADINGS
- write VD-004.PAGESEL
- receive VD-004.D0 (first sensor byte)
- write VD-004.DATA
- receive VD-004.D1 (second sensor byte)
- write VD-004.DATA
- ...
- receive VD-004.Dn (nth sensor byte)
- write VD-004.DATA
- receive VD-004.NREADINGS
- write VD-004.PAGESEL
- receive VD-004.D0 (first sensor byte)
- write VD-004.DATA
- receive VD-004.D1 (second sensor byte)
- write VD-004.DATA
- ...

Once your program has exhausted the available supply of flash memory pages (the number of pages available can be determined by the `NPAGE` register of the VD-005), it should exit with an informative message indicating no further flash memory is available.

4 Programming Environment

The programming environment for this lab is very different from the previous lab. There only two pins: `IO_SDA` and `IO_SCL`, which may be read with `read_io()` or written with `write_io()`.

You do not need to implement recovery from NACK (not-acknowledge) flags in I2C transactions.

5 Provided Code

We will provide you with the following functions and macros:

- the `main()` function
- the `master_i2c()` function, which performs a master read or write operation (use this to communicate with VD-005),
- the `WAIT_FOR_START_BIT()` macro, which will advance the simulation until an I2C start bit is seen; after this macro returns, expect the simulation time to be one simulation cycle past the falling edge of SDA,
- the `WAIT_FOR_STOP_BIT()` macro, which will advance the simulation until an I2C stop bit is seen; after this macro returns, expect the simulation time to be one simulation cycle past the rising edge of SDA,
- the `WAIT_FOR_RISING_SCL()` macro, which will advance the simulation until the next rising edge of SCL; after this macro returns, expect the simulation time to be one simulation cycle past the rising edge of SCL, and
- the `WAIT_FOR_FALLING_SCL()` macro, which will advance the simulation until the next falling edge of SCL; after this macro returns, expect the simulation time to be one simulation cycle past the falling edge of SCL.

6 Suggested Plan of Work

We suggest you follow these steps:

1. Read the comments in the code, including the placeholders in the `slave_i2c()` function
2. Fill in the missing code in the `slave_i2c()` function
3. As with Lab 2 and Lab 3, test your code with "make simulation && ./simulation," "make waves," "make decode," and "make flashdump".

7 Grading

Your code will be inspected for style and correctness by a human reader. This aspect of grading will be fairly lenient and mostly for the purpose of giving you useful feedback. However you may still lose points for egregious stylistic problems, or failing to write code that clearly attempts to solve the problem at hand.

Your code will also be run against the same simulated sensor as you are given in the project skeleton, however the hard-coded sensor "readings" will be changed to different values. **The length of sensor data streams and the number of flash memory pages will also be changed during grading, so make sure your program accounts for this.**

Additionally, your code will be run against a version of the simulated sensor and/or flash memory which is defective, and reports an incorrect value when the `WHO_AM_I` register is read. In such cases, your program should exit with an informative error message. **You may assume that if a VD-004 or VD-005 reports a correct `WHO_AM_I` value, it is not defective.**

The correctness of your code will primarily judged by evaluating the output of `make flashdump` using an alternate set of sensor data, however any printouts your code displays, as well as the output of sigrok via `make decode` may be used as a supplement.

If your code does not compile and run on the CSCE linux lab computers, you will not receive credit.

8 Rubric

- 10 points - Code style
- 10 points - Correct checking of `WHO_AM_I` register
- 10 points - Data is read via I2C master writes from the sensor.
- 10 points - Data is stored in the VD-005 in the proper format.
- 10 points - Data stored in the VD-005 is correct/matches the hard-coded values from the sensor (format must also be correct).
- $\frac{5}{2} \times b$ points - See "Bug Bounty".

Maximum number of points possible: 50.

Keep in mind that some items not listed on the rubric may cause you to loose points, including cheating, submitting code which is inconsistent with what you have demonstrating, plagiarizing code or reflection content, etc.

9 Bug Bounty (Extra Credit Opportunity)

There is an extra credit opportunity available for this lab. If you find a bug in our code, we will increment the b counter in the rubric above once for each bug. In other words, you will earn 2.5 bonus points per bug. If this puts your grade on this project higher than 50 points, you will receive b many bonus points on the final exam (i.e. each bug you find will improve your final exam score by 1%).

To take advantage of this opportunity, you must send us an email with the following example:

- A zipped up copy of your code.
- A clear description of the bug and how to re-produce it.
- A clear description of how the bug causes the code to behave in a manner inconsistent with what is documented in this lab sheet, or in a fashion that is otherwise problematic.

If your bug stems from your own code, or a mis-understanding of the course material, you will neither gain nor lose points (b remains unchanged). However submitting a bug report without all three of the items above will result in b being decremented, and possibly becoming negative as a result.

Minor typos, compiler warnings, etc. do not count as bugs, but please feel welcome to report them if you find them.

If multiple students discover the same bug, they will all receive the bonus. However, reports emailed after a fix has been announced or otherwise distributed to the class are not eligible to receive the extra credit. To receive the bonus, your bug report must be submitted no later than Monday, April 27, 2020 at midnight eastern time.