Department of Electrical and Computer Engineering
University of Wisconsin-Madison

ECE 315
Introductory Microprocessor Laboratory

Syllabus and Lab Manual
(Lab Section 1)

Spring 2010

Michael G. Morrow

Last updated 1/13/2010 2:54 PM

Table of Contents

1 Course Administration 3
1.1 Faculty Supervisor 3
1.2 Laboratory Instructors 3
1.3 Timetable 3
1.4 Goals 3
1.5 Course Prerequisites 3
1.6 Laboratory Schedule 3
1.7 Computer Usage 3
1.8 Required Text 4
1.9 Reference 4
1.10 Student Responsibility 4
1.11 Grading 4
1.12 Laboratory Rules 5
1.13 Course Improvement 5
2 Lab #0 – Hardware Introduction 6
2.1 Prelab Assignment 6
2.2 Set-up 6
2.3 Lab Exercise 6
2.4 Post-Lab Deliverables 8
Grading Criteria 9
3 Lab #1 – Multiplexed LED Display 10
3.1 Prelab Assignment 10
3.2 In-Lab Assignment 10
3.3 Post-Lab Deliverables 12
3.4 Grading Criteria 12
4 Lab #2 – Keypad Interfacing 13
4.1 Pre-Lab Assignment 13
4.2 Assignment 13
4.3 Post-Lab Deliverables 14
4.4 Grading Criteria 14
5 Lab #3 – External Memory Interface 15
5.1 Pre-Lab Assignment 15
5.2 In-Lab Assignment 16
5.3 Lab Deliverables 16
5.4 Grading Criteria 16
6 Lab #4 – Stepper Motor Operation 17
6.1 Pre-Lab Assignment 17
6.2 In-Lab Assignment 19
6.3 Lab Deliverables 20
6.4 Grading Criteria 20
7 Lab #5-6 – Project 21
7.1 Assignment 21
7.2 Sample Project Ideas 21
7.3 Important Notes on Using Analog Inputs and Outputs 22
7.4 Generating Random Numbers 22
7.5 Deliverables 23
7.6 Grading Criteria 23
8 Lab Equipment Reference 25
8.1 Quick Guide to the HP 54622D Oscilloscope 25
8.2 The HP Word 54600 Toolbar 25

1 Course Administration

1.1 Faculty Supervisor

Michael G. Morrow, 3537 Engineering Hall
Ph: 265-9007 email: morrow@engr.wisc.edu
Office hours: Posted on the course web page at URL
http://eceserv0.ece.wisc.edu/~morrow/schedule.htm

1.2 Laboratory Instructors

Current laboratory instructor assignments, contact information, and office hours will be posted on the course web page.

1.3 Timetable

All lab sections meet in 3650 Engineering Hall.

Section 1	Tuesday	2:25pm-5:25pm	(starts 1/2)
Section 2	Monday	7:00pm-10:00pm	(starts 3/)

Work on ECE 315 projects may be performed in 3650 Engineering Hall at other times when the building is open (7:30 AM to 10:00 PM Monday through Friday), but may not interfere with other scheduled classes in the room. The lab is shared with ECE 352. You can find a schedule of the lab usage on the course web page.

1.4 Goals

This course will give you hands on experience with the ADuC7026 microprocessor and an FPGA-based prototyping environment. In addition to developing design and debugging techniques, you will also learn to write well-structured code to meet design requirements. Software development and debugging will be done using the Keil uVision3 integrated development environment (IDE). Logic design will be accomplished in the Altera Quartus development environment.

1.5 Course Prerequisites

Successful completion of this course depends strongly on successfully completing ECE 352 and ECE 353. Concurrent registration in ECE 353 is permitted (but not encouraged) if ECE 315 is taken in the second half of the semester, and the student is willing and able to work ahead of the normal ECE 353 class.

1.6 Laboratory Schedule

Activity	Section 1	Section 2
Lab #0	1/26	3/15
Lab #1	2/2	3/22
Lab #2	2/9	4/5
Lab #3	2/16	4/12
Lab #4	2/23	4/19
Lab #5 - Project	3/2	4/26
Lab #6 - Project	3/9	5/3

1.7 Computer Usage

All students are required to have a CAE account for PC workstations. If you do not already have such an account, please run *newuser* as soon as possible.

1.8 Required Text

The ECE 315 Lab Manual and board schematics available online.

1.9 Reference

Fredrick Cady, *Microcontrollers and Microcomputers*, 2nd edition, Oxford University Press, 2009
William Hohl, *ARM Assembly Language*, CRC Press, 2009
ADuC7026-Cyclone Board Manual (available online)
ADuC7026 Datasheet (available online)
Other materials available on the course web page.

1.10 Student Responsibility

This course focuses on prototyping basic computer systems using programmable and off-the-shelf components. The emphasis is on developing an understanding of the elements involved in designing hardware interfaces and the software to utilize them. The usual practice in this class is for students to be paired off in groups of two, with each group building one ADuC7026 system. One-person groups are allowed if necessary. Three-person groups are not permitted.

Prior to the first lab meeting, you should read the ADuC7026-Cyclone Board Manual and review the board schematics (both are available online).

Unless otherwise specified, all work in this course is to be your own. Of course, members of a group are permitted to work together on a single solution when appropriate. Sharing information between groups is not permitted. In particular, the use of software, including script, written in whole or part by others is specifically prohibited. Evidence indicating copying of work from others or other unauthorized cooperation will be dealt with as academic misconduct. If in doubt, see your lab instructor or the faculty supervisor.

1.11 Grading

Experiments generally have four parts:

- Prelab Preparation
- In-Lab Work
- Demonstration
- Postlab Report

It is expected that students will review the relevant material concerning the ADuC7026 and come to the lab prepared to complete the exercise in a timely manner. Review of the appropriate sections in the ADuC7026 datasheet is expected. Your instructor will inform you of the due date for any prelab assignments – normally they will be due a couple days before the lab meeting. You should have substantially completed the hardware and software design before coming to the lab. If you come to lab without doing so, it will be more difficult for you to complete the experiment during the lab time. Even more importantly, you will then be completing the work out of lab where instructor assistance is not readily available like it will be in the lab. If you are unable to complete the experiment within the lab period, you have until the beginning of the next lab period to demonstrate it to the lab instructor.

During the lab demonstration, the TA will not only exercise your hardware/software system, but will question the individual members of a team on technical details of that system. This might best

be described as an oral quiz. All group members are expected to be fully knowledgeable about all system details, and their performance in lab will be a factor in their grade.

Postlab Reports are group reports. Both members are fully responsible for the entire content of the report, which must answer all of the questions and provide all of the software, hardware designs, and other items described in an experiment. It is comprehensive, so data and observations collected while in lab, calculation and data processing done after leaving the lab, as well as well-articulated descriptions of what was done, how it was done, and conclusions all should appear. Identify any problems encountered in the lab and how they were solved. Only a single copy of your schematics and source code is required from each team – ensure both team members' names are on it. Print the course number, section number, your name, your partner's name and the experiment number on the report heading.

In addition to the structured experiments, a project is assigned to be accomplished later in the semester. Your instructor will also assign a grade for Quality and Effort to reflect his/her assessment of your contribution to your team's efforts, the overall quality of your work, and the level of professionalism displayed in the lab.

Pre-Lab Deliverables	24%
Post-Lab Deliverables	40%
Project	30%
Quality and Effort	6%
Total	100%

1.12 Laboratory Rules

The rules enforced in this lab are the following:

- All work must be your own.
- All labs are to be demonstrated to the TA during the lab period. If you are unable to do so in the lab period, you must make arrangements with the TA to complete your demonstration before the next lab meeting.
- All report text is to be typed. Calculations may be neatly handwritten.
- Each pre- or post-lab report shall have a title page with your team number, team members' names, the lab experiment number, and the lab experiment title.
- No food or drink is allowed in the lab area.
- Lab benches are to be used for lab work only.
- Printing in the lab is restricted to ECE 315 materials only. Do not print the lab manual on this printer. Please do not abuse this privilege so we can keep the printers available in the lab.

1.13 Course Improvement

Your constructive comments and suggestions on how the course could be improved are always welcomed, and are your opportunity to make the class more interesting and useful to you and future students. Ideas for alternative lab experiments are of particular interest. Please discuss them with your instructor or the faculty supervisor.

2 Lab #0 – Hardware Introduction

2.1 Prelab Assignment

1. There is no formal prelab assignment. You should review the ADuC7026-Cyclone board manual and schematic diagrams available on the web page.

2.2 Set-up

Setup your board on the bench, plug the JTAG pod into the board, and connect its power supply to the power strip. The parallel port connector on the board will be used to program the FPGA, so connect the cable on the bench to the board. Plug the USB cable into the JTAG pod and connect it to the USB extender cable on the bench.

On your computer's D: drive, create a directory "lab0". Under it, create subdirectories "arm" and "fpga". You will use this basic organization for all labs. Be sure to copy your work to your network drive and then delete your files from the D drive before logging out of your computer.

> The D: drive on lab computers will be erased when the machines are reimaged, which will occur at various times during the semester. You must always save your work onto your network drive, or it may be lost.

Download the sample Keil and Quartus projects, and extract to the directories you created. In Quartus, open the project, review the FPGA schematic, and then compile it and program the FPGA. The programming hardware should be set to ByteBlasterMV [LPT1] in JTAG mode. Open the Keil project, review main.s, and then build the project and start the debugger.

> **ADuC7026 Sample Project Notes:**
>
> - The Data Abort and Prefetch Abort exception handlers have been coded as infinite loops. If your code produces one of these conditions, execution will stall in the handler. By halting the code and checking the return address stored in the Link Register (R14), you can determine which instruction caused the problem. Remember that the value stored in R14 was the PC when the exception occurred. Running the program again and single-stepping through the code before that address should illuminate the cause of the exception.
> - If you try to re-use code from ECE 353 (which is not a bad idea), be sure **NOT** to reuse your entire ECE 353 uVision3 project. The ECE 353 projects use the simulator and not the actual hardware. This can make for some interesting debugging when your code doesn't seem to be working correctly on your board, because of course; it is running in the simulator. **Always start your projects using the sample project for ECE 315**. This will use the actual hardware, not the simulator.
> - See the Board Manual for instructions on renaming a uVision project.

Run the Keil program, and you should see a binary count on the LED bar. Halt the program, and open the GPIO4 peripheral window (Peripherals→General Purpose Input/Output→GPIO Port 4 on the keil menu.) Click the output bits and note the effect on the LED bar. The Peripherals menu is an extremely useful tool that you should use to your advantage when debugging your hardware and software.

2.3 Lab Exercise

You will spend the remainder of the lab becoming familiar with the ADuC7026-Cyclone Board, and reacquainting yourself with the ADuC7026. Answer the following questions and perform the

required tasks. Be sure to use all available resources to verify that your answers are correct. Where checkpoints are noted, notify your lab instructor to check your work before proceeding. The ADuC7026-Cyclone Board Manual and Schematic Diagram will be necessary resources in these pursuits. You will also need to refer to the various datasheets available on the web page.

1. Are the LEDs in the LED bar connected to be active high and active low? How can you tell from the schematic diagram?
2. Draw a schematic diagram for just the circuitry involved with driving the leftmost LED. Include the pin numbers of all parts involved, and use proper component symbols. What name is assigned to this FPGA pin in Quartus?

Modify your FPGA design to connect the right pushbutton switch to drive the leftmost LED in the LED bar display.

3. Are the pushbutton switches on the board active-high or active-low? What voltage do you expect to be at the FPGA pin when the switch is on (closed) and off (open)? (An active-high switch will generate a logic 1 when it is closed.)
4. Draw a schematic diagram showing the complete circuit used for the left pushbutton switch including its connection to the FPGA. Include the pin numbers of all parts involved, and use proper component symbols. What name is assigned to this FPGA pin?
5. Suppose that you mistakenly set the FPGA pin connected to the pushbutton switch to be an output. What is the worst case current that could flow through FPGA pin in both directions, and under what conditions will it occur? How do these values compare to the current ratings for an FPGA pin?
6. In the Board Manual section *FPGA Pin Naming*, it specifically states not to configure the FPGA with P0.0 or P0.3 as outputs. What problems could arise if the FPGA drove these pins?

Modify your FPGA design to connect the DIP switches to GPIO port 3. Modify the ARM7 code to configure GPIO Port 3 as input, then continuously read Port3 and write that value to Port 4 in the main loop. Reprogram the ADuC and FPGA, and run your code.

Checkpoint: When you are satisfied that your code is working, have your instructor check your work.

7. Are the DIP switches on the board active-high or active-low?
8. Draw a schematic diagram showing the complete circuit used for the leftmost DIP switch including its connection to the FPGA. (Note that all 16-pin resistor networks (RNx) on the schematic diagram consist of 8 resistors connected between pins 1-16, 2-15, etc – just show them on your diagram as individual resistors.)

Modify your FPGA design to connect the rotary encoder outputs to the 3 most significant bits of the LED bar.

Checkpoint: Have your instructor check your work.

9. What are the pin names used by the FPGA for the rotary encoder outputs?
10. How are the A and B outputs related when the encoder is rotated clockwise? Counter-clockwise? Draw a diagram showing one complete cycle of A and B, such that clockwise rotation corresponds to moving left-to-right on the diagram.
11. What information can you obtain from the A and B outputs, and how would you do it? (You do not have to write code, just explain how.)
12. What is the third encoder output for?

Modify your FPGA design to connect GPIO port 4 to the segment lines of the multiplexed 4-digit display (segment A to bit 0, segment B to bit 1, etc.), and connect the upper nibble of GPIO port 1 to the digit lines (bit 4 to the rightmost digit and bit 7 to the left most digit). Use the Peripheral windows in Keil to operate GPIO ports 1 and 4 to test your connections.

Checkpoint: Have your instructor check your work.

13. What are the FPGA pin numbers and assigned names for segment A, the decimal point, and the leftmost digit enable?
14. Are the digit enable and segment lines active-high or active-low?

Modify your FPGA design to hardwire the rightmost digit on and the other three digits off. GPIO port 4 will remain connected to the segment lines of the multiplexed 4-digit display (segment A to bit 0, segment B to bit 1, etc.). Connect GPIO port 3 to the DIP switches, and GPIO port 2 bits 1:0 to the pushbutton switches (left switch to bit 1). Modify your Keil project to continuously read the DIP switches and place the corresponding hexadecimal character on the rightmost LED display. If the right pushbutton switch is pressed, display the value of the most significant nibble of the DIP switches; otherwise display the value of the least significant nibble. If the left pushbutton is pressed, you should display the 1's-complement of the dip switch value instead of the actual value. Delete any unused code in your source file from previous exercises.

15. Submit your ***main.s*** source file with the post-lab deliverables. Your code must comply with the course documentation standards and be reasonably efficient.

This completes this lab exercise.

2.4 Post-Lab Deliverables

Provide hard copies of the following;

1. Answers to the questions in the lab exercise.
2. Your ***main.s*** file from #15.

Grading Criteria

The deliverables will be graded according the following criteria;

1. Correctness.
2. Completeness.
3. Software functionality.
4. Software organization and efficiency.
5. Conformance with software documentation standards.

3 Lab #1 – Multiplexed LED Display

3.1 Prelab Assignment

Review the ADuC7026-Cyclone board manual and schematic diagrams available on the web page. Answer the following questions and turn in at the start of the first lab meeting. All required documentation (manuals, datasheets, etc.) is available on the ECE 315 web page.

1. Draw a schematic diagram showing the segment A drive line and all 4 digit drive lines. You diagram should include the connections to the FPGA, all relevant board components, and the individual LEDs contained within the display itself. Explain how the multiplexed display operates, including the signals are required to turn on LED segments. (Note that the 8-pin resistor networks (RNx) on the schematic diagram all consist of 4 resistors connected 1-8, 2-7, etc – show them on your diagram as individual resistors. Also, the FETs used as digit drivers are N-channel devices that are used as switches – when the gate is at 0V they are off (hi-Z from drain to source), and when the gate is at 3.3V they are on (lo-Z from drain to source).
2. What is the expected current in an LED display segment when it is on? (You may assume the FETs drain-to-source resistance (R_{DSon}) is 0Ω when they are on.) Show your work, and describe how all values were arrived at.

3.2 In-Lab Assignment

This lab will introduce a hardware timer interrupt, and how to use the interrupt handler to control a multiplexed LED display.

1. Establish a ~8.0 kHz timer interrupt using timer 1, and configure the interrupt so that it is handled in IRQ mode. Note that a skeletal interrupt handler for IRQ mode has already been written in *exceptions.s* of the sample project. Along with your code to drive the displays, you will also need to add code to it to clear the timer interrupt, otherwise you will immediately reenter IRQ mode because the interrupt is still pending. The return from IRQ mode back to user mode is accomplished by a specific instruction, **SUBS PC, LR, #4**. This instruction will correct the address in LR, which was the PC at the time of the interrupt so it is 8 bytes ahead of the instruction before the interrupt (and so 4 bytes ahead of the next instruction that should be executed). Since the "S" is used and PC is the destination, this instruction also causes the SPSR register is copied into CPSR, restoring the interrupted mode's state. You MUST ensure that your handler code does not corrupt any non-banked registers.

Notes on Using Interrupts:

- You ALWAYS need to be sure that your ISR clears the interrupt source. If you don't, the interrupt request will still be pending when the processor exits the ISR, so it will immediately re-enter the ISR and effectively lock up the processor. This can result in a condition where even the JTAG debugger pod cannot break into the processor's execution. If this happens, see your lab instructor for help fixing it.
- If you call a subroutine in your ISR, you MUST save/restore R14 in the ISR or it will be corrupted by the subroutine call!

2. Use the GPIO Ports 1 and 4 to control the 7-segment displays, as done in lab #0. Use the upper nibble of Port 1. There are 7 segment drive lines, and 4 digit drive lines. (You can leave the decimal point off at all times by simply driving a constant 0 on its segment line.)

When using a multiplexed display such as this, we drive the segment pattern for one of the digit values and turn on that digit for a period of time, then switch the segment pattern to the other digit and drive it for a period of time, and so on. If we do this quickly enough, all displays will appear to be on continuously. The data for the 4 digits should be stored in a 4-byte array named *DisplayData*. In each invocation of the timer interrupt handler, display the hexadecimal value of the appropriate element of *DisplayData*. In the ISR, you will need to keep track of which digit you turned on last time to know which one to turn on in the current invocation. The display should show the characters 0-9 and A-F to represent the hexadecimal value of the lower nibbles of *DisplayData*. (You should efficiently translate the nibble value to the required segment value – using a look-up table should be an obvious choice here.)

3. In your main program loop, use GPIO Port 2 to continuously read the DIP switches and then set the value of each element of *DisplayData* as follows. The rightmost pair of digits should always display the upper and lower nibble of the DIP switch value, while the leftmost digits display the 2's-complement of the DIP switch value. The main program loop should do nothing else.
4. Using the oscilloscope's digital inputs, configure the oscilloscope to show the 4 digit drive lines on D11-D8 (DIGIT4 through DIGIT1, respectively) and the 7 segment drive lines on D6-D0 (SEG_G through SEG_A, respectively). Capture a trace showing a complete multiplexing cycle starting with DIGIT1. Annotate your trace with the actual characters being displayed when the trace was captured.
5. Determine the current flowing in one of the LED segments. Since we don't have direct access to measure the current, we will need to do it indirectly. To do this, use the oscilloscope's analog probes to measure the voltage across one of the current-limiting resistors in the segment lines. (The scope can automatically show the difference between the two inputs – a handy capability here.) Capture a trace showing the voltage waveforms across the resistor in both the illuminated and off conditions.
6. **Bonus:** Use the idea of pulse width modulation to control the intensity of the display in software using the same 8.0 kHz timer interrupt. Use only the four least significant bits of the DIP switches to set the brightness of the display. The brightness should vary from 0/8 (off) to 15/16 (maximum) of the full intensity that could be achieved. As an example, if the three least significant DIP switches had the value 10 (2_1010), the displays should be at 10/16 of the maximum brightness that they achieved when driving them in #2 above.
    a. Capture traces as per #4 above showing the operation of the display signals at with the switches set to 2_0011 and 2_1110.

In this experiment, we have used the FPGA to serve only as a connection mechanism. We could have chosen to use the FPGA to implement a controller that would autonomously drive the LED displays and offload all of the workload from the microprocessor, so all we had to do in software was write the 8-bit digit values to the controller for it to display and send it an intensity setting as needed.

3.3 Post-Lab Deliverables

Provide hard copies of the following;

3. A brief description of what was done, how it was done, and any problems encountered.
4. A paper copy of the annotated oscilloscope trace(s).
5. Answers to the following questions.
    a. What is the measured peak current flowing in an illuminated display segment? How does this compare to what you had calculated in the prelab assignment? Can you explain any differences? What is the average current for an illuminated segment? How does this relate to the duty cycle, refresh rate, and number of multiplexed digits?
6. Your FPGA schematics.
7. Your source code (only include files that you have modified, do not include files that you made no changes to).

Submit the following electronically as a single ZIP archive. The top level directory shall be “lab1”, with “arm” and “fpga” directories underneath containing the Keil and Quartus projects, respectively.

1. Your Keil project.
2. Your Quartus project.

3.4 Grading Criteria

The deliverables will be graded according the following criteria;

6. Functionality.
7. Software organization and efficiency.
8. Conformance with software documentation standards.
9. Hardware design correctness and efficiency.

4 Lab #2 – Keypad Interfacing

4.1 Pre-Lab Assignment

Review the process of scanning a matrix of switches, and inspect the keypad connections on the board schematics. Answer the following questions.

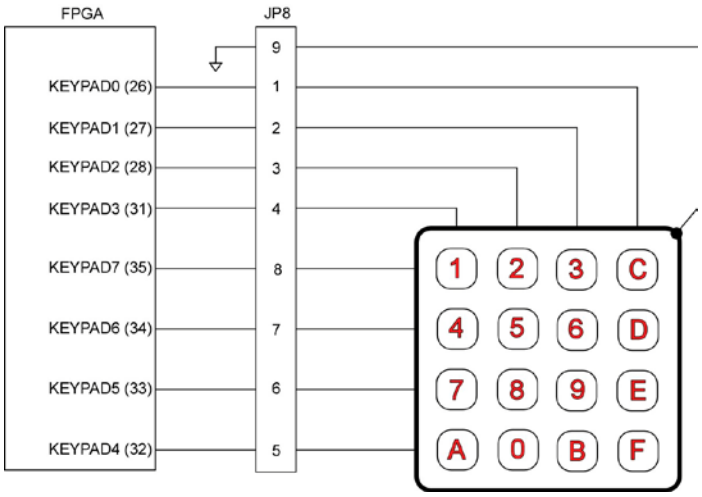
1. In this lab, we will **drive the columns** of the display, and **read the rows**. How do we scan the keypad? How do we know if a switch is pressed? Briefly describe the process.
2. What is an open-drain driver? Why do we use them for driving lines low instead of a standard totem-pole driver?
3. Describe how you will *efficiently* perform debouncing of the switches and detection of new switch events. (The debouncing requirements can be found later in this assignment.) You should be able to accomplish this by using only logical operations and two half-words of memory.

Note that the keyboard lines will not be driven back to a logic 1 level after they are driven low. When you set the open-drain driver's input to a 1, the output goes to a hi-Z state and then that line (and any lines connected to it by closed switches) is pulled up to a logic 1 level by a 10kΩ resistor. You will find that there is a significant amount of parasitic capacitance due to the long conductors and the FPGA input capacitance, so the lines will take a *relatively* long time to charge back to a logic 1. (Relatively long compared to the ADuC7026 processor speed, that is.) You will need to be concerned about this in your scanning algorithm – you will want to release the drivers to a hi-Z state as soon as you have read the inputs to ensure that the lines have as much time to charge as possible before you read the input again.

4.2 Assignment

Reading a matrix-connected keypad is more complicated than reading a simple switch. Since the connections to the keypad switches are shared, we need to scan through the rows and columns looking for closed switches. In this lab, you will perform scanning of the keypad, and then use the 7-segment displays to continuously show the last 4 keys that were pressed.

To do the keypad scanning, you will need to make use of the FPGA's ability to implement open drain drivers. The keypad should be connected to your FPGA as shown below. (Pin 9 is an electrostatic-discharge protection ground connected to a conductive mesh under the keypad surface.)



When using the keypad, please observe the following;

- Plug the keypad into JP8 on the ADuC7026-Cyclone board. The cable has a dot on pin 1.
- Only use open-drain buffers to drive the keypad lines. These can be found in Quartus under ***primitives→buffer→opndrn***.
- There are already pull-up resistors installed on the board for all pins of the keypad interfaces.
- As each key press is detected, display it on the rightmost 7-segment display, and display the three previous key presses on the remaining 7-segment displays. (I.e., the key press values should appear to move from right to left across the display as new key presses are detected.) The display should show the value as 0-9, A-F to match the key label.

You can ignore phantom key issues, i.e. assume any key seen as pressed is actually pressed. You must debounce and check all keys for events each time. Multiple simultaneous key presses should be handled correctly; the order in which they are displayed is arbitrary. Scan the keypad every 10ms using the timer 0 interrupt mapped to FIQ mode. Assume that any switch bounce will be less than 10ms, and consider a key pressed when it is seen pressed two times in a row.

The LED display is to be driven by the timer 1 interrupt, as done previously in Lab #1. The connections to the keypad should be made using the ADuC7026 GPIO port 2. Connect KEYPAD0 to P2.0, KEYPAD1 to P2.1, etc.

Once your keypad scanning is working, use an analog scope probe to observe one of the driven lines. Capture a trace showing the line being pulled low and then released, with no switches pressed. You should see a very distinct RC charge curve when the line is released. Using the trace and what you know about the circuit, estimate the total capacitance on the line. (Using the scope cursors to find key time/voltage points is much better than trying to measure things on the trace print-out!)

Bonus: Add a repeat capability to the keys, so that if a key is held for more than 1s, a key press event is then generated approximately once a second until it is released.

4.3 Post-Lab Deliverables

Provide hard copies of the following;

1. A brief description of what was done, how it was done, and any problems encountered.
2. A hard copy of the scope capture and your capacitance estimate. Explain how you calculated the capacitance value.
3. Your source code (only include files that you have modified, do not include files that you made no changes to).

Submit the following electronically as a single ZIP archive. The top level directory shall be “lab2”, with “arm” and “fpga” directories underneath containing the Keil and Quartus projects, respectively.

1. Your Keil project
2. Your Quartus project

4.4 Grading Criteria

The deliverables will be graded according the following criteria;

1. Functionality
2. Software organization and efficiency
3. Conformance with software documentation standards

5 Lab #3 – External Memory Interface

5.1 Pre-Lab Assignment

The external memory interface on the ADuC7026 permits us to add external memory or memory-mapped I/O devices to an ADuC7026 system. A diagram showing how basic memory-mapped I/O ports would be constructed is available on the course web page at http://eceserv0.ece.wisc.edu/~morrow/ECE315/ARM7-FPGA/aduc7026_timing/timing.html . You will also find diagrams showing the basic signal relationships and timing adjustments for the external memory interface there. You should review these materials and the relevant sections of the ADuC7026 datasheet prior to lab.

Your first task is to implement (in the FPGA) two 8-bit memory-mapped input ports and two 8-bit memory-mapped output ports. The ports are to be mapped into the ARM memory space so that they are accessible in external memory region 0. Configure the memory region for 8-bit operation with one wait state in the read/write strobe interval (XMxPAR[3:0] and XMxPAR[7:4], respectively). Note that 1 wait state means that your read/write strobe should be 2 clocks in length. Connect the ports as follows;

Input port 0 (even addresses) – Rotary encoder A to bit 7, B to bit 6, all other bits read as 0
Input port 1 (odd addresses) – DIP switch[7:0]
Output port 0 (even addresses) – LED Bar[7:0]
Output port 1 (odd addresses) – LED Bar[9:8] to least significant bits

Do minimal decoding for the ports, so that they occupy the entire memory region. Note that you will need to connect to the ADx lines using bidirectional pins, and build the required demultiplexing and decoding circuitry in the FPGA. Design the circuitry as asynchronous logic, i.e. the flip-flop clock inputs in the output port registers should be derived from the memory control signals. In keeping with good engineering practice, your hardware design should be as simple as possible while meeting the design requirements.

In ECE 353, we discussed the use of medium-scale integration (MSI) devices to construct I/O ports. In the Altera Quartus software, there are models of many of these MSI devices that can be used to create your ports without having to draw every low-level gate. (You can find them in the **Symbol** tool under the ***others\maxplus2*** directory.) Of particular value in this experiment are an 8-bit latch (74373), an 8-bit flip-flop (74374), an 8-bit buffer (74244) and a 3-to-8 decoder (74138). (Note that if you double-click one of the symbols in your schematic, Quartus will open a schematic of the symbol so you can review how it is constructed. You can also find datasheets for the actual MSI parts on the course web page.) **The Quartus schematic diagram has been started for you, and can be downloaded from the course web page.**

Submit the following items:

1. An explanation of how your decoding logic works.
2. A paper copy of your completed schematic diagram for the input/output ports.

5.2 In-Lab Assignment

Once you have constructed the ports as required above and tested them, use the oscilloscope digital inputs to capture a read bus cycle and a write bus cycle. Clearly annotate the traces showing the signal names, and show where the wait state was inserted. Your traces must include the signals /MSx, /WS, /RS, A16, AE, and AD[7:0]. (You can display the AD[7:0] lines as a group if the scope that you are are using supports groups.) Specifically annotate the traces to show when the address became valid, when the data became valid, and where the wait state is inserted.

You are to use the timer 1 interrupt to read the value of the rotary encoder's A/B outputs at an approximately 5kHz rate. (Documentation on the rotary encoder is available on the course web page.) Based on the value that you read, determine if the encoder has moved since the last time you checked it, and if it has moved, determine which direction the encoder has moved. All but one LED bar segment should be on. Use the encoder movement information to cause the non-illuminated LED bar segment to move in step with the rotary encoder. If the rotary encoder moves clockwise, the LED should move to the right. If the rotary encoder moves counter-clockwise, the LED should move to the left. If the illuminated segment gets to the end of the bar, it should appear at the other end.

5.3 Lab Deliverables

Provide hard copies of the following;

1. A brief description of what was done, how it was done, and any problems encountered, including;
    a. Describe your algorithms for determining the direction of the rotary encoder.
    b. Describe your algorithm for manipulating the LED segment.
    c. Describe what the effects would be if you configured the external memory region for 16-bit operation instead of 8-bit operation. Draw a memory map of the first 16 bytes of the memory region where your ports exist. (You might want to test your answer in the lab.)
2. Your final FPGA schematics.
3. Your source code (only include files you have modified, do not include files that you made no changes to).
4. Your annotated scope captures showing a read and write bus cycle.

Submit the following electronically as a single ZIP archive. The top level directory shall be "lab3", with "arm" and "fpga" directories underneath containing the Keil and Quartus projects, respectively.

1. Your Keil project.
2. Your Quartus project.

5.4 Grading Criteria

The deliverables will be graded according the following criteria;

1. Functionality.
2. Software organization and efficiency.
3. Conformance with software documentation standards.
4. Hardware design correctness and efficiency.

6 Lab #4 – Stepper Motor Operation

6.1 Pre-Lab Assignment

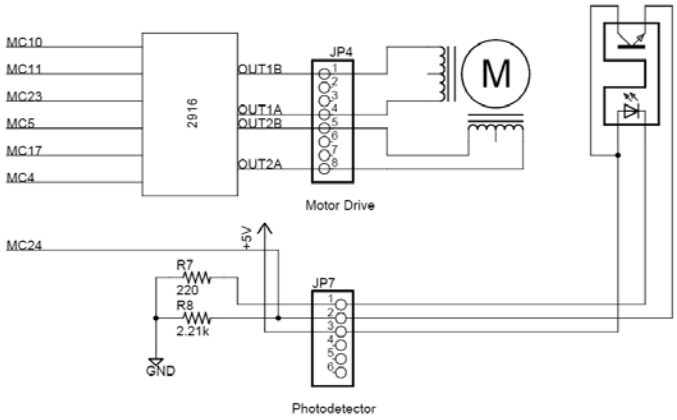
You may not be familiar with a stepper motor. They are very widely used in applications that require precise positioning without the need for any position sensing/feedback. Their primary disadvantage is their relatively low torque. An explanation of how a stepper motor works in general is available on the course web page (**HSI Stepper Motor Theory**). Read at least pages 1 through 4. Like all motors, a stepper motor consists of a rotor (the part that rotates) and a stator (the part that's stationary). In a stepper motor, the rotor features two gear-like cylinders that are turned one-half tooth spacing with respect to one other. One gear is a permanent magnetic north (N) pole and the other is the south (S) pole. The stator is also gear-like, but it has a different number of teeth than the rotor. In the stator, each tooth is an electromagnet and can be made into a north or south pole by driving a current through a coil of wire wound around it.

For this lab, we will use a Vexta® Stepper motor which will operate at up to 200 or 400 steps per revolution in a full or half step mode, respectively. Review the Motor Control Board schematic diagram, the Allegro 2916 datasheet and the Vextra® motor specifications, all available on the course web page. The Allegro 2916 consists of two independent H-bridge sections that can control both the direction and amount of current flowing through each of the stepper motor's coils. The current in the coil is limited to a maximum of approximately 750mA. Each H-bridge section has three inputs that are connected to FPGA I/O pins;

- PHASEx (MC10, MC5) – this pin controls the direction of current through the coil.
- I1-x, I0-x (MC11, MC23, MC17, MC4) – these pins control the amount of current that flows in the coil. The values on the pins are interpreted as shown below. Unless you are doing the bonus part of the lab, the coils should either be off or at 100% current. (Note that if you operate at partial current, you will be able to hear the motor "singing" due to the PWM current control. This is not a problem.)

I1-x	I0-x	Percentage of Maximum Current
0	0	100%
0	1	67%
1	0	33%
1	1	0%

The stepper motor also has an optical sensor output connected to FPGA pin MC24 that will detect when the bar mounted on the motor shaft passes through it. Although not used in this lab, it allows you to determine an initial position for the motor shaft. The motor and sensor connections are shown below. Note that although the Vexta motor can be operated as a unipolar motor, we have connected it for operation as a bipolar motor.



Submit the following;

1. Prepare a table showing the required signals to the PHASE1/2 and I1-1/2, I0-1/2 inputs to the Allego 2916 to implement full steps using “one phase on” stepping.
2. Prepare a table showing the required signals to the PHASE1/2 and I1-1/2, I0-1/2 inputs to the Allego 2916 to implement full steps using “two phase on” stepping.
3. Prepare a table showing the required signals to the PHASE1/2 and I1-1/2, I0-1/2 inputs to the Allego 2916 to implement half steps.
4. Write a subroutine ***Step*** that will take two parameters in R0 and R1. If R0 is positive, cause the stepper motor to move one step (or half-step) in a clockwise direction. If R0 is negative, drive the stepper motor to move one step in a counter-clockwise direction. If R0 is zero, do nothing. (Just make an assumption as to which way the motor will rotate – you can test this in the lab and modify your subroutine if needed.) You should use only a single look-up table and a single index variable in your subroutine. The two least significant bits in R1 will be used to set the stepping mode in use, as shown below. If the parameter in R1 is different than the previous invocation, you may start the new step sequence at the start of your look-up table.
    a. R9 = 2_00 – use “one phase on” stepping.
    b. R9 = 2_01 – use “two phase on” stepping.
    c. R9 = 2_10 – use half-steps.
    d. R9 = 2_11 – see BONUS item.

Submit the following;

1. Your subroutine Step.
2. On a separate sheet, submit a proposal for the project that you complete in labs 5 and 6 (and then demonstrate during lab 6). Your project must meet at least the following requirements outlined in the Lab #5-6 assignment.
    a. The proposal is to include a one-page narrative description of what the project is and how it will work. Ensure that you identify specifically where in the project each of the requirements listed in Lab #5-6 will be met. Include a functional block diagram showing the project’s logical organization in sufficient detail to permit a reasonable evaluation of the project scope and complexity.

6.2 In-Lab Assignment

IMPORTANT: When connecting the Motor Control board to your ADuC7026-Cyclone board, always observe the following sequence;

1. Disconnect all power from both boards. (Just unplug the Motor Control board power supply from AC power, do not disconnect it from the Motor Control board.)
2. Connect the Motor Control board to the ADuC7026-Cyclone board.
3. Connect the stepper motor to the Motor Control board JP4 and JP7. The connectors are keyed to prevent improper orientation.
4. Connect power to the ADuC7026-Cyclone board.
5. Connect the Motor Control 12V power supply to AC power.

When powering down the boards, reverse the above order.

1. Establish a 5.0 kHz timer interrupt using timer 1, and configure the interrupt so that it is handled in IRQ mode. This ISR will manage the LED 7-segment displays, as in Lab #1, as well as debounce the pushbutton PB1 and issue events. In addition, your ISR will now manage the movement of the stepper motor. The ISR will cause the stepper motor to move in response to the value of a global memory halfword ***StepCount*** that is set by the main program. The value in ***StepCount*** is assumed to represent a clockwise direction for positive values and a counter-clockwise direction for negative values. Your ISR will decrement or increment, as appropriate, the count for each step taken until it reaches 0. You may call ***Step*** no more than once per interrupt. Operate the motor at 200 steps (or half_steps) per second. Use the global memory byte variable ***StepMode*** as the R9 parameter to ***Step***. Your ISR should continuously update the LED display with the hexadecimal value of ***StepCount***.
2. Write a main program that will check for pushbutton switch events; when a PB1 event is detected, you will read the value on the DIP switches. The least significant 6-bits shall be interpreted it as a 2's-complement number (-32→+31), and added to a global memory word ***StepCount***. The most significant two bits shall be stored as a value (0-3) in the variable ***StepMode***.

Once you have demonstrated your working program, modify your program to change the step rate to 800 steps per second. Run the program and observe the operation of the motor.

Bonus (5 points):
Modify your program so that it implements micro-stepping when the Step subroutine receives parameter R9 equal to 2_11. (You can find information on micro-stepping on the web.) In the ISR, you should operate at 200micro-steps per second. Your micro-steps should be as small as the motor controller allows. Demonstrate your finished program to your TA.

6.3 Lab Deliverables

Provide hard copies of the following;

1. A brief description of what was done, how it was done, and any problems encountered, including;
    a. What happened when the step rate was increased to 800 steps/s? Why do you think that it happened?
    b. Describe how your Step subroutine works, especially how you control the stepping mode.
    c. If you did the bonus item, explain how your micro-stepping worked.
2. Your final FPGA schematics.
3. Your source code (only include files you have modified, do not include files that you made no changes to).

Submit the following electronically as a single ZIP archive. The top level directory shall be "Lab 5", with "arm" and "fpga" directories underneath containing the Keil and Quartus projects, respectively.

1. Your Keil project.
2. Your Quartus project.

6.4 Grading Criteria

The deliverables will be graded according the following criteria;

1. Functionality.
2. Software organization and efficiency.
3. Conformance with software documentation standards.
4. Hardware design correctness and efficiency.

7 Lab #5-6 – Project

7.1 Assignment

The idea is to build a device that performs some useful or interesting function, using the ADuC7026 and the Cyclone FPGA. You can add additional hardware to the project as desired. The project requirements are set up so that you can choose the level of difficulty of your project and the corresponding maximum attainable grade, as noted below. Note that a higher difficulty project must also meet all requirements of the lower difficulty projects.

Max project grade = 70:

1. All digital I/O ports are to be created in the FPGA and accessed via the external memory interface. You may not use the ADuC7026 GPIO ports as GPIO.
2. Project functionality can be limited to use of only the existing switch/display interface elements.

Max project grade = 80:

1. You must use analog input and/or output in some way in your project.

Max project grade = 90:

1. You must create at least one peripheral device in the FPGA that interrupts the ADuC7026 for service. An example would be a keypad scanning circuit that only interrupts the processor when a key is pressed, and returns that key number when serviced. Basically, the peripheral you create must offload as much of the processing requirements as possible from the CPU, and use interrupts to signal the CPU when service is required. The peripheral device design must be designed such that the interrupt request is cleared by the CPU writing to some location in the external memory space.

Max project grade = 100:

1. Implement all external memory interface logic in FPGA as a synchronous design using the FPGA's CLK1 clock input. Note that this is a 20MHz clock that is unrelated to (and slower than) the ADuC7026 clock, so you will need to ensure that you configure the ADuC7026 external memory interface timing to ensure that it can work. To demonstrate the correctness of your implementation, one read-only port register must be implemented such that reading it will return the number of times (modulo-256) that the external port registers have been written to. Ensure that lengthy bus cycles (i.e. many wait states) do not count as more than one write. The data that is actually stored on a write should be the data seen immediately before the rising edge of the write strobe. Remember to synchronize all control signals from the ADuC7026 to your logic.

If you have other ideas that don't quite fit this model, please discuss them with your lab instructor to see if they can be accommodated.

7.2 Sample Project Ideas

The most satisfying projects are generally those that are very interactive. Some ideas that may inspire a project idea are listed below.

- Using the audio output to generate sounds to create a musical instrument.
- Build a "clapper" that responds to noise.
- Games often make for good projects.
- Don't be afraid to add a little hardware, especially sensors (i.e. phototransistors, etc.). Talk to your instructor if you're not sure about feasibility.

-

For additional display capability, one interesting idea is to use two DAC channels to drive the oscilloscope in XY mode. This would permit you to generate line drawings on the oscilloscope. By using the persistence feature of the oscilloscope, you can change the brightness of the line by controlling what percentage of the time the beam spends there. Some possible uses of this might be an electronic Etch-a-Sketch or a display for a game with geometric shapes.

7.3 Important Notes on Using Analog Inputs and Outputs

If you choose to use the analog-to-digital converter (ADC) in your project, there is also an important device errata that you need to be aware of regarding the triggering of ADC conversion. The work around is to trigger a single conversion in software (i.e. a timer ISR), then disable further conversions. You can find the device errata sheet on the course web page. Note that Analog Devices refers to its errata as “Silicon Anomalies”.

If you are using any of the analog inputs or outputs with the internal voltage reference, be sure to set the REFCON register to connect the internal voltage reference to the pin on the ADuC7026 so that it will have a filter capacitor. If you do not do this, the internal voltage reference can become unstable and oscillate, which will cause the DAC outputs and ADC inputs to be unusable.

Note that the DAC data registers must be written as words, not half words. The simulator will accept half-word values, but not the actual hardware.

7.4 Generating Random Numbers

A common need in a project, especially a game, is to be able to generate one or more random numbers. By definition, a digital system cannot generate a random number, so if we need a series of random numbers we will use a pseudo-random sequence generator. The numbers in the pseudo-random sequence are randomly distributed, but the sequence will repeat eventually, and it will be exactly the same each time we generate it. In a digital system, we basically need to decide just how random we really need to be. If the pseudo-random sequence is sufficiently long, and we can start from a point that is different each time we run a program, the results will appear random to the user. So, we have two basic problems – how to generate an initial seed value, and then how to generate a pseudo-random sequence using that seed value.

To generate a random seed value, we can either try to observe some random phenomenon, or use an unpredictable event. One random phenomenon is the electrical noise that is present on all systems. In the lab, measuring the voltage present on an unconnected ADC input may provide sufficient noise. If the project uses human intervention to start (such as pressing a button to start a game), the instantaneous value of a counter that is continuously clocked by a signal such as the CPU clock or the FPGA clock will be essentially random. By reading the counter value at the time that the button is pressed will effectively generate a random seed value.

The initial random seed value is then used to initialize a pseudo-random sequence generator. One relative easy way to generate a pseudo-random sequence is to use a linear feedback shift register (LFSR). In fact, the ADuC7026 uses an LFSR model for the watchdog timer’s secure clear function. By using a larger LFSR, we can generate longer sequences. There are numerous references on the web regarding LFSRs. One of the more easily readable ones is http://www.ee.ualberta.ca/~elliott/ee552/studentAppNotes/1999f/Drivers_Ed/lfsr.html .

7.5 Deliverables

Provide hard copies of the following;

1. A brief description of what was done, how it was done, and any problems encountered, including;
    a. Identify specifically which difficulty level your project meets, and describe how your design met each of the project requirements.
    b. Any interesting, complex, or unusual elements of your design.
2. Your FPGA schematics.
3. Your source code (only include files you have modified, do not include files that you made no changes to).
4. Each team member will complete the confidential assessment form on the next page and turn in to the lab instructor.

Submit the following electronically as a single ZIP archive. The top level directory shall be “lab6”, with “arm” and “fpga” directories underneath containing the Keil and Quartus projects, respectively.

1. Your Keil project.
2. Your Quartus project.

7.6 Grading Criteria

The deliverables will be graded according the following criteria;

1. Project complexity.
2. Functionality.
3. Creativity.
4. Software organization and efficiency.
5. Conformance with software documentation standards.
6. Hardware design correctness and efficiency.

ECE 315 Project – Confidential Team Assessment

Name: _____

Rank as a percentage the relative contributions of your team's members to the success of your project. (The total must be 100%.) Also, identify each person's major responsibilities, what specific hardware and software they were responsible for, and write comments to explain your assessment.

Team Member	Percent of Effort
you	

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

8 Lab Equipment Reference

8.1 Quick Guide to the HP 54622D Oscilloscope

Generally, an oscilloscope is the instrument to use when you need high resolution or view the actual level of signals. Therefore, you should use a scope when you need to see small voltage excursions on your signals or when you need to resolve small time intervals.

The HP 54622D is a two-channel, 100MHz, sampling oscilloscope which is capable of making and displaying two analog measurements simultaneously at 100MHz. It also has a 16-bit digital logic analyzer built in, although we will not use it in this course. As with all other oscilloscopes, the HP 54622D normally presents data with time along the horizontal axis, and voltage amplitude along the vertical axis.

To use the HP 54622D, you must ground the probe(s) to the reference voltage, which is usually a ground (0V) in your circuit, by connecting the black alligator clip of the probe to the reference voltage of your circuit. When you press the *AutoScale* button, the HP 54622D will automatically adjust its horizontal and vertical scale to display one or two periods of periodic signals. (Be aware that using *AutoScale* when no signal is present will result in a high gain setting on the vertical scales, which may make low amplitude noise appear to be a valid signal if you aren't paying attention the volts/div setting.) You may adjust the horizontal (time) scale by turning the large knob under the *Horizontal* menu of the oscilloscope. Since the HP 54622D has two channels, you must make a separate adjustment for each channel. To adjust each vertical (voltage) scale manually, turn the *Volt/Div* knob under the *Analog* menu.

HP 54622D's have all sorts of automated measuring features that you can use to simplify your work. When the desired wave form is displayed, press one of two automated measurements buttons (*Quick Meas* and *Cursors*) under the *Measure* menu. An array of options will appear at the bottom of the screen. Press a corresponding gray button under the desired option to make an appropriate automated measurement.

8.2 The HP Word 54600 Toolbar

The *Word 54600 Toolbar* is a plug-in to Microsoft Word that enables a communication link between the Windows NT workstation and the HP 54645A digital oscilloscope. With this software, you can quickly and easily transfer screen images, waveform data and instrument setups. When using this application, you must ensure that the serial port switch is in the ***Oscilloscope*** position (***C***).

Starting the Word 54600 Toolbar

1. Login to the Windows NT workstation, and turn on the HP 54622D digital oscilloscope.
2. Double-click on the *Word 54600 Toolbar* icon on your NT workstation. Word will be started with a floating toolbar as shown in Figure 3-1.
3. Online help is available by pressing the rightmost button on the toolbar.

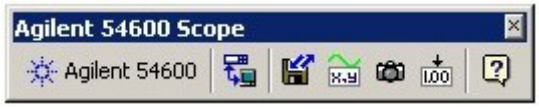


Figure 8-1 The Word 54600 Toolbar.

Connecting to the Oscilloscope
Before you can communicate with the oscilloscope, you must establish a serial connection to it.

1. Ensure the serial port switch is in the Oscilloscope position.
2. Click the ***Connect to Scope*** button.
3. Click ***Search***.

The oscilloscope should be found on COM2.

Capturing Screen Images
You can transfer bitmap pictures of the oscilloscope screen to your PC for viewing, printing, and storage in various formats. You can directly insert these images into Word (or save as graphics files) for documenting and presenting your work.

1. Ensure that you are connected to the oscilloscope. (See above.)
2. Click the *Get Screen Image* button.
3. Select the desired destination and download the image.

Capturing Waveform Data
The digital oscilloscope's time and amplitude data can be transferred to the PC. This is different from the image transfer since you can manipulate the data using your local host NT workstation. You can review your data and save it in a variety of formats or you can easily copy and paste the data into other applications for further analysis.

1. Click the *Get Waveform Data* button.
2. Select the number of data points to be downloaded.

Instrument Setups
You can easily transfer setups to your PC and later send them back to the instrument to conveniently restore a previous setup. All instrument front panel settings, including timebase settings, vertical settings, and trigger settings, can be copied from the oscilloscope to the NT workstation. The setup can be saved in a file and can then be sent back to the instrument to restore the saved settings at a later time.

1. Click the *Save / Load Scope Settings* to invoke the instrument setup window.