Lab Overview

In this lab assignment, you will do the following:

- Add RS-232 communication to the hardware developed in Labs #1 & #2.
- Replace the C501 processor with an Atmel AT89C51RC2 processor.
- Learn how to use the Atmel FLIP utility to program the processor flash memory.
- Learn how to configure the 8051 for serial communication and how to write serial device drivers.
- Learn how to use internal XRAM and external XRAM (using the NVSRAM).
- Begin learning how to use the SDCC compiler and Eclipse IDE to develop C programs.

The signature due date for this lab assignment is **Tuesday, October 27, 2009**.
The submission due date for this lab is **5:30pm Wednesday, October 28, 2009**.

This lab is weighted as 20% of your course grade.

Finish Lab #3 early, to give you more time for Lab #4 and your final project.

Lab Details

1. Refer to Lab #1 for comments regarding layout considerations, labeling, wiring, etc. All signals and pin numbers on all ICs must be labeled.
2. Determine how to program the Atmel AT89C51RC2 using the EMP-100 programmer to set the lowest security levels, to enable reset at address 0000h, and to enable the XRAM. See the programming guide notes, that will be available on the course web site.
3. Replace the C501 on your board with the Atmel AT89C51RC2. Verify that the oscillator circuit and run-time reset circuit work correctly, like you did in Lab #1. The ALE signal should come up at the correct frequency after every power cycle and reset. Note: The AO bit of the AUXR register must be '0b' (its default value) in order for the processor to emit ALE like the C501 processor.
4. [**Optional**] Add a supervisory circuit (e.g. Microchip MCP-101) to protect the processor Flash memory against corruption due to power supply brownout or shutdown issues. See Atmel application note "External Brown-out Protection for C51 Microcontrollers with Active High Reset Input".
5. Design and implement your RS-232 circuit utilizing the MAX232 driver/receiver. This circuitry is not memory mapped, but instead will use the RX and TX lines on Port 3 of the 8051. On your RS-232 connector, you may connect RTS to CTS, and you may connect DTR to DSR and DCD.

   TIP: You can determine which pin on the cable is the TX pin from the PC by pressing and holding a key on the keyboard while probing the cable pins with an oscilloscope to see which pin is toggling.

   NOTE: DO NOT probe the +10V and -10V RS-232 voltages with a digital logic probe.

   Your parts kit should contain four 1.0uF caps suitable for the charge pump capacitors (C1, C2, C3, C4) for the MAX232 chip. You can use larger capacitors up to 10uF instead, if you desire. While some newer versions of the MAX232 chip can use smaller capacitors, some older versions of the chip prefer the larger caps. Using a smaller capacitor value saves space on your board; however, using a large capacitance value reduces ripple on the +10V and –10V charge pump outputs.

6. Implement the required circuit features for /EA and /PSEN to enable the Atmel UART bootloader to execute when you come out of reset. Use a **momentary pushbutton and pull-down resistor** for /PSEN, and hold that button when coming out of reset in order to force bootloader operation; then, after the bootloader has started, release that button to eliminate drive fight issues on the /PSEN line. Use a header/jumper for the /EA input. Note that if you use these hardware conditions to enter the bootloader when you come out of reset, then the Atmel bootloader is entered regardless of the values of BLJB, BSB, and SBV.
7. Verify that FLIP can communicate with the Atmel bootloader.
8. Verify that your final code for Lab #2 runs correctly on the new processor. You can use FLIP to download the code to internal flash memory on the processor.
9. Learn how to configure a terminal emulator program (TeraTerm or RealTerm are recommended) on a host computer to allow you to communicate over the serial port. Students report that RealTerm works under both Vista and XP. Students report that TeraTerm works better (much faster) than the HyperTerminal terminal emulator included with Windows. [Note: early versions of HyperTerminal did not correctly handle echoing of characters over the serial port.] Be sure that you have the serial cable hooked up to the correct serial port on the computer, and that the terminal emulator is configured to use that same serial port. When first testing your hardware, you should configure the terminal emulator to communicate directly to the appropriate COM port on the PC at 9600 baud, 8 data bits, 1 stop bit, no parity and no flow control. After you verify the hardware is working fine, you can increase the baud rate.

   NOTE: If you get an error message 'bytes not written correctly' when downloading your hex file with one terminal emulator program, try the other terminal emulator program and see if the error persists.

10. [**Optional**] This optional program just aids in making sure your system is wired correctly. Write an assembly program which initializes the 8051 serial port and then continuously (in an infinite loop) transmits the character 'U'. Using an oscilloscope, verify that the transmitted patterns correspond with the ASCII value for this character and that the baud rate is correct. Verify that the characters appear on screen.

    Now, modify the program to make it echo the characters it receives from the 8051 serial port. Every time a valid character is received, that same character should be transmitted back out the serial port.

NOTE: If you want to simulate your code using Emily52, note that SFR's are **not** emulated in our version of Emily52, so you can't simulate all features of the 8051, such as the real-time aspects of serial port operation, timers, and interrupts. However, you can still use the simulator to verify much of your code. You can simulate interrupts in Emily52 by pressing 'v' for 'vector'.

11. Complete Homework #8, regarding PAULMON2, makefiles, Eclipse, and SDCC/MICRO-C.

12. [**Required Element**[1]] Configure your hardware to use the NVSRAM as additional XRAM data memory in your system. It should respond to data reads and writes in the address range of 0x0400-0x83FF. This will provide you with a total of 33KB of system data memory (1KB of XRAM built into the Atmel AT89C51RC2 + 32KB in the NVSRAM).

13. [**Required Element**[1]] Visit the PAULMON2 web site (the link is available on the course web site) and learn about this free monitor program. Download PAULMON2 (paulmon21.asm and extra.asm) and understand how to configure its assembly code to match your memory map – you will only have to make a couple of minor changes to the code. You may see that pre-assembled versions of PAULMON2 that use different memory maps are available on the PAULMON2 web site; however, don't use those versions.

    Remember that you need to configure PAULMON2 before assembling it with the AS31 assembler. As you're analyzing the equates at the beginning of the PAULMON2 code, think about using the following values for your situation - will these work for you?

```
.equ base, 0x0000 ;location for PAULMON2 (beginning of Flash)

.equ vector, 0x2000 ;location to LJMP interrupt vectors (in Flash)

.equ pgm, 0x2000 ;default location for the user program (in Flash)

.equ bmem, 0x1000 ;beginning of memory (extra code could be in Flash)

.equ emem, 0x7FFF ;end of the memory (end of user code memory in Flash)
```

    Also add a new line of assembly code to the beginning of the PAULMON2 monitor program to enable the full 1KB of XRAM in the AT89C51RC2 processor. Note that when the processor comes out of reset, it has only 256 bytes of XRAM available. In order to gain access to all 1KB of internal XRAM, user code would need to program the XRS1:XRS0 bit field to 11b. (The data sheet also describes dependencies on the XRAM bit in the HSB, and the EXTRAM bit in AUXR.)

    You must assemble your own version of PAULMON2 with the AS31 assembler in order to match your memory map. The AS31 assembler and notes regarding its use are available on the course web site. Note that AS31 requires a different assembly source file syntax than the ASM51 assembler you have been using for your own code, and PAULMON2 is written with the AS31 syntax.

    Assemble your base PAULMON2 code to create one hex file. Then assemble the extra.asm code separately to create a second hex file. If using the device programmer to program your processor, you can then load both hex files into the device programmer buffer before programming your Flash. The main PAULMON2 code and the extra.asm code modules do not need to be linked together, since they are ORG'ed at different addresses and occupy different parts of the programmer buffer.

14. [**Required Element**[1]] Program the processor with the PAULMON2 monitor program (**including extra.asm**). If your serial port is working correctly, then when you turn on your system with PAULMON2 installed, and after you press the ENTER key, PAULMON2 should print a welcome message to your terminal emulator screen.

    Learn how to use the monitor to modify and examine hardware. Become familiar with all monitor commands, including the three extra commands ('L', 'S', 'E') provided by extra.asm. To download a hex record file to your board, use the 'D' (download) command in PAULMON2, and then use the Transfer/Send Text File menu item within the terminal emulator to transfer the hex file to your board's XRAM. Note that since the internal XRAM is located in data memory space and is not accessible with the /PSEN signal, you will not be able to execute code out of internal XRAM (i.e., you will not be able to execute the hex records that you download using the 'D' command).

- **Determine and document the maximum baud rate at which you can reliably run PAULMON2 on your system.** Using an oscilloscope, verify that the baud rate is correct.

15. [**Required Element**[1]] Verify that you can write your entire XRAM (internal and external) using the PAULMON2 block fill capability.

    Step 1) Fill the entire XRAM space 0x0000 to 0x83FF with a specific value, like 0x55.

    Step 2) Fill the rest of data memory space 0x8400 to 0xFFFF with a different value, like 0xAA. This operation shouldn't have any bad effect, as there is nothing else in your hardware that should respond to data addresses in that range (with either /RD or /WR active).

    Step 3) Dump memory from 0x0000 to 0x83FF and verify that all XRAM cells contain 0x55. If you see blocks that do not have 0x55, something may be wrong with the hardware or with PAULMON2.

NOTE: It's possible to use part of your SRAM for code storage and be able to execute code from SRAM. In that case, you'd be able to download code to the NV-SRAM using PAULMON2, and that code would remain in memory after you turn power off to the system. When you turn power back on, the code would already be ready to run. Slight adjustments to your memory map and chip select and output enable logic might be needed. A portion of your SRAM (e.g. 0x8000-0x83ff) would need to be enabled by /PSEN and /RD to respond during both code read and data read operations.

NOTE: For the rest of the semester, use the Atmel FLIP utility to program your code into the processor flash memory.

NOTE: For the following C programs, remember that your C code has to finish initializing the AT89C51RC2 chip. In order to gain access to all 1KB of XRAM, user code would need to program the XRS1:XRS0 bit field to 11b. (The data sheet also describes dependencies on the XRAM bit in the HSB, and the EXTRAM bit in AUXR.) Be sure to review the sdccman.pdf file regarding the *_sdcc_external_startup()* function. If you specify an XRAM size of 0x400 on the linker command line, you need to make sure there are actually 0x400 bytes of XRAM enabled in the hardware. Likewise, if you specify an XRAM size of 0x8400 on the linker command line, 0x8400 bytes of XRAM must be enabled.

16. [**Required Element**[1]] Learn how to use SDCC to generate hex records for your hardware using the compact (or large) memory model. Know how to verify that SDCC has been configured with the correct addresses for ROM/RAM by analyzing the SDCC output files. Verify that the correct addresses were generated in your code listing file (.rst) and hex record file (.ihx). You should also examine the other output files to see how SDCC has allocated space for objects in memory (check the other output files, such as .mem, .lnk, and .map).

    Learn how to use Eclipse to allow you to do the following:
    (a) Open a new SDCC project. **NOTE: Create a subfolder for your source files (e.g. src).**
    (b) Edit your C code and examine the files in your project.
    (c) Compile your C code and create a hex file.

NOTES: The Eclipse IDE is free (see links on course web site). The Eclipse development environment is quite powerful. It supports managed make, where the tool automatically generates and maintains the makefile based on the information you use to configure Eclipse, or standard make, where you generate the makefile you want Eclipse to use.

For source code control, the free Subversion (SVN) version control system is recommended:
http://subversion.tigris.org and **http://tortoisesvn.tigris.org/**

The Dunfield make utility is quite limited and doesn't have the same options as other make utilities you might be used to, such as UNIX make. If you want to use a stand-alone version of make, you are encouraged to download GNU make from the course web site, since it has better functionality and is free. Makefile examples are available on the course web site.

If you want to use a different 8051 compiler, such as Keil or Franklin, this is acceptable with some caveats:

1) You need to have the compiler available in the lab during signoffs, so that you can build code and demonstrate knowledge of the tool set. You will need to have a demo version or licensed version available on your notebook computer, if the tools are not installed on the computers in the ECEE Dept.

2) You will not get as much help from the TAs or instructor if they are not familiar with your tool set.

3) You must realize that examples given in class or on the web will be specific to SDCC or MICRO-C. We cannot support more tools than those related to SDCC or MICRO-C.

NOTE: When asking the instructor or TAs for help with your program via e-mail, be sure to attach all relevant files to your e-mail. Attach the source files (.c, .h), and the .rst, .mem, and .map files.

NOTE: The following C program has a little complexity. Don't try to code it all at once. Instead, identify elements of the assignment, and then write and debug in an incremental manner. For example, here are some suggestions on how one might approach the assignment:

1) Become comfortable with the compiler and Eclipse IDE/makefiles. Write very simple programs (like an 8051 pin toggling program) to gain confidence and understanding of the tools. Refer to the SDCC syntax examples that are available on the course web site.
2) Learn how to initialize the hardware properly, including the serial port.
3) Learn how to use putchar() or putch() to output single characters to the terminal emulator.
4) Learn how to input single characters from the terminal emulator, using the getchar or getch() functions. Use the putchar() or putch() function to echo received characters to the screen.
5) Learn how to create a buffer using the malloc() command. Learn how to free up memory space once the buffer is no longer needed. Learn how to use pointers to write to and read from a buffer.
6) Write a function that can print out buffer contents, using the specified format.
7) Identify other independent elements of the program assignment, and write functions to implement those elements.
8) After you have the individual functions working, then concentrate on how to use those functions to complete the programming assignment.

USE AN INCREMENTAL APPROACH TO SAVE YOURSELF MUCH DEBUGGING TIME!

START YOUR ASSIGNMENT EARLY, AS IT MAY TAKE MORE TIME THAN YOU MAY THINK.

17. [**Required Element**[1]] Write a C program which first allocates a heap of size 1800 bytes, and then prompts the user to specify an even number buffer size between 20 and 1400 bytes. The program must then allocate a buffer (buffer0) of the requested buffer size in XRAM using the `malloc()` function. The program must then malloc in XRAM a second buffer (buffer1) which is half the requested buffer size. If the malloc fails for either buffer, then any successfully allocated space must be freed and the user must be prompted to choose a smaller buffer size.

    The program must then prompt the user to enter characters. Every time a valid printable character is received at the 8051 serial port, that same character must be transmitted back out the 8051 serial port and must also be stored in buffer0. Use pointers to external memory to access the buffer. The use of buffer1 is not specified in this assignment, and buffer1 may be used for any other purpose you desire. Valid printable characters include upper/lower case letters, numbers, punctuation, single space, etc. For this lab, special control characters, such as line feed, carriage return, horizontal tab, etc., are not valid printable characters.

    The program must keep count of how many characters are received, and every time the character '?' is received, the program must **echo the character** and must also report the **total number** of printable characters received in buffer0 since the last '?', as well as the **starting address** and the **total allocated size** of each buffer (buffer0 and buffer1). The C program must clearly report these numbers on the terminal emulator screen, **along with descriptive text**. The C program must then provide some statistics on buffer0's contents, by displaying the number of times each number '0'-'9' and each lower case vowel 'a', 'e', 'i', 'o', 'u' appears in buffer0. For example, the program may report the following statistics for buffer0: "a-5; e-50; i-0", etc. Statistics for other printable characters in the buffer do not need to be displayed. The program then must empty each buffer by transmitting all the characters which were stored in buffer0, with a maximum of 70 ASCII characters displayed on each line of the screen. buffer0 will then be empty. The quality of your user interface will be a factor in the grade you receive.

    If buffer0 fills completely before a '?' command is received, any excess character subsequently received is echoed out the serial port, but is not added to buffer0 (it is discarded). Once the '?' command is received, then buffer0 is emptied, as described above.

    If the '/' character is ever received, the program must display the current contents of buffer0 in hex, **but must not empty the buffer** – the data will remain in the buffer until the buffer emptied in response to a '?' command. Display the data on the PC screen in hexadecimal (not ASCII), with 16 bytes of data per line, in the following format:

    AAAA: DD DD DD DD DD DD DD DD DD DD DD DD DD DD DD DD

    This format is similar to what you see when using the device programmer or when dumping memory contents using PAULMON2, where AAAA is the starting address (in hex) for each block of 16 data values DD (in hex).

    In order for you to maximize your grade, your code must gracefully handle erroneous inputs, such as the user trying to specify a buffer size outside the allowed range. You should verify that your code handles cases like this correctly **before** you get signed off. During signoff, you will be evaluated on the quality of your code at that point in time. You do not get an infinite number of chances to get feedback from the TA or instructor and make improvements to your code based on that feedback prior to your signoff, so be well prepared before your signoff. Code with a good user interface, robust error handling, and good structure and comments will score higher than code lacking these features.

- Submit a copy of your neat and fully commented program (source code only, **not** .LST listing file).

18. **[Supplemental Element[1]]:**

Use the processor's Counter 0 to count an external event of your choice (some signal which you hook up to both the Counter 0 input, pin P3.4). One good choice for an external event would be a different port pin on the 8051 toggled by firmware.

Configure two pins as pulse width modulation outputs, using the PWM capability of the Atmel processor. Configure one output with a 25% duty cycle and one with a 55% duty cycle.

Prove that your hardware works by providing firmware which demonstrates the required functionality. Your program must provide a well-designed user menu which allows the user to perform the following options:

- Clear Counter 0 (reset counter to zero)
- Run Counter 0 (set TR0 bit)
- Stop Counter 0 (clear TR0 bit)
- Run PWM (turn on both PWM outputs)
- Stop PWM (turn off both PWM outputs)
- Read Counter 0 and print out current value (verifiable numbers) to the serial port
- Enter Idle mode (set IDLE bit in PCON register)
- Enter Power Down mode (set PDE bit in PCON register)

If the event you are counting is a port pin toggled by firmware, provide an additional menu option which allows the user to toggle that port pin high and low.

Prove that you can control the resolution of your counter to one count per event. Note that the maximum count rate is 1/24 of the oscillator frequency.

Prove that you have successfully entered Idle mode by examining ALE, PSEN*, and XTAL2 with a digital logic probe or oscilloscope. Prove that you can exit Idle mode and continue running code when an interrupt is received at external interrupt 1 while the processor is in Idle mode. You can use C code or inline assembly to generate the ISR to handle the event.

Prove that you have successfully entered Power Down mode by examining ALE, PSEN*, and XTAL2 with a digital logic probe or oscilloscope.

- Submit a copy of your neat and fully commented program (source code only, **not** .LST listing file).

19. **[Supplemental Element[1]]:**

Hook up a momentary pushbutton switch to one of the unused Port 1 pins and configure the pin as an input. Debounce the switch in firmware (debounce both the switch press and release). Your debounce algorithm should correctly handle a button bounce time of at least 10ms, even if you do not observe any switch bounce with the pushbutton included in your parts kit. Each time the switch is pressed, increment a value written to the screen (or to the optional 74LS374 debug circuit from Lab #2). Demonstrate your solution and prove that each button press is interpreted as a single event. Usability will be evaluated during the demo. Solutions which provide a better user experience (such as responsiveness of the system to quick button presses and good instructions on the screen) will be rated higher than those solutions which are not as enjoyable to use.

- Submit a copy of your neat and fully commented program (source code only, **not** .LST listing file), including **a description of your debounce algorithm**.

---

[1] Required elements are necessary in order to proceed to the next lab assignment. Supplemental elements of the lab assignment may be completed by the student to qualify for a higher grade, but they do not have to be completed to successfully meet the requirements for the lab. The highest possible grade an ECEN 5613 student can earn on this assignment without completing any of the supplemental elements is a '**B-**'. The highest possible grade an ECEN 4613 student can earn on this assignment without completing any of the supplemental elements is a '**B+**'. ECEN 4613 students can qualify for full credit for this lab assignment by completing the required elements and any one of the supplemental elements.

20. Demonstrate your assignments and get your lab signoff sheet signed by the TA or the instructor. Be well prepared before attempting your signoff. Submit a copy of your neat and fully commented program (source code only, not .LST listing file). In addition to the items listed on the signoff checklist, be sure to review the lab for additional requirements for submission. **Style points will be deducted for poorly structured and poorly commented code.**

NOTE: You can demonstrate your hardware functionality using PAULMON2. Make sure you are very familiar with all commands for the PAULMON2 monitor. To demonstrate your XRAM and RS-232 hardware, modify values in your XRAM using the monitor, and then verify those values by reading them. Consider developing just one or two programs which allow you to demonstrate all optional hardware and firmware functionality. This will reduce the amount of time it takes to get signed off.

NOTE: Make copies of your code, SPLD code, and schematic files and save them as an archive. You will need to submit the Lab #3 files electronically at the end of the semester.

NOTE: Notes about SDCC are available on the course web site. If you're using SDCC, you should absolutely read these notes, as they should save you a significant amount of time.

NOTE: A formatted version of the AS31 assembler documentation is also available on the course web site. AS31 (not ASM51) is used to assemble the PAULMON2 source code.

TIP: When loading hex files with PAULMON2, use the fastest baud rate to shorten download time.

TIP: To speed testing, you can create text files with specific contents and send the text files from the PC to your embedded system to simulate a large number of characters being entered on the keyboard**. A fast baud rate will save you time.** Note that TeraTerm allows you to drag and drop a file into the terminal emulator window.

NOTE: When you're using a debug monitor, the monitor initializes the 8051 serial port, so you don't have to. However, note that the MICRO-C `serinit()` library function does not initialize the SMOD bit (it assumes SMOD is in its reset state of 0), so you might see a baud rate error if you use MICRO-C and PAULMON2 together. PAULMON2 initializes SMOD to 1, but since `serinit()` assumes that SMOD is zero, you will see twice the expected baud rate when using `serinit()` in a system that uses PAULMON2. SMOD needs to be correctly set in your code.

NOTE: Whenever you're not using a monitor and you're burning your code into Flash or an EPROM, remember to initialize the serial port prior to using any serial input/output function calls (e.g. write your own serial port initialization routine or call a library routine, like the MICRO-C `serinit()` function).

NOTE: Flow control only tells the transmitter to stop transmitting - it doesn't tell that system to stop receiving. You can send flow control characters from your terminal emulator by pressing Control-S (^S) and Control-Q (^Q). XOFF = DC3 = ^S = 0x13 XON = DC1 = ^Q = 0x11

NOTE: We are using SDCC in this course. However, if you purchased the MICRO-C development tools from Dunfield Development Systems, you can use the MON51 monitor in addition to PAULMON2.

NOTE: When using the PAULMON2 monitor, which has automatic baud rate detection, you must cycle power and press ENTER after each baud rate change: just pressing the 8051 run time reset button will not clear the current baud rate.

ECEN 4613/5613 **Embedded System Design** **Fall 2009**

Lab #3 Signoff Sheet

You will need to obtain the signature of your instructor or TA on the following items in order to receive credit for your lab assignment. This assignment is due by **Tuesday, October 27, 2009**. Labs completed after the due date will receive grade reductions.

Print your name below, circle your course number, and then demonstrate your working hardware & firmware in order to obtain the necessary signatures. All items must be completed to get a signature, but partial credit **is** given for incomplete labs.

Student Name: _____ **4613** or **5613** (circle one)

Signoff Checklist

Required Elements

- ❑ Schematic of acceptable quality (all components shown):
- ❑ Pins and signals labeled, decoupling capacitors, and two 28-pin wire wrap sockets present on board:
- ❑ Very good knowledge of a monitor and a terminal emulator:
- ❑ Using PAULMON2, demonstrates highest baud rate supported as: _____
- ❑ Demonstrates XRAM block fill, all 33KB of XRAM are functional:
- ❑ Demonstrates use of FLIP and ISP Bootloader:
- ❑ Knows how to use SDCC and Eclipse IDE:
- ❑ Knows how to analyze output files (.RST, .MEM, .MAP) for correct addresses:
- ❑ C serial program functional and code commented:
- ❑ Hex display of buffer contents:

_____

Instructor or TA signature and date

Supplemental Elements (Qualifies students for higher grade.)

- ❑ Counter 0 input and register read firmware functional:
- ❑ PWM control works correctly:
- ❑ Correctly enters Idle mode and exits via external interrupt 1:
- ❑ Correctly enters Power Down mode:
- ❑ All other software menu items function correctly:
- ❑ Good user interface; program is easy to use: _____

- ❑ Pushbutton switch hardware
- ❑ Firmware debounce functional and code commented _____

Instructor/TA Comments: □ □ □ **Instructor or TA signature and date**

FOR INSTRUCTOR USE ONLY	Not Applicable	Poor/Not Complete	Meets Requirements	Exceeds Requirements	Outstanding
Schematics, SPLD code	☐	☐	☐	☐	☐
Hardware physical implementation	☐	☐	☐	☐	☐
Required Elements functionality	☐	☐	☐	☐	☐
Supplemental Elements functionality	☐	☐	☐	☐	☐
Sign-off done without excessive retries	☐	☐	☐	☐	☐
Student understanding and skills	☐	☐	☐	☐	☐
Overall Demo Quality	☐	☐	☐	☐	☐

Comments:

NOTE: This signoff sheet should be the top sheet of your submission.

Submission Sheet

Instructions: Print your name below, circle your course number, sign the honor code pledge, and then demonstrate your working hardware & firmware in order to obtain the necessary signatures. All items must be completed to get a signature, but partial credit **is** given for incomplete labs. Separate this sheet from the rest of the lab and turn in this signed form, a full copy of complete and accurate schematic of acceptable quality (all components shown), a printout of your fully and neatly commented source code (**not** .LST or .RST listing files), and the answers to any applicable lab questions to the instructor in order to receive credit for your work. No cover sheet please. Note that receiving a signature on this signoff sheet does not mean that your work is eligible for any particular grade; it merely indicates that you have completed the work at an acceptable level. [**Watch for instructions about probable electronic submission via CULearn**.]

In addition to the items listed on the signoff checklist, be sure to review the lab for additional requirements for submission, including:

- ❑ Signed and dated signoff sheet as the top sheet (No cover sheet please)
- ❑ Submission Sheet with signed honor code pledge
- ❑ Full copy of complete and accurate schematic of acceptable quality (all old/new components shown). Include programmable logic source code (e.g. .PLD file), if using an SPLD.
- ❑ Full copy of fully, neatly, clearly commented source code. Ensure your code is neat and easy to read.
- ❑ Debounce algorithm description and code (if this supplemental element is completed)

Make copies of your code, SPLD code, and schematic files and save them as an archive. You will need to submit the Lab #3 files electronically at the end of the semester.

Make sure your name is on each item and staple the items together, with the signoff sheet as the top item.

Student Name: _____ **4613** or **5613** (circle one)

Honor Code Pledge: "On my honor, as a University of Colorado student, I have neither given nor received unauthorized assistance on this work. I have clearly acknowledged work that is not my own."

Student Signature: _____

FOR INSTRUCTOR USE ONLY

Submission Evaluation	Not Applicable	Poor/Not Complete	Meets Requirements	Exceeds Requirements	Outstanding
Required Elements Code Quality/Style/Comments	☐	☐	☐	☐	☐
Supplemental Elements Code Quality/Style/Comments	☐	☐	☐	☐	☐
Overall Submission Quality	☐	☐	☐	☐	☐

Overall Assessment	
Adjustments/Late Penalty	
Final Grade	

Comments: