EE247 Programming for Embedded Systems Project #2 Spring 2016

"Terminal command" Program
Expected to be working in two weeks (Feb23, 2016)

The goal of this project is to use the serial port to the Windows PC (running Hyperterminal) in order to control the microcontroller in doing a variety of tasks. We would like to be able to control a stepper motor, but choose the speed and direction by commands given through the terminal facility, as well as blink the LED's on and off, and sample the potentiometer or photosensor through the A/D converter. The student is invited to include any other features he believes would be of interest or entertaining.

The "terminal" program is a unit of code consisting of the "hcc_terminal.h" and "hcc_terminal.c" files. In addition, use of the terminal will require that you include some other files, SCI_Functions.c and hcc_types.h, and possibly others. You will need three functions that allow terminal to communicate with the serial port (they are in SCI_Functions.c). You need to pass these functions to the terminal using the "terminal_init" function when you initialize the terminal before installing commands and entering the main loop.

int putch(char c) // sends a single character. The call is a success if the same character is returned. If minus 1 is returned, the attempt failed, most likely because the "TDRE" bit (Transmit Data Register Empty) was not asserted. It is acceptable for this project to simply do a busy wait for TDRE and always succeed.

int getch(void) // gets a single character (as an int) from the serial port. (Getch should return "-1" to indicate that there was no character to be picked up. Normally this should not happen if you check RDRF (Receive data Register Full) before calling this function.

int kbhit(void) // checks RDRF to see if there is a character to be picked up. Returns "TRUE" (preferably a "1") if RDRF is one, and "FALSE" if TDRE is zero.

Sample code for using the terminal can be found in the example main program distributed earlier. For example code for getch and putch, you can also look in the Codewarrior sample project for the HCS08SH8 demo board (choose that from the menu when you start Codewarrior.) It's on the CD that came with the board (mayu be missing in some boxes). The demo project also includes code for initializing the serial port and the clock, both needed. Part of the purpose for this project is to get used to using files. I recommend you take "SCI_Functions.c" and "SCI_Functions.c" and copy those files into your project, modifying them as needed, for examplem by adding kbhit() and changing the functions to return integers rather than characters to be compatible with the terminal program. (I did that in the copies I distributed.) Copy the terminal files into your project too, and possibly also some other utilities. (The utils.h and utils.c files have code that converts between strings and numbers.)

We've looked at al lot of theis stuff in the class.

Examples of commands we want might include:

'start xx' This command starts the stepper motor at a default speed, or makes the speed xx (rpm?) as a parameter is given. Decide what units to use for speed.

'speed xx' This command changes speed to a given value.

'stop' This command stops the stepper motor

'reverse' This command reverses the stepper motor

'speedpot' Take the stepper speed input from the potentiometer (cancelled by 'speed' or 'start' or 'stop' commands)

'pot' Give the value of the potentiometer

'photo' Give the value of the photosensor

'LED xx' Turn on or off the two LEDs depending on xx. Maybe 01 = off/on

'Buttons' Print to the screen the state of the two buttons

What you should not try to do for this project: Don't try to use slower speeds for the serial port. The demo program uses 115200 bps, and that should be fine. If you try to go too slow, such as the Hyperterminal default of 2400 bps or the commonly used 9600 bps, you may run into problems because the functions "print" and others may hang in a busy-wait state, preventing your microcontroller from doing other things. That's because we are using busy-waits instead of interrupts. It's also likely that your stepper will be sporadic when communications is going on for the same reason. (Be sure to feed the dog inside any busy-wait loop.) You should step yopur motor along in the main program, NOT in any of the commands. The commands should just change the speed and/or direction. We will talk about use interrupts as a way of dealing with these and other timing issues later. You should not try to use interrupts yet. We are also not going to try to use the clock at this time. If you want to include a clock in your project as an extra, go ahead. But make sure all the essentials are working first.

You DON'T have to actually run a stepper motor. That's optional. You just have to have two outputs that show the appropriate waveforms seen on an oscilloscope.

Your report is to be informal. It is to consist of an abstract (a summary paragraph which summarizes the entire report), a copy of your C program (include appropriate comments; include both header and c code files), your calculations or observations to show motor speeds for different settings, and your observations of how well the various commands work. Include a schematic showing your stepper circuit. (Copying in what you had from project #1 is OK.) Finally, include a conclusions paragraph which should include a statement saying whether it worked or not, and if not, why not and describing what seems to be wrong.

Looking ahead: Project number three is going to be mostly this same stuff, but we are going to include a clock, and we will use interrupts for the clock, the serial port, and maybe some other things like the A/D converter. We will also use some of the sophisticated timer facilities to run the stepper instead of using a loop under program control. We'll also slow down the serial port to 9600 bps or 4800 bps. Project number 4 is going to be your idea; be thinking of neat things to do.