Assignment #5 Winter 2013
SYSC 2003 Computer Organization
Department of Systems and Computer Engineering

You must submit the files identified below <u>using the electronic Submit application</u> found in your SCE account on the undergraduate network. The submission process will be canceled at the deadline. No assignments will be accepted via email or on disk.

<u>***THIS ASSIGNMENT MUST BE DONE IN TEAMS OF 2 STUDENTS.*** **Only ONE student must submit the assignment, clearly stating who the members of the team are.**</u>

Due Date: April 2nd, 2013 @ 2:00 PM

<u>You must use the Pair Programming technique (check the course webpage for details). There will be questions about the assignment on the Final exam; students not being able to respond them will lose the Assignment marks.</u>

Important instructions about submissions:

- Submit ONLY ONE ASSIGNMENT PER TEAM.
- Each file in your Assignment should include a Header with the names/numbers of BOTH students
- Students submitting TWO copies of the assignment will receive the LOWER of the two marks and will receive a demerit of 20% of the final mark.
- Students who, by mistake, submit more than one copy, should contact the instructor BEFORE the deadline to fix the problem.

Objectives: Programming simple I/O devices in the Project Board. C programming of the HC12 with ICC12. Programming advanced I/O devices. Hardware interrupts.

In this assignment, we will continue building a Control system that we started discussing on Assignment 3. As a reminder, ALL THE EXERCISES MUST RUN ON THE BOARD. ALL THE RESULTS (FINAL AND/OR INTERMEDIATE) MUST APPEAR ON THE TERMINAL.

The idea is to build a Control system for a mobile robotic controller. It will include software to control the robot’s speed, direction, collision detection and other control applications. We will start by creating very simple subroutines, but, in order to understand the whole idea of the system you will have built at the end of the course, we will start with a general description now. All the systems in the house are controlled using a single microcontroller.

There are a few subsystems controlled:

1. Speed: the robot's speed should be controlled according to the desired speed value. A PWM controlled motor will be used with that purpose.
2. Collision Detection: if a robot approaches an obstacle, it should rotate its direction. A pulse motor will be used with that purpose (connected to the front wheels of the robot).
3. Emergency management: the robot should detect the current temperature in the building and issue an emergency call if a high temperature is detected.
4. Interface with the user: the user can use a keypad to turn on/off the robot, change the robot's direction and speed. Also, an LCD display and LEDs are used to give information about the robot's status.

We are a part of the Software Engineering team, which is in charge of building the software for this control system. This device is going to be built by a multidisciplinary team including electrical engineers (in charge of the electrical components), mechanical engineers (in charge of the mechanical components), and non-technical staff (user manual's writers, marketing personnel, etc.).

During the requirements analysis, it was decided to build the software on an embedded platform using an HC12 microcontroller (like the one available in the lab). Although most of the hardware for the robot is not still available (i.e., the actual robot has not been built yet), the Electrical Engineering team has already designed the interfaces to be used. We know that we can use the lab's Experimental Board to start doing some basic tests that will serve as the basis for the actual house, as follows:

- The LEDs in the system will be connected to the same circuits used for the lab's LEDs interface in the lab;
- The heater will be connected to our lab's heating device interface;
- The panel display in the robot is the LCD display that we have in the lab,
- The temperature sensor is connected to the AD interface (like in the lab)
- The sensors, buttons of the user interface, etc, use exactly the same interface than the one used for the keypad in our lab

Therefore, we have all the equipment needed to start developing the software for this control system. In that way, when the mechanical and electrical components of the robots are built and ready for testing, we can integrate our board, software, connect the tool interfaces to the I/O lines in the robot, and we are ready to do integration testing on the actual robot.

In order to build such complex application, we will start with very simple tasks. At this point, you will:

a. Define the different subroutines discussed here, using the Policies included on each exercise
b. Write a Main program (to be delivered with your subroutine) with the simple purpose to test your subroutine
c. Write good documentation for your main program and your subroutine

In brief, you have to solve the following exercises:

1. [5 marks] **assign51.prj, assign51.c, assign51asm.s, DP256reg.s, vectors_dp256_NoICEb.c (provided), assign51.SRC, and assign51.s19**

Construct an INTERRUPT-BASED version of Exercise 4.1, that is, an INTERRUPT-BASED program that will detect the keys pressed in the keypad and act accordingly.

Build a **C application** that will check the keys pressed and will act according to the exercise specifications.

The program runs forever until you PRESS '0'.

DISPLAY EACH KEY PRESSED ON THE TERMINAL(in this exercise you just need to display every key on the terminal).

Then, add an interface with the LCD display. This routine must:

. Display: "Speed: ****   " on the first line (**** is a number representing the current speed in km/h. Every time the "E" key is pressed, this number should be incremented; when you press 'D', the number should be decremented).
. Display "Temperature XXC" (where XX is the current temperature) on the second line.

You can use Assembly or mixing C and Assembly.

Hint:

- First build a routine that only takes care of the display with "dummy" values, and ensure they are displayed properly.
- Display every button pressed on the terminal for testing.

2. [5 marks] **assign52.prj, assign52.c, assign52asm.s, DP256reg.s (provided), assign52.SRC, and assign52.s19**

Build a controller for the robot's motor (the one moving the robot's wheels). The motor is driven by a DC motor interfaced to the PWM circuits in the lab (IMPORTANT: THIS IS NOT THE FAN IN THE EXPERIMENTAL BOARD; IT IS THE PWM-CONTROLLED MOTOR). A light detector is used to count pulses on the PACA for calculating the speed of the motor.

In this part, you must build a simple controller for the motor. The motor should increase speed until 10 RPS is obtained (or higher, but not faster than 30 RPS). Your program should use the keypad ISR to reach this speed and to slow down and turn the motor off.

You may use any of the timer services to perform the required logic, but all logic must be contained within ISRs, with the main() program doing nothing after the initialization phase.

3. [5 marks] **assign53.prj, assign53.c, assign53asm.s, DP256reg.s (provided), assign53.SRC, and assign53.s19**

You will now write a program to control the temperature. We suggest you to develop this in two steps, but you must only submit part b) (if you just submit part a. will be evaluated, and you might get partial marks).

a) First, write a program to measure the current temperature (read temperature from the temperature sensor), and display it on the terminal windows using C's printf( ) function. Your main program shall repeatedly initiate a temperature conversion and then wait until it is complete to read the current temperature using it to display. The AD conversion routine must be interrupt-based.

b) Now, make the main program control the temperature via the heater. Turn on heater if the temperature is at or below 100F; otherwise, turn it off. The heater is turned on/off by setting/clearing bit 7 in Port M. Keep in mind that your main program will be in charge of turning on/off the heater. You should keep track of the current temperature in a global variable**.**

4. [30 marks] **assign54.prj, assign54.c, assign54asm.s, DP256reg.s, vectors_dp256_NoICEc.c (provided), assign54.SRC, and assign54.s19**

Software integration

Put together the whole system, according to the specifications above. Write the software for the whole system. It must work as in the specification. It should integrate all the components: keypad, LCD display, motors, temperature control, beeper, LEDs, etc.

All the functions should be included now in the RTI, AD, PWM, OC and Keypad ISRs. You have to react to the user's request, set the LEDS, react to the commands, and integrate the opening/closing of the window to the system. The control system should be active. The main() program will just install the ISRs and execute an infinite loop (you can put your display functions on the main program).

YOU MUST PREVENT INTERRUPT INTERFERENCE.

<u>Suggestion</u>: Work incrementally. You will receive partial marks for a correct but incomplete version of the software (submit only when you have a complete working version, even if that version does not contain the complete functionality; display a message on screen and a comment discussing what functions have been included in your submission).

Optional: use key "0" to restart the system. This will make easier to test different conditions without reloading the program.

5. [20 BONUS marks] **assign55.prj, assign55.c, assign55asm.s, DP256reg.s, vectors_dp256_NoICEd.c (provided), assign55.SRC, and assign55.s19**

NOTE: this exercise will be marked by Prof. Wainer.

Now that you know everything about this hardware and how to program: what would you do with it?

The idea of this optional exercise is to allow you to be creative.

You can extend the vehicle software we built throughout the term, or build a complete application from scratch. You can use any I/O devices.

Mandatory activities:

a. The main program must run in the background, doing not important tasks only. It installs the ISRs, and then executes low priority tasks (for instance, displaying information on the LCD display).

b. All the I/O devices that can generate interrupts should be programmed to process interrupts. The only exception is the keypad (according to the application, the keypad can be part of a low priority task, and could be handled in the main program when no interrupts are running in the foreground).

Checklists and Hints

1. Remember to create a new version of vectors_dp256_NOICE.c with entries for your new ISRs. Notice that we are using the PAI input edge interrupt, not the overflow interrupt.
2. Designing your interrupt-driven program involves two fundamental chores (1) division of labour amongst ISRs and main thread – read the assignment description carefully for instructions vis-à-vis and (2) use of global variables to communicate/synchronize
3. Create an init() function in which you place all of the hardware initialization code.
4. Does the motor seem to rev ? Is it going way too fast ? In our work, we have found the following :
   a. You need to use the scaled Clock B.
   b. For a nice motor speed use Clock B = bus clock and have Scaled B divide Clock B by 0xFF. Set the duty period to be 100 and let the duty cycle range from 0…10ish.

5. When initializing the PWM for the DC motor, you may want to initialize all parameters. In doing so, you will set the period. What about the duty ? In the final product, your program will be varying the duty to speedup/slowdown but
    a. If you want to test that your PACA ISR is running, you will need to have your motor running (if motor is stopped, no PACA interrupts). Initialize duty = about 10.
    b. In your final program, you will want to start with the motor off, so initialize the duty=0.
6. The bigger the duty cycle, the faster the motor spins. For the safety of our DC motor, please put a safety catch in your program to prevent the duty cycle from exceeding a safe maximum, or from going negative (when you're stopping the motor). For example :

```
const int MAX_DUTY = 15;
const int MIN_DUTY = 0;
…
if (duty < MAX_DUTY) duty ++; ….
if (duty > MIN_DUTY) duty --; …
```

7. To enable/disable in C : INTR_ON() or INTR_OFF() or _asm("cli") or _asm("sei")
8. Troubles with the board ? Interrupts don't seem to be working ? Please first try the board with the demo program on the webpage.
9. **Minimize the number and the length of any printf()'s**. These use the serial channel and markedly slow down and change the timing of the program execution, and will result in missed interrupts.
10. Don't forget to set the DDR's on Port M and K. When setting the DDR for Port P (for motor's direction), do a bit-set affecting only the relevant bit; remember that P0..3 are used as part of the LCD and of course P7 is used as PWM for the motor.

Assignment 6 Marking Criteria:
An assignment will receive **UNS** (UNSatisfactory 0) if it is late **OR** any of the directions are not followed (including late submission or printing rude messages).

Assignments will receive the marks specified on each exercise if it assembles and runs cleanly when run by the TA, AND subroutine policies have been followed.