

## **CEG 3136 – Computer Architecture II**

### **Lab 4 – Using Interrupts**

#### **Objectives:**

To demonstrate the concept of interrupts.

#### **Tasks:**

- Read chapters on Interrupts (Chapter 12) and HCS12 Timer (Chapter 14) in the Cady text and consult the MC9S12DG256 Reference Manual.
- Complete the prelab work according to instructions presented at the end of this document before arriving in the lab.
- Demonstrate your functioning system to the TA at the end of lab session or the following week.
- Submit to Virtual campus your source code files in a zipped file and the lab report in Word format.

#### **Equipment Used:**

- Windows PC
- Dragon-12 Trainer

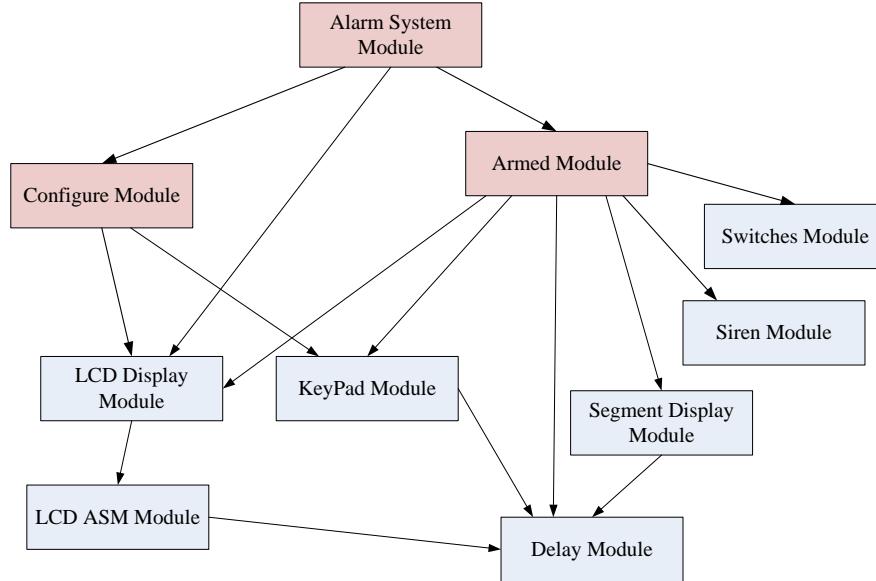
#### **Components Used on Dragon-12 Board:**

- 7-segment LED Displays
- KeyPad
- DIP Switch
- LCD Display
- PC Speaker

## Lab Software

### Introduction

The CodeWarrior project is provided for the alarm system. As in lab 3, it uses the keypad, the 7-segment displays and the LCD. In addition the PC speaker on the card is used to sound the alarm, that is, it shall act as an alarm siren. The software modules defined for this project are shown below.



By now, you should be familiar with most of the modules presented in the above diagram. There is one new module: the Siren Module used to manipulate the PC speaker. In addition interrupts are used in the Alarm System project, that is, a number of modules shall be modified to use interrupts.

Using interrupts in displaying the characters on the 7-segment displays eliminates the `segDisp()` function from the Segment Display module. It will also improve the functioning of the displays by eliminating any flicker when keys on the keypad are pressed.

Interrupts with the timer are used to implement delays and thus the Delay module is redesigned. Also the PC speaker is driven by the timer with interrupts; a waveform is applied to the speaker for creating sound. Finally timer interrupts are also used in the keypad module to read keys from the keypad.

Update the Keypad and Segment Display modules as well as create the Siren module according to the following guidelines. The new Delay module is provided to you and serves as an example to help design other modules that use timer interrupts. Note that you will need to use the LCD Display (with no modification) from Lab 3 (supplied in the project files).

1. **Delay Module:** This module is redesigned of the Lab 3 Module and re-written using C programming. The `pollDelay()` and `setDelay()` subroutines have been eliminated. The entry points to the C module are:

```
void initDelay(void); For initialisation of the module.
```

`void delayms(int):` To provide a delay of a number of milliseconds. This function plays the same role as in lab 3. However its implementation has changed. See the description of the interrupt service routine (ISR) below.

`void setCounter(volatile int *):` Note that this function receives an address to an `int` variable in a pointer variable. The function will save this address in a module global pointer variable and an interrupt service routine decrements the contents of the referenced variable every millisecond (see below). The Armed module logic has been changed to use a counter directly to countdown 10 second delays instead of calling a `polldelay()` routine as in previous labs. To disable the counter `setCounter` should be called with the argument `NULL`.

Delays are implemented using the timer channel 0 configured for output compare as follows:

- The channel is configured to generate interrupts at 0.1 ms intervals. An internal variable is used to count 10 intervals to give a 1ms delay. Every 1 ms, two other counter variables are decremented.
- The ISR servicing the timer channel 0 interrupt decrements a principle counter variable (every 1 ms) in memory for use with `delayms`. The counter variable is then be used to generate a larger delays by initialising the counter.
- As well, it decrements (every 1ms) the value referenced by an integer pointer set by `setCounter`. This allows the armed module to pass and monitor a counter variable directly.

2. **Segment Display:** This module is a modification of the module from Lab 3. It will provide only 3 entry points: one to initialise the module, `initDisp()`, one to clear the displays, `clearDisp()`, and one to set characters to be displayed on the individual 7-segment displays, `setCharDisplay()`. The function, `segDisp()`, that makes the characters appear on the display is no longer required. Instead use a timer channel to generate a sequence of interrupts. The interrupt service routine changes the display enabled (and corresponding code). Note the difference in how characters are displayed compared to Lab 3 (i.e. no flicker occurs when keys are pressed – comment in your report on this difference).
3. **Siren Module:** This module drives the PC speaker that mimics the alarm siren. It shall provide the following entry points:

`void initSiren(void):` Initialises the hardware for implementing the siren. Note that the speaker is attached to the pin of timer channel 5.

`void turnOnSiren(void):` Turns on the alarm siren, that is, the PC speaker. This is achieved by generating a waveform on the pin of the timer channel 5. Use interrupts to generate this waveform.

`void turnOffSiren(void):` Turns off the alarm siren, that is, the PC speaker. This is achieved by removing the waveform on pin 5 of the timer channel 5. Ensure that a 0 V level is applied to the output pin after the siren is turned off.

4. **Keypad Module:** Revise the module to use timer interrupts for scanning the keypad. As in lab 3, the module provides the following entry points (but this time as a C module rather than an assembler module).

`void initKeyPad():` Initialize the hardware (i.e. Port A) connected to the keypad and a timer channel used to generate interrupts at regular intervals.

`char readKey():` Read a key from the keypad (returns the ASCII code of the key pressed). The value of the key is returned once the key is released (this means you must debounce both

edges of a key press). Actually, the interrupt service routine (ISR) will be reading key presses. This function simply reads the results from variables set by the ISR.

`char pollReadKey():` Simply checks if a key has been pressed and returns its value; otherwise returns a `nul` character (0) (NOKEY constant). Like `readkey`, this function scans variable(s) set by the ISR that reads key presses.

Consider the following guidelines to help design the new Keypad Module:

- Recall the steps in reading a key: detecting a key press, debounce the key press, determine the key code of the key pressed, convert the key code to an ASCII character, wait for the release of the key press, debounce the key release.
- Design an interrupt service routine to perform these steps. When a key has been read, the ASCII code of the key pressed is stored in a variable to be read by the foreground program (that is, `readkey` or `pollreadkey`). The foreground program should clear the variable to indicate that the key has been read (set the variable to NOKEY).
- Determine the time between interrupts that would be appropriate for the achieving the above steps. In particular consider timing for debouncing.
- DO NOT include any delay in the interrupt service routine. Such routines must be kept short.
- Consider defining a **state machine** which reflects the steps given in the first point.
- Present the key ASCII code to the main program only after the key has been released.

### On using Interrupts:

You will be using a number of interrupt service routines (ISR). One interrupt is invoked by one of the Timer channels configured as a single output-compare channel for updating the display. Another timer channel is configured to create delays (timer channel 0). Another interrupt on the timer channel is used to generate the waveform applied to the PC speaker (timer channel 5). Finally another channel is used to generate interrupts for reading the keypad. Note that all ISRs are very short either flag actions to the main code via variables, or update registers/variables (as in the case of updating the displays). See the above descriptions for more details.

## **What-to-do LIST**

### **PreLab related part:**

- a. Complete a draft lab report in Word which includes the following:
  - Draw an electronic diagram of the components on the Dragon-12 Trainer used in the alarm system, including the keypad, the LCD, the 7-segment displays, the DIP switch and the PC speaker.
  - Design of the updated and new module. This includes the updated modules, Keypad and Segment Display module, and the new Siren Module.
- b. Do write the project code using CodeWarrior and ensure that the project compiles with no errors before coming to the lab session.
- c. You will need to show the completed prelab work at the lab session.

### **In the lab:**

- a. Load your software into the Dragon-12 card using Codewarrior. Run the alarm system and test all functions of the alarm system to ensure that both displays (7-segments and LCD) and the keypad are operating properly as well as the siren.
- b. Show to the TA your up-and-running system. He or she will record your success.

### **The report**

- a. Include your design material – circuit diagram, software design, etc.
- b. An explanation on the elimination of the display flicker.
- c. Also provide your source code in a seperated zip file.