CEG 3136 – Computer Architecture II
Lab 2 – Hardware Interfacing – Keypad

Objectives:

To introduce the interfacing of the Motorola 9S12DG256 through an implementation of a keypad unit.

PreLab:

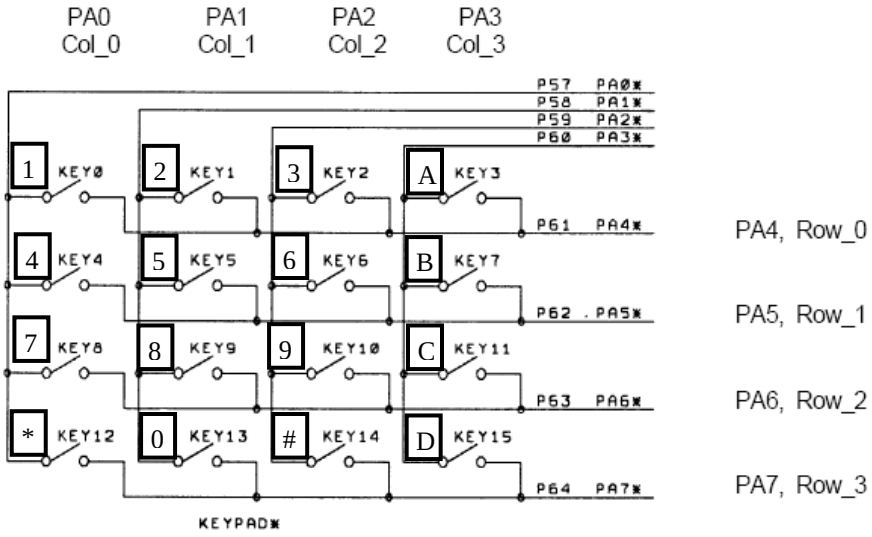
- Marks are assigned for the pre-lab work.
- Please read this entire document before coming to the lab session.
- Prepare a draft of the lab report.
- Read up on parallel ports Chapter 11 in the text book and Sections 3.1.1, 3.1.3 and 3.1.10 in the document "S12MEBIV3.pdf" that is part of the data sheets for the MC9S12DG256. Information on interfacing to keypads is provided in Section 18.2 (Array of switches) in the textbook and in section 4.4 of the Dragon-12 User Manual.
- Attend the Tutorial 3 on configuring the parallel ports.
- Design your system (hardware and software):
    - Provide a circuit diagram that will be used in the lab (that is how the keypad and switches are connected to the MCU including pull-up resistors).
    - Document the design of your modules in the lab report draft (using C as pseudo-code for defining subroutine algorithms).
- Code and correct your programs using MiniIDE.
- Please show your prelab work (draft lab report and assembled programs) to the TA at the start of the lab session.

Equipment Used:

- Windows PC
- Dragon 12 Trainer

Description:

In this lab you will be developing software to read from a keypad connected to the micro controller via parallel port A on the Dragon-12 and integrating the module into the Alarm System Software. Please consult the lab manual and Dragon 12 documentation for details on the various components used in the lab circuit. In particular in the Dragon 12 documentation (Section 4.4), review the operation of the keypad connected parallel port A as shown in Figure 1 below. Note that the Dragon-12 provides some direction to reading the keypad – be aware that the proposed approach in the Dragon-12 approach will NOT be used in this lab.



Keypad connections:
PA0 connects COL0 of the keypad
PA1 connects COL1 of the keypad
PA2 connects COL2 of the keypad
PA3 connects COL3 of the keypad
PA4 connects ROW0 of the keypad
PA5 connects ROW1 of the keypad
PA6 connects ROW2 of the keypad
PA7 connects ROW3 of the keypad

Figure 1 – Connection of Keypad to Port A

Notes on Contact switches (keypad keys)

Momentary contact switches are typically used in a keypad. A switch closure can be detected by the microcontroller unit (MCU) using the simple circuit shown in Figure 2. The pull-up resistor provides logic 1 when the switch is opened and logic 0 when it is closed. Since the MCU's I/O ports have built-in pull-up resistors when pins are set as inputs, no external pull-up resistors are required. The implementation can then be simplified to the circuit shown in Figure 3.

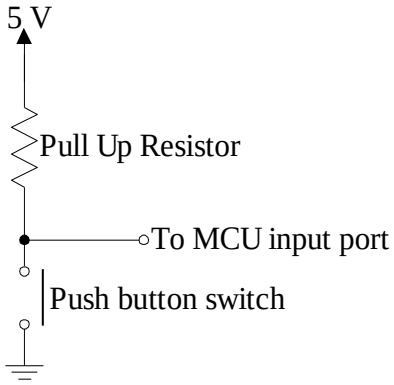


Figure 2: A Keypad Switch

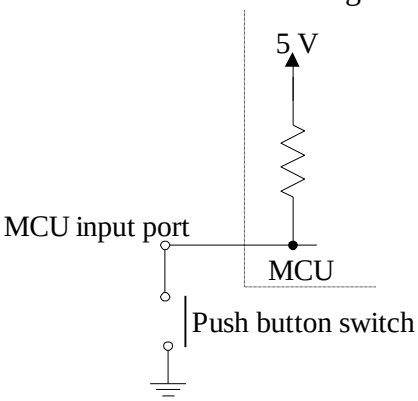


Figure 3: A Keypad Switch with MCU built-in Pull-up Resistor

Unfortunately, switches are not ideal in that they do not generate a crisp 1 or 0 when they are pressed or released. At the fast sampling speed of the controller, a seemingly brisk and firm switching action may become comparatively slow. In fact, during a switching action, switch contacts oscillate ("bounce") between the connected and disconnected states after initial contact or separation before it settles. This random process typically lasts about 2-20ms. Figure 4 illustrates the switch bounce behavior when a key is pressed and released. Therefore, in determining when a button is pressed, the MCU must ensure that a firm physical connection has established. This can be done by waiting about 10ms after the first connection or separation is detected, and checking again to confirm if the same button has been depressed. Once a key press has been confirmed, the MCU must wait for the release of the key (i.e. detect the lagging-edge of the key press). Otherwise the MCU will read the single key press as multiple key presses.

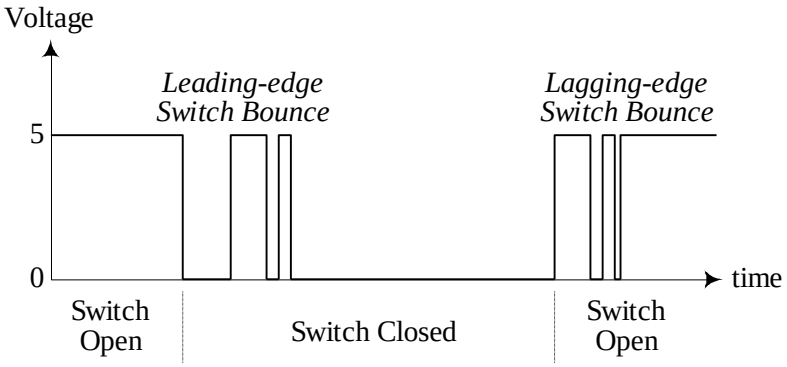


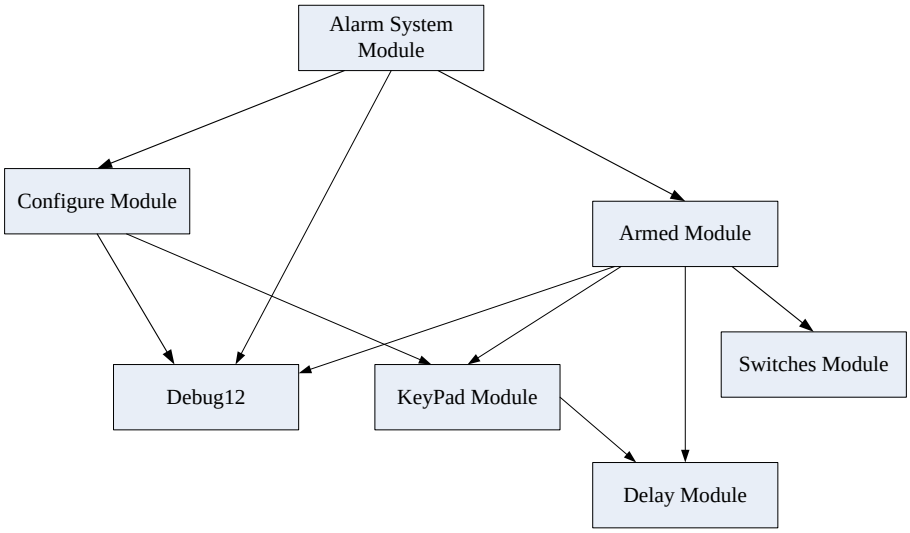
Figure 4: Switch Bounce

An approach for Scanning the Keypad

When a key is pressed on the keypad, a distinctive code will appear in the PORTA data register when a 0 is written to the line associated to the column of the key pressed (see section 4.4 of the Dragon 12 documentation). Use a table to translate the value read from the PORTA data register to a character shown on the Dragon 12 Board (Figure 2 above also shows how the keys are mapped). Using such a table will simplify the scanning process of the keypad. Take the time to think through the scanning procedure of the keypad, that is what are the steps to be taken by the CPU to determine when a key is pressed and what key is pressed. Tutorial 3 should help providing some ideas. Document the table and your scanning approach in your report.

The Alarm System Assembler Software

The main task of this lab is to develop a Keypad module, integrate the module into the Alarm System software as shown below, and update the Delay module.



All modules except for 2 are provided. The modules you are to develop shall be stored in separate files and provide functions as follows:

1. KeyPad.asm: Contains the code for the Key Pad Module. This module shall contain the following functions/subroutines:
   a. void initKeyPad():initialize the hardware (i.e. Port A) connected to the keypad
   b. 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).
   c. char pollReadKey(): simply checks if a key has been pressed and returns its value; otherwise returns a nul character (0).
   d. Note: In assembler, use register B to return the character values for *readKey* and *pollReadKey*.
2. Delay.asm: Modify the Delay module from Lab 1 to include an additional function/subroutine, *void delayms(int num)*, that can create a delay of *num* milliseconds where *num* is the subroutine parameter. This additional subroutine is meant to support the Keypad module that requires delays for debouncing. In assembler, use the D register to pass the value of *num* to *delayms*.

The Switches Module is a new module not described in lab 1. Its function is to monitor the status of the switches connected to PORT H of the micro controller. These switches represent the door and window switches that the alarm system is to monitor. For simplicity, debouncing of these switches is not implemented. This does have a side effect: a switch closure can be detected without actually closing a switch (just try tapping the switches with your finger). You may consider adding logic for debouncing to the module. This is an optional task. Do not debounce each switch individually. Think about debouncing all switches simultaneously.

Switch Module Design

The module contains the following functions/subroutines.

a) *initSwitches*: Initialises Port H pins as input pins for reading the status of the switches. Pull-up resistors are enabled since the switches pull the voltage on the pins to ground.

b) *getSwStatus*: This subroutine simply returns the contents of the PORT H data register which reflects the status of the pins. When a pin is high, the switch is open, when a pin is low, the switch is closed.

```
/*---------------------------------------
 * Function: initSwitches
 * Parameters: none
 * Returns: nothing
 * Description: Initialises the port for monitoring the switches
 *              and controlling LEDs.
 *---------------------------------------*/
void initSwitches()
{
   DDRH = 0; // set to input (switches)
   PERH = 0xff; // Enable pull-up/pull-down
   PPSH = 0xff; // pull-down device connected to H
                // switches ground the pins when closed.
}

/*------------------------
 * Function: getSwStatus
 * Parameters:  none
 * Returns: An 8 bit code that indicates which
 *          switches are opened (bit set to 1).
 * Description: Checks status of switches and
 *              returns bytes that shows their
 *              status.
 *--------------------------*/
byte getSwStatus()
{
    return(PTH);
}
```

Include Files

`Reg9S12.inc`: This include file contains the EQUATES for the symbols to hardware register addresses and can be used in all Modules that require access to hardware registers. There is a problem of defining a symbol multiple times. If this file is included in both *Keypad.inc* and *Delay.inc*, then when the program *Alarm.asm* is assembled, a number of errors will indicate that symbols are multiply defined. To resolve this problem, the conditional directives are used to include the file contents only if it has not yet been include as follows:

1. In the `Reg9S12.inc` file define the following symbol is defined `REG9S12 EQU 1`.
2. In the `Reg9S12.inc` file add the following directives at the beginning and end of the file to sandwich it's content:

```
ifndef REG9S12  ; include only if not yet included

        < content of file >
endif
```

   If the `REG9S12` symbol is not defined, then the contents of `Reg9S12.inc` file will be included, otherwise it will not since it already has been included.

`sections.inc`: The provided include file "sections.inc" contains definitions of the various sections for the project. Inclusion of this file in all module files allows assembling modules individually without errors due to missing section definitions. As in the case of the Reg9S12.inc file, note the conditional directives in the file to allow for assembling all modules together (see the next paragraph).

Hints

Here are some hints and guidelines to help you in your development:

1. To experiment with the hardware, you can test your configuration by simply modifying the parallel port registers. Using the D-Bug12 MM (memory modify) command, initialize the appropriate ports. You can use the same command to modify the data registers to check the control of the keyboard (use memory display (MD) command to see the effect of a key press in the data register).
2. Share the preparation by sharing the modules between the two partners in the team. For example, one member can be responsible for the design and coding of the Delay module and say the subroutine to translate the code from port A to an ASCII value, while the other prepares the rest of the KeyPad module.
3. Here are some guidelines to help in designing the *readKey* subroutine. **Do follow** these guidelines. Reading a keystroke from the keypad and translating the keystroke to an ASCII value requires the following logical steps:
   a. **Monitor the keypad until a key press is detected**: This can be accomplished by writing all zeros to the output pins and monitoring the value of PORTA to see if all input pins are 1. For example, if pins 0 to 3 are input pins, then reading from PORTA gives 0x0F when no key is pressed. In this case, when a value other than 0x0F is found in PORTA, then a key has been pressed or noise has occurred on one of the input lines. This step can be implemented as a simple loop that continually tests PORTA.
   b. **Debounce the key press**: After a key press is detected, wait 10 ms (call to *delayms(10)*). If PORTA's value does not change, then assume that a key is pressed. Thus the value of PORTA is saved before the delay and tested against the values of PORTA after the delay.

You may wish to consider a loop so that only when PORTA maintains a value showing a key pressed (i.e. for 10 ms) the loop is broken.

c. **Determine the code of the key pressed**: Set each of the output pins to 0 (the other output pins are set to 1). If the input pins are all high, cycle to the next pin. When a low level is found on one of the input pins, then PORTA contains a unique code that can be used to determine the key pressed. Consider implementing this step is a separate subroutine. Please take the time to understand how this logic works by examining the circuit showing how the keypad is connected to the MCU.

d. **Wait for the release of the key and debounce**: This is simply implemented with a loop that tests PORTA until all input pins are 1 (after setting all output pins to 0) followed by a delay of 10 ms (call to *delayms(10)*). This is actually a rather simple approach. To be full proof, the value of the PORTA should be checked to ensure that all input pins are still 1.

e. **Translate the code to an ASCII value**: Translate the code representing the key pressed to an ASCII value. Create a table (a 2 column matrix) to be searched for the key code (first entry of each row of the matrix). When the code is found, return the corresponding ASCII code found in the second entry of the matrix row). Use a separate subroutine to implement this step which can be called by *readKey*. Indexed addressing plays an important role here.

4. To implement *pollReadKey*, consider that *readKey* provides all the necessary logic for actually processing the keypad when a key is pressed. It is not necessary to repeat all the logic for *pollReadKey*. Consider simply checking a key press for a short period of time, say 2 ms, and return *nil* (value 0) if no key press is detected. When a keypress is detected, use a short debounce period, say 1ms, to gain confidence that indeed a key has been pressed before calling *readKey* to complete the debouncing (to wait for key bounce to terminate and the release of the key) and translation of the key code to an ASCII value.

What-to-do LIST

PreLab related part:

a. Design your system hardware and software (according to guidelines provided) that is to be included in your report:
    a. Complete the circuit provided in the lab write-up to show how the keypad and switches are connected to the MCU and how MCU pull up resistors are used.
    b. Document the design of each module using C as pseudo-code.
b. Prepare the assembler code for your system:
    a. Write the code for each module (according to the guidelines provided). The code should be assembled before you arrive at the lab.
c. You must show the Word draft lab report (that includes your design) and assembled code (no errors) upon arrival at the lab to your TA (marks will be assigned for this work). It is important to have this part of the lab ready since debugging and getting the software to work can take some time. It will not be possible to design and create the software at the lab session.

In the lab:

a. Load your .s19 files onto the MC9S12DG256 RAM. Run your program and test all the numbers generated by the keypad and the keys A and C that are used to control the alarm system.
b. Show to the TA your up-and-running system. He or she will record your success.

In your report:

a. Include the completed hardware diagram (show how the keypad is connected to the microcontroller).
b. Provide the final design of the software modules you developed. Include C "pseudo-code" and a description of the algorithms used. Provide your assembler source code files (in a zip file).