OREGON INSTITUTE OF TECHNOLOGY

**Computer Systems Engineering Technology Department**

***CST 204 - Introduction to Microcontrollers***

**Lab 3 – Hex Keypad “Service” and Application**

# Introduction to Lab 3

**LAST LAB**:

In lab 2, you implemented the LCD capability on the Explorer 16 Development Board. The “test application” cycled “**LINE 1 TEXT**” – **BLANK** – “**LINE 2 TEXT**” – **BLANK** on a 1 second change basis. Then your “clock application” put a digital clock on Line 1 of the LCD.

The LCD configuration involved:

* PIC32 I/O pin **RB15** ≡ **RS** signal, **RD5** ≡ **RW’** , **RD4** ≡ **E**, and lastly, **RE7**–**RE0** ≡ LCD Data Bus signals **DB7**–**DB0**.
* Configure the LCD module with **Function**, **Entry**, and **Display** settings using the LCD “driver” functions that you wrote. These driver calls included **lcd\_write**, **lcd\_read**, **lcd\_wait\_for\_not\_busy**, and **lcd\_write\_string**.
* **instr\_busy\_wait** was a useful “pre-*Real Time Executive*” function that *blocked* execution for approximately N cycles of 200 ns, where N is an input argument.

**THIS LAB**:

In Lab 3, you will add a hex keypad “service”. Its sole job is to **scan** the keypad and indicate if a key has been pressed. Any application can then use this indication. The service will run in the real time executive updating the keypad “status” once every 10 ms loop and communicating the status through a global variable defined in the **data.S** file:

* **KEY\_PRESSED** = 0 if **no** key is pressed (**all** keys released)
* **KEY\_PRESSED** = 1 if **any** key is pressed

A *pressed key* will be identified by two global variables **keypad\_row\_mask** and **keypad\_col\_mask**. These masks will indicate the row and column by a “1” bit in the row/col position:

**0001** – **Row or Col 0 (binary code 1)**

**0010** – **Row or Col 1 (binary code 2)**

**0100** – **Row or Col 2 (binary code 4)**

**1000** – **Row or Col 3 (binary code 8)**

The row and column positions on the keypad face are defined below. The row/col masks will also be used by the service to drive the output pins **RD3**–**RD0** into the keypad row switch contacts and sense the column switch contacts at input pins **RB3**, **RG9**, **RB1**, and **RB0**.

The keypad interface will implement the following enhancements:

1. Switch debounce on column input pins
2. On a multi-key press, first key sensed in scan-order assumed and ignore any others
3. On a multi-key press, two-key lockout operation

# Objectives for Lab 3

The microcontroller objectives are:

1. Be able to initialize port pins for input as well as output functionality (**PORT**, **TRIS**, **ODC**)
2. Be able to configure Digital input pins vs. Analog input pins (**AD1PCFG**)
3. Be able to configure Digital Output pins with *internal pull-up resistors* (**CNPUE**)

The keypad objectives are:

1. Be able to scan a hex keypad and react to a single key press
2. Be able to implement a software debounce function
3. Be able to implement a lockout strategy for simultaneous key presses

# Description for Lab 3

1. System Function:
   1. The **app\_clock** application is assumed to running, displaying the time on Line 1 of the LCD. The time shown below is for illustration only and the clock time should be advancing.
   2. If no key is pressed, the text should read:

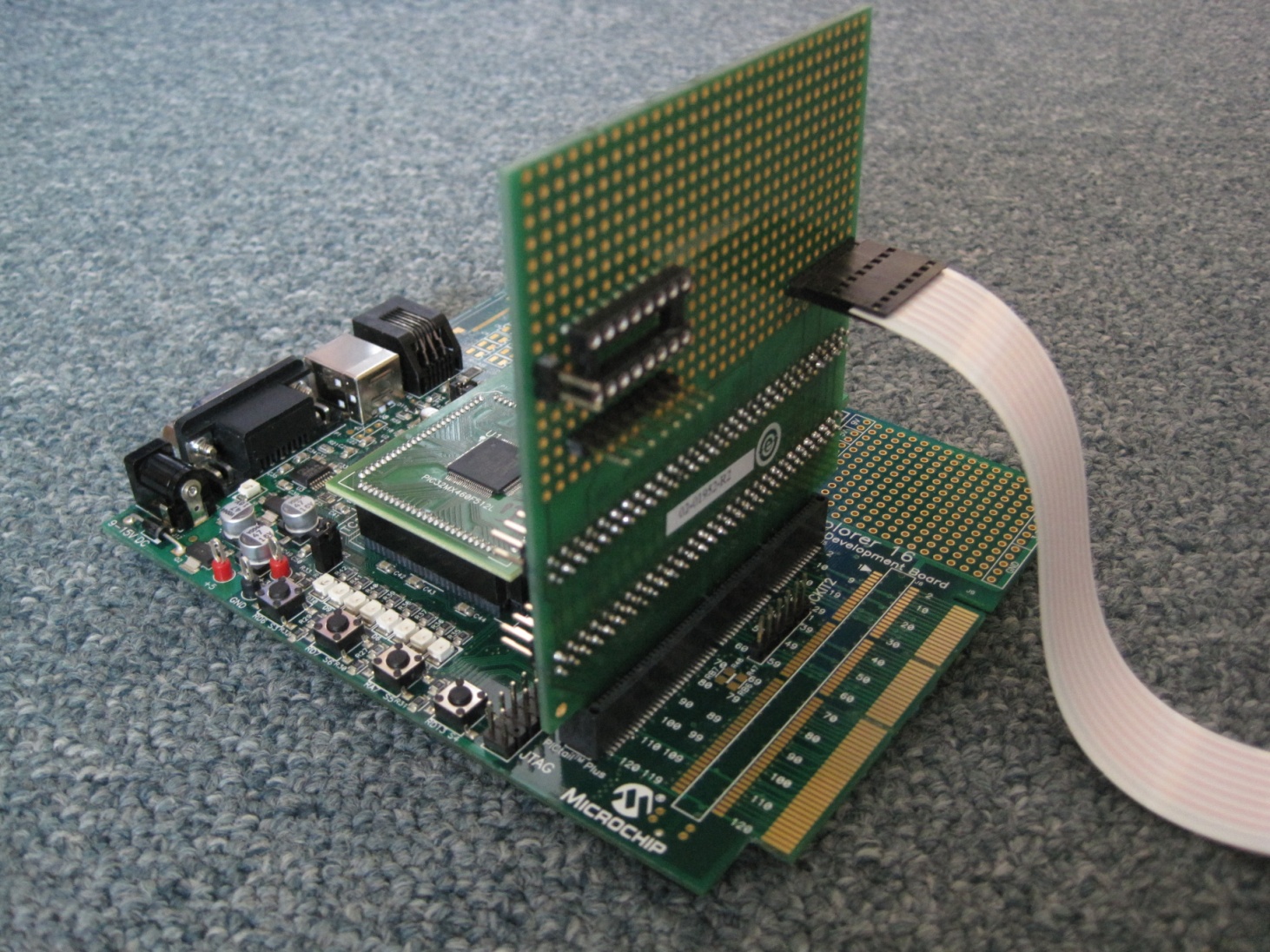
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **T** | **i** | **m** | **e** | **:** |  |  | **1** | **0** | **:** | **1** | **0** | **:** | **2** | **3** |  |
| **N** | **o** |  | **k** | **e** | **y** |  | **p** | **r** | **e** | **s** | **s** | **e** | **d** |  |  |

* 1. When a key is pressed, for instance at **ROW mask 0100** and **COL mask 0001**, the text should read:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **T** | **i** | **m** | **e** | **:** |  |  | **1** | **0** | **:** | **1** | **0** | **:** | **2** | **3** |  |
| **R** | **O** | **W** | **:** |  | **2** |  | **&** |  | **C** | **O** | **L** | **:** |  | **0** |  |

The ROW and COL numbers displayed are the binary codes for the *row and col masks*.

1. Hardware Issues:
   1. The keypad interfaces to the Explorer 16 board through a PICTail proto board with an 8-pin single row header. The PICTail board plugs into the Explorer 16 through J5. The keypad plugs into the PICTail board with the SILVER STRIP in the flat plastic cable position toward the INSIDE of the PICTail board. (Or, the outside COPPER STRIP of the flat cable is at the edge of the PICTail board.)



**To Keypad**

**Edge with Silver Strip**

* 1. The keypad interfaces to the PIC32 through eight port pins. Four pins will be output from the PIC32 and shall be used to energize the rows. Four pins will be input and will be used to sense the column position for a pressed key. The pin assignment is as follows:

Note: RB0, RB1, RG9, and RB3 will serve as inputs and require pull-up resistors for the floating or HI-Z case, i.e. no key pressed. These pull-ups will be available using the PIC32 “Change Notice” capability. RD0-RD3 will serve as open-drain outputs (drive 0 or floating) and should ideally have pull-ups as well. However, in this application, no pull-ups will be present on RD0-RD3 because no suitable Change Notice pin functions could be identified while considering future pin function usage. This is okay, though, since the floating case may cause additional power consumption but not right-out failure. If pull-ups “had to be present”, then external components would be required.

1

2

3

A

4

5

6

B

7

8

9

C

0

F

E

D

**(Row3) RD3 →**

**(Row2) RD2 →**

**(Row1) RD1 →**

**(Row0) RD0 →**

**↓ ↓ ↓ ↓**

**RB0 RB1 RG9 RB3**

**(Col0) (Col1) (Col2) (Col3)**

* 1. The **output pins** RD0, RD1, RD2, RD3, need to be configured with **OPEN-DRAIN** enabled. (See note above.) Again, this is accomplished through the TRISD and ODCD SFRs.
  2. The **input pins** RB0, RB1, RG9, RB3, need to be configured with:
* **“Analog input pin in Digital mode”**: You must modify the AD1PCFG SFR by setting the appropriate PCFGx bits. AD1PCFG is a SFR within the Analog-to-Digital (ADC) module within the PIC32. The PCFGx bits within the AD1PCFG SFR correlates to Analog input signals ANx (reference the pin diagram in the data sheet). Again, the ANx Analog input pin function is shared with a Rxx Digital pin function. However, a particular pin can only be used for Analog or Digital input. This lab needs Digital pins. AD1PCFG determines if a pin is used as an Analog input or a Digital I/O. The chip pinout (PIC32 DATA SHEET, page 6) indicates which Analog input signals share pins with which Digital I/O signals.

**For instance, the pinout indicates that Analog input signal AN0 shares the same physical pin as Digital I/O signal RB0. Therefore, PCFG0 within AD1PCFG must be set in order to make PIC32MX460F512L pin 25 a Digital I/O pin (RB0) versus and Analog input pin (AN0). At any time, a pin can only be either an Analog or Digital pin, but not both. In addition to RB0, locate the ANx signals for RB1 and RB3 as well, and set the matching PCFGx bits within the AD1PCFG SFR.**

* It turns out that RG9 is a Digital signal only and is not affected by AD1PCFG (Verify for yourself!). Again, you can use the AD1PCFGSET register address to selectively set individual bits at the same time.
* **“Port pin pull-up enabled”**: You must modify the CNPUE SFR by setting the appropriate CNPUEx bits. The CNPUEx bits correlate to the Change Notification signals CNx and they determine if a pin has an internal pull-up resistor present. The chip pinout (PIC32 DATA SHEET) indicates which pins have Change Notification capability – One Change Notification function being the internal pull-up capability. Locate the CNx signals for RB0, RB1, RG9, and RB3 and set the matching CNPUEx bits within the CNPUE SFR. Again, you can use the CNPUESET register address to selectively set individual bits at the same time.

1. Software Issues:

* Using Windows File Explorer, copy the working **lab2.X** project folder into a NEW **cst204:\labs\lab3\** folder. All of the files should go over verbatim. If you have trouble, start with a new **lab3.X** project and manually copy the Lab 2 files.
* Run MPLAB X and open the **cst204:\labs\lab3\lab2.X** project. **Within the Projects Tab**, right-click the Project Name and “Rename...” the **lab2** project to **lab3**, and, **CHECK the “Also Rename Project Folder” box**. This will result in **cst204:\labs\lab3\lab3.X**. It is not enough to just copy the files.
* The migrated files from **lab2** should be:
  + Within the **cst204:\labs\lab3\lab3.X\source** folder (Added to **Source Files** Project Folder)
    - * app\_clock
      * app\_heartbeat.S
      * config\_bits.S
      * data.S
      * hardware.S
      * lcd.S
      * main.S

Note that app\_test\_lcd.c is not required and should be removed.

* + Within the **cst204:\labs\lab3\lab3.X\library** folder (Added to **Libraries** **Files** Project Folder)
    - * lab1\_libs.a

(**Note: Library directory path** must be specified under File ⇒ Project Properties (lab2) ⇒ Categories: XC32 (Global Options) ⇒ xc32-ld ⇒ Option categories: Libraries ⇒ Library directories)

* Modify the following files to add the required items.
  + **hardware.S** will add the assembly code to configure the GPIO pins that interface to the keypad.
    - * **gpio\_config** – Add configurations to this *existing function*:
        + Configure pins **RD3**–**RD0** as open drain, digital output

1. Establish default **data value = 1111** (**PORTDSET**)
2. Establish **Open-drain** configuration (**ODCDSET**)
3. Establish **digital output** configuration (**TRISDCLR**)
   * + - * Configure pins **RB0(AN0),RB1(AN1),RB3(AN3)** as digital input
4. Establish **digital input** configuration (**AD1PCFGSET**). See Section 12.1.2, 12.1.3, TABLE 4-13 of PIC32 Data Sheet. In the TABLE, **PCFGx** correlates to **ANx**.
   * + - * Configure pins **RB0(CN2),RB1(CN3),RB3(CN5),RG9(CN11)** with Internal Pull-up Resistors
5. Establish **internal pull-up** configuration (**CNPUESET**). See Section 12.1.6 and TABLE 4-35 of PIC32 Data. In the TABLE, **CNPUEx** correlates to **CNx**.

* Add the following files to add the required items.
  + **keypad.S** will add the assembly code to implement the keypad “driver” functions.
    - * **keypad\_input\_pin** – Reads a ***column pin value*** by using the **current row mask** and **column mask** values. This is used in the “scan” process implemented in the **svc\_keypad** function defined below. It must be C compatible.

**Prototype**:

* + - * + **int keypad\_input\_pin (void);**

1. **Energize row by outputting PORTDCLR with keypad\_row\_mask**

**(PORTD = 1110 -–> 1101 -–> 1011 -–> 0111)**

1. **Input PORTB pins**
2. **Mask off all PORTB bits except RB0, RB1, and RB3 (AND with 0b1011 = 0xb)**
3. **Input PORTG pins**
4. **Mask off all PORTG bits except RG9 (AND with 0b1000000000 = 0x200)**
5. **Right shift 7 RG9 places**
6. **Compose RB3-RG9-RB1-RB0 bits (in this order) by ORing (4) and (6) results**
7. **Sense row/col pin value by ANDing Composite RB3-RG9-RB1-RB0 bits with keypad\_col\_mask**
8. **Set v0 = 0 if Composite = 0, else v0 = 1**
9. **Deenergize row by outputting PORTDSET with 0b1111 = 0xf**
   * **svc\_keypad.c**
     + - **void svc\_keypad(void)** – This performs the keypad scan and updates the **KEY\_PRESSED** global variable appropriately. It will do so by implementing the following state machine specified below.

Modify **data.S** to add the global variables: **keypad\_row\_mask**, **keypad\_col\_mask**, **svc\_keypad\_state**, **KEY\_PRESSED** (Add and organize in **data.S**.)

**Col pin = 1**

**Col pin = 0 / KEY\_PRESSED = 1**

**Col pin = 0**

**Col pin = 0**

**Col pin = 0**

**Col pin = 1**

**Col pin = 1 / KEY\_PRESSED = 0**

**Col pin = 1**

0

1

2

3

**{STATE-Debounce “0”:**

**- Check column}**

**{STATE-Debounce “1”:**

**- Check column}**

**{STATE-Key Pressed:**

**- Check column}**

**{STATE-Key Released:**

**- Advance row mask**

**- Check all columns}**

**State Machine Summary:**

**State 0**: {In this state because all keys have been released. Its purpose is to detect a potential key press.} Advance the **keypad\_row\_mask** value. Loop through the four **keypad\_col\_mask** values, calling **keypad\_input\_pin** each time until either **keypad\_input\_pin** = 0 (a key is found pressed) or all four column pins have been checked.

**State 1**: {In this state because a column pin was found pressed 10 ms ago, i.e. last RTE loop. Its purpose is to detect if a key is “actually” pressed after being detected, i.e. checking AFTER debounce delay.} Call **keypad\_row\_mask**. Note that the call will make use of the current **keypad\_row\_mask** and **keypad\_col\_mask** values. If **keypad\_input\_pin** = 0, the key is assumed pressed, otherwise, it is assumed to be not pressed.

**State 2**: {In this state because a key has been sensed pressed. Its purpose is detect a potential key release. While in this state, all other keys are locked out.} Call **keypad\_row\_mask**. Note that the call will make use of the current **keypad\_row\_mask** and **keypad\_col\_mask** values. If **keypad\_input\_pin** = 1, the key is assumed to be released, otherwise, it is assumed to be still pressed.

**State 3**: {In this state because a column pin was found released 10 ms ago, i.e. last RTE loop. Its purpose is to detect if a key is “actually” released after being detected, i.e. checking AFTER debounce delay.} Call **keypad\_row\_mask**. Note that the call will make use of the current **keypad\_row\_mask** and **keypad\_col\_mask** values. If **keypad\_input\_pin** = 1, the key is assumed released, otherwise, it is assumed to be still pressed.

* + **app\_test\_keypad.c**
    - * **void app\_test\_keypad(void)** – will continually poll the **KEY\_PRESSED** global variable and output messages on LINE 2 of the LCD according to the description specified earlier in this document. You will use the **LCD\_BUFFER[]** array previously defined. It will do so by implementing the following state machine:

Modify **data.S** to add the global variable: **app\_test\_keypad\_state**

**KEY\_PRESSED = 0 / Output KEY PRESS msg**

**KEY\_PRESSED = 1 / Output KEY PRESS msg**

**KEY\_PRESSED = 0**

**KEY\_PRESSED = 1**

1

0

# Completion of Lab 3

* Demonstrate operation to the instructor.
* Zip the **cst204:\labs\lab3** folder and rename it **cst204:\labs\lab3.zip** and upload.