LAB 5: MATRIX KEYPAD INTERFACE USING PARALLEL I/O

THE VELLEMAN 16-KEY MATRIX KEYPAD

The purpose of this lab is to use the MC9S12C32 (HCS12) microcontroller to control a peripheral device, interfaced through parallel I/O ports and accessed using either program-controlled or interrupt-driven I/O. The peripheral device for this lab is a matrix keypad, as used on a variety of products (phones, keyless entry systems, appliances, etc.) The keypad is pictured in Figure 1. Note that there are 8 pins on the bottom of the keypad, with pin spacing ("pitch") that will allow the pins to be inserted into a standard breadboard. This keypad closely resembles the simulated keypad in *CodeWarrior* that was used in ELEC 2200 projects. **Refer to the keypad scanning example in Chapter 18 of the Cady text book and homework projects from ELEC 2220.**

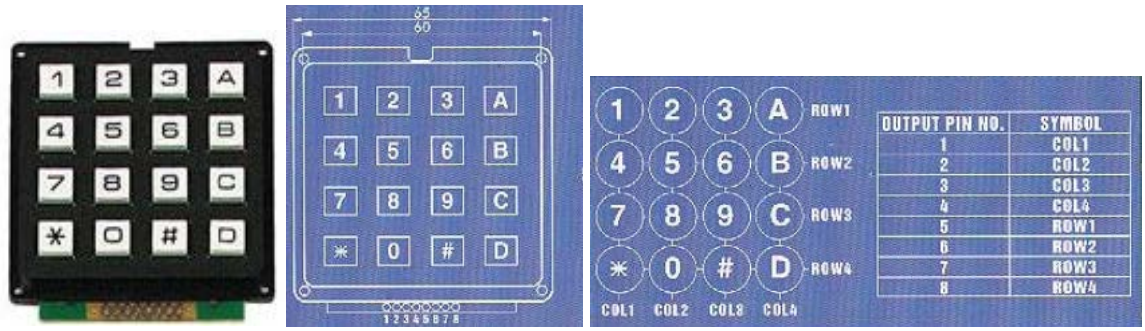

Figure 1. Velleman 16-Key Matrix Keypad

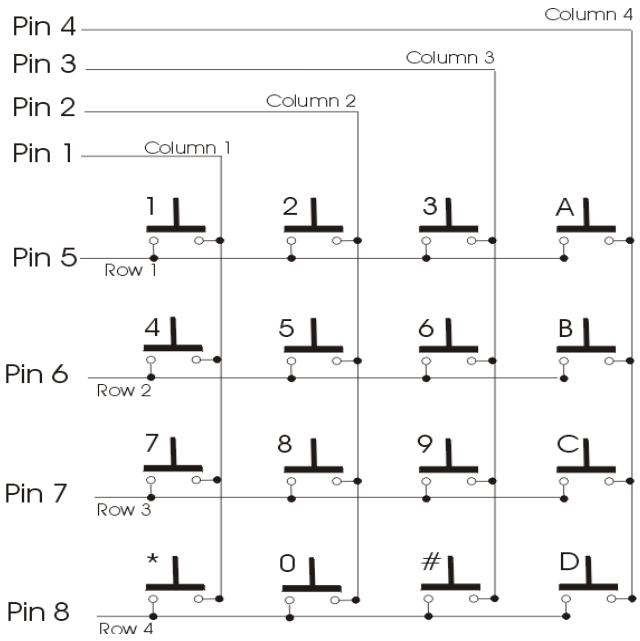

Figure 2. Matrix Keypad Circuit Diagram

The keypad's electrical circuit, shown in Figure 2, contains no "active" electrical components. It is simply a matrix of wires and switch contacts, arranged in rows and columns, with a key and a pair of contacts at each row-column intersection. Springs normally hold the keys away from the contacts, resulting in open circuits between the row and column wires. When a key is pressed, the contact closure connects the corresponding row and column wires, creating a short circuit between them. When the key is released, its spring pulls it away from the contacts, restoring the open circuit. Detection of a pressed key thus requires detection of a short circuit between a row wire and a column wire, with the key number determined from the row and column numbers.

INTERFACING THE MATRIX KEYPAD TO THE MICROCONTROLLER

To detect a pressed key and identify the key number, the computer must "scan" the keypad to look for a short circuit condition between a row and a column wire. If a short is detected, the computer must determine which row and column wires have been shorted together to identify the pressed key. The most common way to scan a matrix keypad is to set all row and column wires at the same voltage, V1, and then drive one of the column wires Ci to a different voltage, V2. This is illustrated in Figure 3, in which columns 1, 3 and 4 are driven high, with column 2 driven low; the four row lines are normally held at logic 1 with pull-up resistors. If switch 5 is pressed, column 2 is shorted to row 2, resulting in the 0 on column 2 forcing row 2 low. If no switch is pressed, all rows remain pulled up to logic 1. (The roles of the row and column wires can be reversed if desired.)

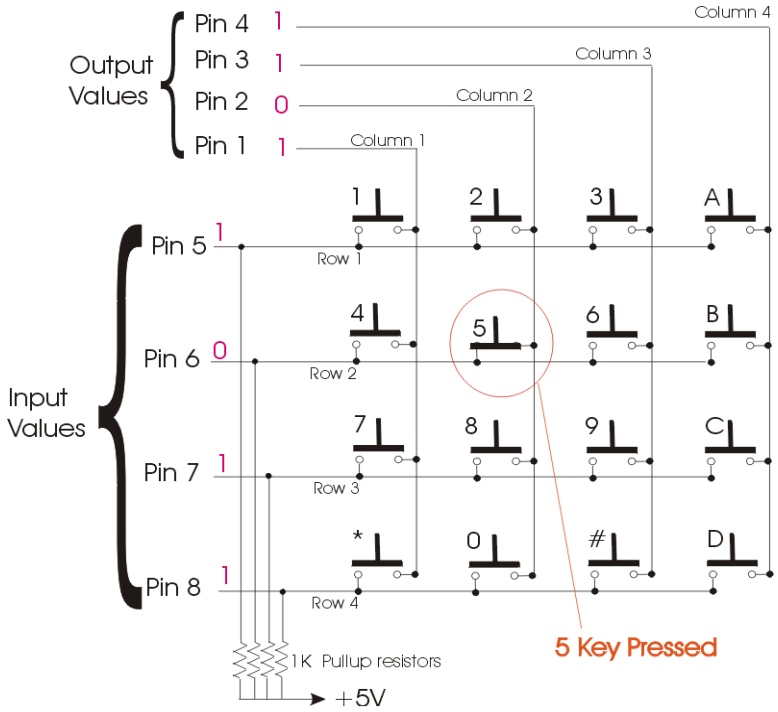

Figure 3. Keypad scanning example

To allow the keypad to be scanned as described above, four bits of a parallel port must be set up as outputs to drive the column lines, and four bits of a parallel port must be set up as inputs and then read to check the states of the row lines. In this configuration, the voltages used are V1 = +5 volts (logic 1) and V2 = 0 volts (logic 0). Each of the four row lines should be connected to +5 volts through a pull-up resistor of about 1K ohms to pull the line up to the logic 1 state, or else the internal pull-up resistors in the HCS12 activated on the parallel input port pins connected to the row lines (refer to Section 11.7 of the Cady text, or the MC9S12C32 data sheet for a description of pull-up control.) The column lines are driven by writing the desired pattern to the output port. When not scanning the keypad, the column lines would normally be driven to their "inactive" (logic 1) states. To scan the keypad, one of the column lines must be "activated", i.e. driven to the logic 0 state, with the other column lines driven to their inactive (logic 1) states. Since the row lines are pulled up to logic 1 by pull-up resistors, forcing one of them to the logic 0 state can be done only by shorting the row line to a column line that has been driven to the logic 0 state. Therefore, when reading the states of the row lines via the input port, if the state of any row line is logic 0, it must be shorted to the activated column line. Note that if a row line is shorted to an inactive column line (in the logic 1 state), then the row line will remain in the logic 1 state.

The keypad scanning process requires activating one column, with the other three deactivated, and sampling the row lines. If any row is in the logic 0 state, then a key is being pressed at the intersection of that row and the activated column. If the rows are all high, then no key is being pressed in the activated column. This process is repeated in a continuous loop, activating each column, with the others deactivated, until a pressed key is detected.

If interrupt-driven operation is desired, an interrupt signal should be generated whenever a key is pressed on the keypad. This can be done by using the logical OR of the *active-low* keypad row lines, as shown in Figure 4, to activate the ***IRQ#*** pin; the ***IRQ#*** pin should be configured for edge-triggered operation. Figure 4 shows a 4-input AND gate - DeMorgan-equivalent logic symbol. ($ABCD = \overline{\overline{A} + \overline{B} + \overline{C} + \overline{D}}$) . This will cause ***IRQ#*** to go low whenever any row line changes to 0. In this case, all column lines should be driven low to allow any key press to pull one of the row lines low, triggering the interrupt. Then the scanning procedure described above would be executed to determine which key has been pressed.

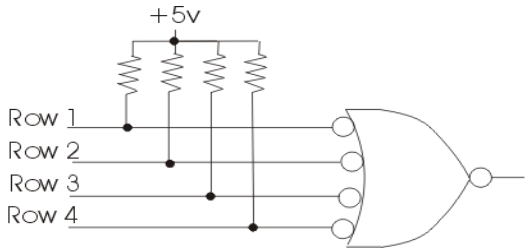

Figure 4 – Logical OR of active row lines (produced by AND gate for active-low signals)

Note that a pressed key can "bounce" and that it can remain pressed for an arbitrary amount of time. The CPU should be interrupted when the key is first pressed, and should

not respond to additional interrupts until the key stops bouncing and returns to its inactive state. If debouncing is necessary, the I (interrupt mask) flag can be set to 1 to disable interrupts for a designated period of time (using a delay loop, for example), and then reset to 0 to re-enable interrupts.

LAB OBJECTIVE

For this lab, the "main program" is to display a binary number on 4 LEDs, with the number incrementing approximately once per second in an endless loop (the time does not need to be exact for this exercise). If a key is pressed, as detected either by a software test or via an interrupt, the keypad scanning routine should identify and display the key number on the LEDs, instead of the count. This key number should remain on the LEDs for approximately 10 seconds, and then the program should resume displaying the incrementing count on the LEDs. The counter should not stop during these 10 seconds, so when counter display resumes, the count should be approximately the time of the interrupt plus 10. (Hint – if using interrupts, the interrupt service routine can set a global variable that can be tested by the main program to see if a key number is being displayed.)

PRE-LAB ASSIGNMENT

Reading

Review section 18.2 of the Cady text (2nd edition), which discuss simple input devices, including an example of a 16-key matrix keypad. If using interrupts, review the material from the previous lab session.

Hardware Design

Parallel input and output ports T and AD are to be used for the keypad interface. The only additional hardware required is the keypad. 10K pull-up resistors for the keypad row lines can be used if desired, but it would be preferable to use the internal pull-up resistors of the HCS12. The I/O port connections to the "peripheral devices" should be made as listed in Table 1. Note that Port T drives two "devices" – outputs to the keypad column lines and inputs from the keypad row lines. Port AD is to drive the 7-segment display decoder. If using interrupt-driven operation, a 3-input AND gate (7411) will be needed to combine the row signals to drive the ***IRQ#*** line, which is also bit 1 of Port E. You may use additional LEDs for debugging, if you wish.

Parallel Port Pins	Connected Devices
Port AD, bits 0-3	LEDs
Port T, bits 7-4	Keypad column lines 4-1 (outputs)
Port T, bits 3-0	Keypad row lines 4-1 (inputs)
Port E, bit 1	IRQ# (if interrupt used)

Table 1. Parallel input/output port connections.

To aid in debugging, be prepared to use the logic analyzer and/or oscilloscope to observe the states of the various peripheral device lines.

Software Design

Review the earlier labs to recall how to initialize and access I/O ports in C, and to set/clear/test individual bits of a byte. Thoroughly comment your program to demonstrate your understanding of the keypad scanning operation.

Your test program should execute in a continuous loop. If using program-controlled I/O, the main program should continuously scan the keypad for pressed keys and periodically update the count shown on the 4 LEDs. If using interrupt-driven I/O, the main program should only update the LEDs, with the keypad scanning routine executed only if interrupted by a pressed key. If a pressed key is detected, the key number should be displayed on the LEDs, instead of the count, for approximately 10 seconds. Refer to the discussion above for a description of the keypad scanning process.

If using interrupts, the test program should comprise a main program and an interrupt service routine, both written in C. The main program should initialize the column lines of the keypad, enable interrupts, and then enter a continuous “do-nothing” loop. Note that all four column lines should initially be driven low so that any pressed key will cause one of the four row lines to go low and trigger an interrupt.

LAB PROCEDURE

1. Double-check the power supply connections (+5v and ground) between the DragonFly12 module and the EEBOARD.
2. Connect the Port AD pins to the DIGITAL I/O pins 3-0, as indicated in the table above.
3. Mount the keypad on your breadboard and connect its columns to PT7-PT4 and its rows to PT3-PT0, as indicated in the table above. Ideally, you should activate the internal pull-up resistors in the input port in software. However, if you choose to utilize external 1K resistors (provided) as pull-ups, connect those.
4. Verify that can access the I/O ports properly, using the test methods learned in previous labs. If you experience problems, you should test the I/O ports using the test programs from Labs 2 and 3 and the logic analyzer.

5. Enter, compile and download your keypad scanning program. Execute your program, verifying that correct key number is displayed for each of the 16 keys.
6. Demonstrate your working programs to the lab instructor.

LAB REPORT

1. Briefly describe your hardware design. Include a schematic diagram.

2. Include a printout of your C program, including well thought through comments.

3. Provide a logic analyzer screen capture showing where the LEDs changed from an incrementing count to a key number.

4. Discuss your results.