Keypad Interfacing
Keypad

- Typical keypads have 12 or 16 buttons

- A mechanical keypad simply consists of a set of vertical wires (one for each column) and a set of horizontal wires (one for each row)

- When you press a button, you press the row wire onto the column wire, shorting the two together, thus making an electrical contact

There are 7 pins, corresponding to the 3 columns and 4 rows
To test if a key is pressed, we simply test to see if there is a connection between a row and a column.

For example, if pin 2 is electrically connected to pin 3, then we know key “1” is pushed.
Approach

• Scanning
  – We scan through the rows and columns (very quickly) to see if any key is being pressed

• Issue: debouncing
  – Mechanical switches “bounce” slightly before coming to rest; we have to ignore the bounces

Switch output signal

5 to 20 ms
• Set up 3 pins for output; 4 pins for input
• Then scan thru columns, outputting a low on each in turn
• For each column, scan thru rows
• If no key is pressed, input is high; otherwise is low
• If low, key code (1..12) is determined by (row,col)

Need pullup resistors - any value is ok if it limits the current to < 25 mA
Debouncing

- Debouncing can be handled in hardware or in software

- A software solution is “wait and see”; namely, if you detect a key press, wait 10 ms and see if it is still pressed

Figure 7.42 Hardware debouncing techniques
function getkey()
    for each column (0,1,2)
        output a low to that column’s pin

    for each row (0,1,2,3)
        read pin for that row
        if pin is low
            wait 10 ms and check again
            if pin is low
                return the code corresponding to
                the key at (row, col)
            end
        end
    end
end
return 0 to indicate no key is pressed
• It’s not a good idea to have a circuit that can be damaged if you have a bug in the software

• For example, what if you accidentally made PT3 an output?

• Even if you don’t have a bug, what if a person presses two keys at the same time?
Current limiting resistors

- Let’s add small current limiting resistors on each pin in case we make a mistake in the software and accidentally make a pin an output; or a person presses two keys simultaneously.

- The resistor values should be high enough so that current is less than 25 mA.

- \( \frac{5V}{R_1+R_2} < 25 \text{ mA} \)
  - or

- \( R_1 + R_2 > 200 \text{ ohms} \)
- Will R1, R2 affect logic levels?
Logic Levels

• Standard CMOS logic levels are:
  – Input range is 0V to 1.5V for logic low
  – Input range is 3.5V to 5.0V for logic high

• So we need to make sure that our inputs are in this range

• We don’t want voltage drops across R1 and R2 to cause invalid logic levels

• Consider resistor R2
  – It is on the input to the MCU
  – The input is high impedance; very little current flows through R2
  – So very little voltage drop across R2; its value can be large or small
Logic Levels (continued)

- R1 may affect the logic level

- Consider what happens with switch is closed and pin PTx is outputting low (V=0)
- We have a voltage divider with R1 and the 10K pullup
- We want the output voltage to be a valid logic low (i.e., 0V to 1.5 V)

- So
  
  \[ (5V)(\frac{R1}{R1+10K}) < 1.5V \]
  
  \[ R1 < 4.3K \]
Consider physical layout

- The keypad can plug right into the protoboard

- Let’s plan on using a resistor DIP pack (1.5 Kohm) to connect the Port T pins on H1 to the keypad pins

- Instead of using PT0:PT3 for input and PT4:PT6 for output, we might want to pick different pins to simplify the wiring

You will also need the pullup resistors (10Kohm DIP pack)
• Use these connections to simplify wiring

• Pin numbers on header (H1) and keypad (KP)

<table>
<thead>
<tr>
<th>Connections</th>
<th>H1</th>
<th>KP</th>
</tr>
</thead>
<tbody>
<tr>
<td>PT0 19</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>PT1 20</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>PT2 21</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>PT3 22</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>PT4 23</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>PT5 24</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>PT6 25</td>
<td>7</td>
<td></td>
</tr>
</tbody>
</table>

**Connections**

1.5K

*pin numbers on H1 header*
One more issue

• If the user presses and holds a key down, this must be interpreted as a single keystroke
  – Must not only look for key depressing but also for key releasing
  – We can do this by keeping track of the last key pressed

• The function “getKey” can save the last key pressed in a static variable called “lastKey”
  – If the same key is being pressed, then don’t treat as a new key; just return zero
  – If a new key is being pressed, then return that keycode and save it in “lastKey”
  – If no key is being pressed, then save a zero in “lastKey”
C Code for function

• Function header and comments

```c
// Get a keypress (if any) from the keypad.
// If a key is pressed, it returns its code, which is an integer
// from 1 (top left) to 12 (lower right).
// If no key is pressed, it returns 0.
// If the same key is still pressed from last time, return 0.
char getkey(void);
```

• We will also need these (global variables)

```c
// Define the Port T bits corresponding to columns
char columnBits[] = {
    0x04,   // column 0 (left) is pin PT2
    0x01,   // column 1 (center) is pin PT0
    0x10    // column 2 (right) is pin PT4
};

// Define the Port T bits corresponding to rows
char rowBits[] = {
    0x02,   // row 0 (top) is pin PT1
    0x40,   // row 1 (2nd) is pin PT6
    0x20,   // row 2 (3rd) is pin PT5
    0x08    // row 3 (bottom) is pin PT3
};
```
char getkey(void) {
    char col, cbit, row, rbit, key;
    static char lastKey = 0;

    // Scan across columns 0,1,2
    for (col=0; col<3; col++) {
        // Write a 0 to the bit corresponding to the column;
        // write 1 to all other bits in Port T
        cbit = columnBits[col];
        PTT = ~cbit;

        // Scan across rows 0,1,2,3
        for (row=0; row<4; row++) {
            // Get the mask corresponding to this row
            rbit = rowBits[row];

            // Read Port T and check only that bit; a key is
            // pressed if the bit is 0.
            if (!(PTT & rbit)) {  // A key is pressed
                DelayuSec(10000);
                if (!(PTT & rbit)) {  // check same key again
                    key = keycodes[row][col];  // get keycode
                    // If this is still the same key, return 0
                    if (key==lastKey) return(0);  // It is a new key being pressed
                    lastKey = key;
                    return(key);
                }
            }
        }
    }
    lastKey = 0;
    return(0);
}

// We got to here, so no key is being pressed
lastKey = 0;
return(0);

Notes

• “Static” means that the variable retains its value after the function exits
• It is similar to a global variable, except that it only visible within this function
• keycodes[r][c] is an array that contains the code for each key, at row r and column c
• Could just be the numbers 1..12, or anything you want
Summary / Questions

• A mechanical keypad is a very simple device, and thus low cost and rugged. We make up for the simplicity using software, to create a sophisticated input device.

• Keypads can also use capacitive switches or Hall-effect switches (with similar issues in interfacing)

• If you only needed a few of the keys on the keypad, can you design a simpler interface?