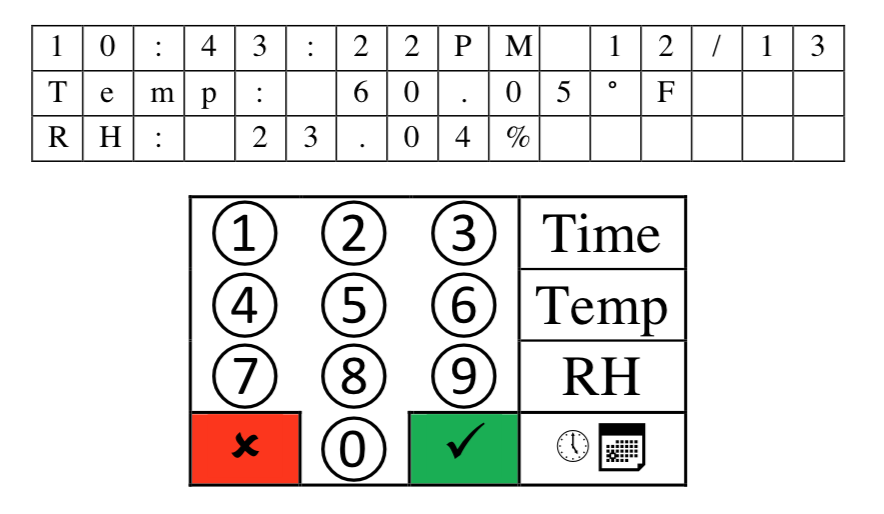
# **Chapter 2: UI and Interaction**

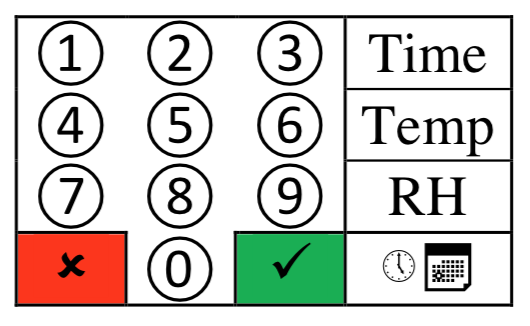
### **A. Front View of Unit**



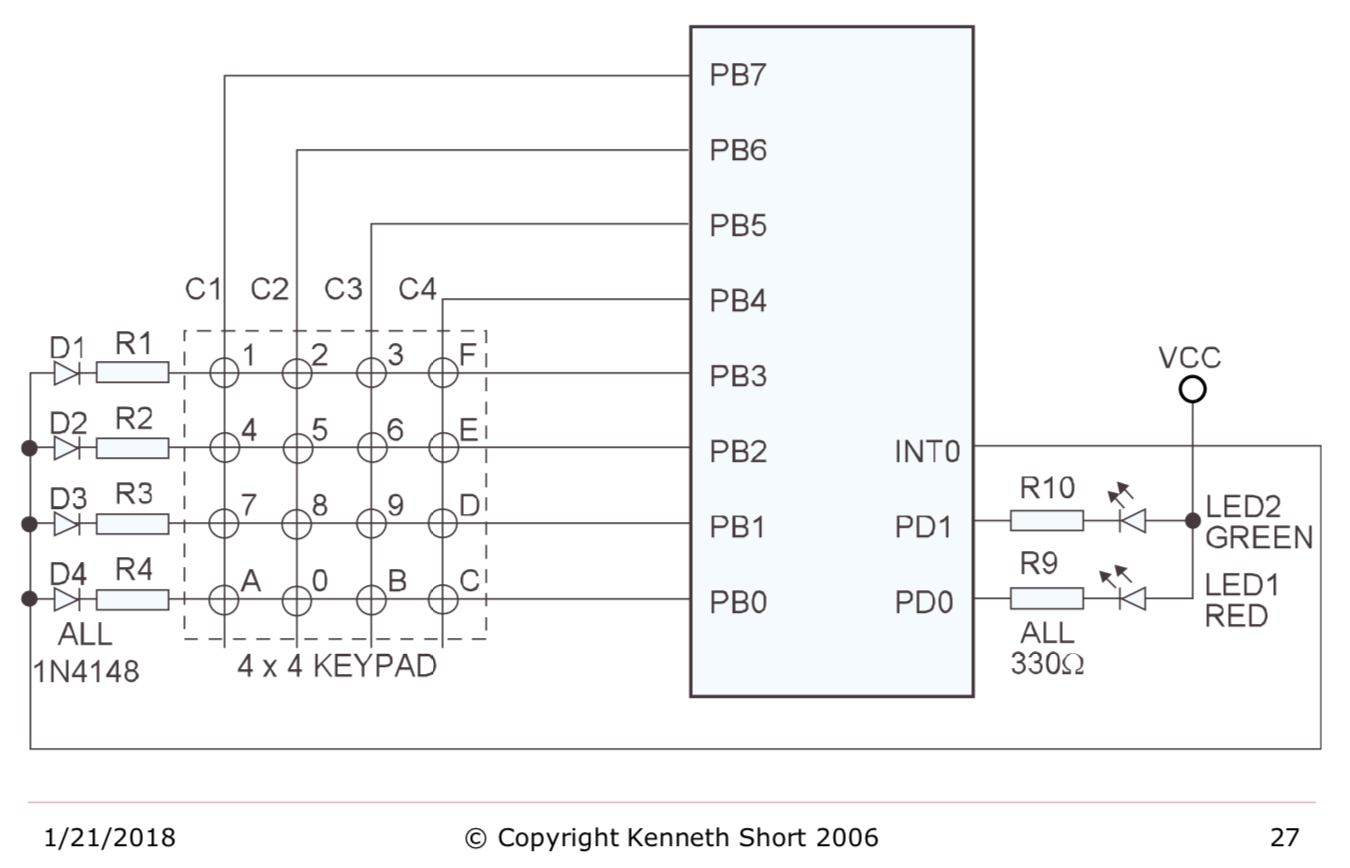
Here we can see the front panel of the product. On top we have the LCD which is composed of 3 lines of 16 characters each. Below the LCD is the 4x4 Keypad which is the main way for the user to input data into the product. Here the number buttons represent numbers and the buttons on the right most columns starting from the top represent, time, temperature, humidity and alarms.

### **B. Keypad**

The keypad found in this product is essentially a 4x4 matrix which results in 16 buttons. It is important that we have a 4x4 matrix because it allows us to minimize the number of pins we need to use.



In a 4x4 matrix we only need to use 8 pins if we connect the keypad to the microcontroller using the matrix approach. This is because when we use an unencoded matrix the number of ports needed to support all buttons in the matrix is equal to the addition of the number of rows and the number of columns. Compare this to a linear unencoded approach, which would result in using 1 pin per button. This would mean that in order to support 16 buttons a linear unencoded approach would need 16 pins of the microcontroller, on the other hand the matrix approach only needs 8 pins to support all 16 buttons.



The way a matrix approach on a 4x4 keypad is depicted above, courtesy of Professor Kenneth Short. Out of the 8 pins we are provided at Port B we assign have of the pins to be outputs PB4,PB5,PB6,PB7 and the other half as inputs PB3, PB2, PB1, PB0. We can see that the columns are connected to the input pins and the rows are connected to output pins. The intersections on this matrix are where the buttons would be.

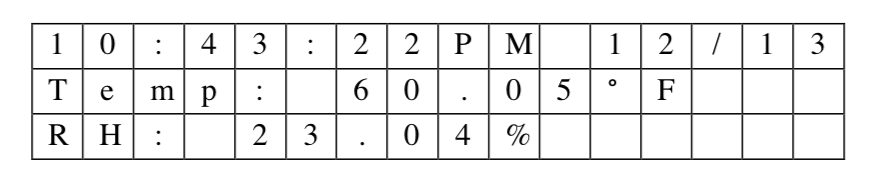
When a user presses down on a button one of the input pins will register a HIGH signal. Let us say that we pressed down on button 9. This would result in PB1 registering a HIGH, so we have identified the row in which the button that was pressed resides. So how do we identify the column in which the button resides. This can simply be done by only allowing one of the output ports to be HIGH at a time. This way, when we register a HIGH in the input we know what row we are in as well as which columns since only one column is allowed to be HIGH at a time. The microcontroller much faster than the user so it will never not register that the user has pressed a button been though only one column is HIGH at any given time.

A problem that was addressed in the design of this product in regard to the keypad was the phantom key problem. In the phantom key problem is as follows, let us say that buttons 6, E, and D are pressed, and that the column C3 is HIGH. The microcontroller will read PB2 and PB1 are both HIGH. This is because buttons 6 is connected to button E which is connected to D and hence both rows will drive a HIGH. However, because only column C3 is high the microcontroller will assume that buttons 6 and 9 are being pressed. Then when it goes to make C4 HIGH it will register that buttons E and D are HIGH. Resulting in the phantom key 9 being detected when it was not pressed. The way to solve this problem is to add a diode at every intersection of the buttons to prevent the current from flowing to buttons that have not been pressed.

### **C. LCD display**

The Liquid Crystal Display (LCD) found on this product consists of 3 lines, of 16 characters per line, LCD panels and a CMOS controller driver which is mounted to the pc board that the LCD is mounted to.

LCDs are among the most cost-effective ways to add a display output to an embedded system.



The specific LCD module found on this product is the Electronic Assembly (EA) DOGM163W-A. This module has a Sitronix ST7036 Dot Matrix Controller Driver incorporated into its pc board. This driver allows us to set up an SPI connection between the driver and the microcontroller. This SPI connection lets the microcontroller control the LCD and have it display characters without having to specifically define the dot matrix sequence of the characters. This is because the controller driver has all the dot matrix sequence for all alpha-numeric characters predefined. So when the microcontroller tells the controller driver to print “X” the controller driver search its memory for the dot matrix of “X” and sends this dot matrix sequence to the LCD which then displays the “X”.

We connect the microcontroller and the controller driver by using a 10-pin header and a flat ribbon cable. This was done in order to avoid having messy wiring. We used the Serial Peripheral Interface (SPI) as the communication protocol between the microcontroller and the controller driver. SPI has the advantage of being supported by this LCD module as well as being a synchronous serial communication with full duplex. Meaning that we require a common clock between both devices and that both devices can transmit information simultaneously. Some may argue that SPI is too slow and that instead we should have used a parallel communication interface. However, one must note that the LCD module is very slow. For example, the execution time of an instructions can range from around 40 microseconds to about 1.64 milliseconds depending on the instruction. We can definitively say that using a parallel interface will not improve the speeds at which we can display new data on the LCD since the bottleneck is the ST7036 not the communication interface that we were using. However, if we had used a parallel interface then we would use more pins on the microcontroller which adds cost to the whole project. For this reason, we used SPI.

It is important to keep in mind that the LCD module must be initialized every time we power on the device. For this reason, we created a software subroutine init\_lcd\_dog. This subroutine essentially sets up the SPI. We select the DOG module using the Slave Select pin, we set up MOSI in order to send data to the LCD. We set our common clock to be SCK.

Once we have initialized the LCD we have to create a subroutine to update the display, called update\_lcd\_dog. This subroutine allows us to display the contents of our SRAM memory arrays, dsp\_buff\_1, dsp\_buff\_2, dsp\_buff\_3 on the LCD display. We have 3 memory arrays because we have 3 rows on our LCD, and each of these arrays is of size 16 because each line in our LCD has 16 character spaces.

With these two methods we are able to send messages to the LCD from the microcontroller. Due to the nature of both of these methods, they can be used to display any message on the LCD. This is because neither method creates the message that is to be sent they simply set up communication. This means that one could re-use these methods and add to the product more sensors and have their values displayed on the LCD by simply passing the proper buffer array which is created in the microcontroller.

### **D. Back View of the Unit**



Here we see the back panel of the product. It is composed of an On/Off switch at the top left corner. On the bottom left corner, we see a Reset button which is how the user can reset the system without having to power off the product. Lastly on the bottom right we have a power supply which connects to 120 V AC.