Hochschule Konstanz
Department of Computer Science

Josef Müller
Ubiquitous Computing Lab

Degree: Applied Informatics (AIN)
Lecture: Ubiquitous Computing

Report for the Laboratory 1:

Arduino

1 Getting started

1.1 Implement a blinking LED

In this task, a circuit for an LED should be built and a code that makes it blink should be programmed. An LED is digitally defined by two states: HIGH and LOW. If the LED should be on, then the PIN to which it is connected is set to HIGH. To switch off the LED, this PIN is set to LOW.

The time for the blinking of the LED can be controlled in different ways: For example, a button can turn on the LED when it is pressed. For this the PIN for the button is set to the value PULL_UP. This causes a current to flow at the button PIN, which is then interrupted when the button is pressed. This state is read continuously in the loop(). As soon as the button PIN is LOW, i.e. the current is interrupted, the LED should flash. In this condition the LED pin is set once to HIGH and after a delay to LOW. So the LED flashes when the button is pressed. Also a timer can make the LED blink after a certain time. For this a time x is saved when the program is started. After each loop passes, it is then checked whether the current time y is greater than the starting point x in addition to an offset. If this is the case, the code jumps to the condition that leads to the flashing of the LED, which updates the time x and makes the LED flash.

Instead of flashing an LED, you can also fade it. The LED is no longer defined by the HIGH and LOW states, but by values from 0 (LOW) to 255 (HIGH). This leads to the fact that finer gradations of brightness can be achieved. To let the LED fade, it must be controlled by a PIN that gives an output as a Pulse-width Modulation (PWM). It is still a digital output, but the proportion and length of LOWs and HIGHs within a second determines the brightness of the LED.

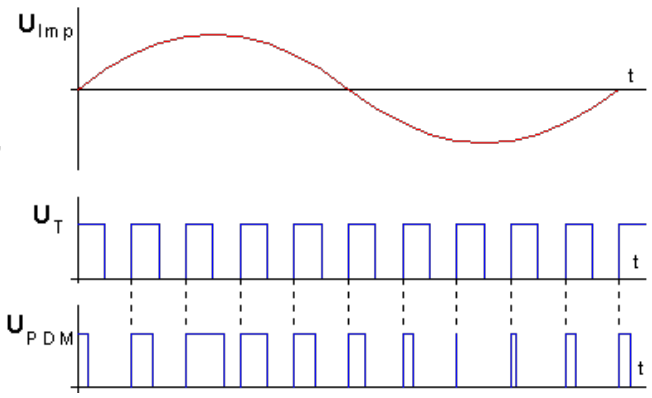

Figure 1: Example of PWM

Similar to the blinking LED, the state of two buttons is polled: one to increase the brightness and one to decrease it. If one of the buttons is pressed, a value is counted up or down from 0 to 255. This value is written analog to the PIN. With PWM the share of HIGHs and LOWs defines how bright a LED is perceived. At medium brightness the share of both LOWs and HIGHs is 50 %. The higher the value, the higher the share of HIGHs and the more often the LED is switched on within one second. Proteus is unfortunately not able to display this state realistically and lets the LED flicker. In fact, our eye should not even notice that the LED is switched on and off several hundred times a second.

1.2 Implement a buzzer

A buzzer is a component that starts vibrating at a certain frequency when a voltage is applied. This allows tones to be perceived.

When implementing the buzzers, it was useful to describe the tones as notes instead of the frequency. For example, 440 Hz corresponds to an A. This information was defined in a separate .h file.

The melody and the length of each note were saved in two separate arrays.

The buzzer was connected to a digital PIN and to ground. By means of the "Tone" library the buzzer can be controlled. The "tone" and "noTone" commands are used to tell the buzzer whether it should play at a certain frequency or not. Similar to the "fading LED" the buzzer is controlled via PWM. This means that different tones can be played depending on the number and length of the HIGHs and LOWs.

The difference to simply turning it on and off is that the buzzer does not have only two states. It can play either a sound or no sound, but in different frequencies. To be able to play the frequencies, the PWM is used. The blinking LED doesn't need such a thing, because it only has the HIGH and LOW states. On the other hand, the fading LED also uses PWM to represent the different brightness levels.

The buzzer plays sounds as soon as a button is pressed or a timer expires. It is necessary that the timer does not block, so that the state of the button can be queried. How the button works is described in the previous subsection and therefore not explained again. As soon as the current from the button is interrupted, the Arduino jumps to the fulfilled condition and plays the melody. The same happens as soon as the timer expires. The non-blocking timer is also explained in the previous chapter.

1.3 Merge previous ideas

The idea was that a LED should blink as soon as a sound is playing. This could be useful if you want to send Morse signals / telegrams for example. The receiver would thus not only receive an acoustic signal, but also a visual signal. This way, if one component fails, the other could continue to transmit the signal.

It was realized by connecting a simple LED to an output PIN. This was turned on when the "tone" command was executed and turned off when the "noTone" command was executed. Unfortunately this implementation did not work. The LED stayed on continuously while playing the melody and turned off when finishing.

2 Matrix: How to work with a LED matrix

In this assignment, you were allowed to experiment with matrices and implement various use cases. The difficulty in this task was to deal with the various controllers and libraries that can help to control matrices. The libraries "*MAX_72XX*", "*MD_PAROLA*" and "*LEDControl*" were used in these tasks. Each library has its own area of application. For each task, it was therefore necessary to consider which library would be useful.

2.1 Matrix (Random) Counter

For the matrix counters, two 8x8 pixel matrices were to be used to display numbers. Random numbers were to be displayed in the first part and numbers from 0 to 128 in the second part. There were two possibilities for this implementation: Each number is displayed on a separate matrix or all matrices are combined to display one number.

For the first part, this question was irrelevant, as only two-digit numbers had to be displayed. For the second part, however, it made sense to ask this question. The second option, i.e. merging the matrices, proved to be easier to implement.

For both parts the library "MD_Parola" was used, because in contrast to "LEDControl" it was not necessary to define each number as a matrix of ones and zeros.

In the first part, each loop called a small function that generated two random numbers from zero to nine and packed them into a string. Each digit was then displayed on a separate matrix.

This creates an instance of "MD_Parola" whose functions can be called. The instance is given the hardware type of the controller to control the matrices when it is initialized. In this case it is the MAX7219. The instance allows many settings - text size, font to displaying moving text.

The random counter could be used as a password generator, for example, on a keychain. Indeed, it is common to see key fobs that generate a login password that can be used to log into accounts.

The second part was to implement a counter from 0 to 128. Here the thought was to pack the matrices together and display the numbers more compactly. "MD_Parola" can display text across multiple matrices. For this the instance must be told how many MAX72XX controllers are connected.

In three nested loops, each digit of the maximum three-digit number was incremented and merged into a string. This had the advantage of displaying the leading zeros as well and not having to work with complicated formatting functions.

2.2 Drawing faces on the matrix

For this task, full control over the matrix was necessary. Therefore, the library "MD_Parola" was no longer an option, as it is only suitable for textual displays. Instead, "LEDControl" was used, which gives control over each individual pixel of the matrix. That allows customs drawings.

For this, two matrices of ones and zeros were defined. A one means that the LED in the LED matrix should be lit at this position. A zero means that the LED is switched off at that position.

To select the matrices, two nested loops were defined that counted up an index. The index determined which number matrix should be selected. With the library "LEDControl" it is necessary to control every single row of the LED matrix. For this purpose, the inner loop was used to iterate over the number matrix and pass each row to the "LEDControl" instance.

Two facial expressions were chosen for the task: a laughing smiley and a frightened smiley.

A button could have been used instead of the first loop. When the button was pressed, a random value between 0 and 1 could have been generated. This would have determined which facial expression would have been selected. The functionality could have been extended considerably by pressing the button too often, which would have led to further facial expressions. This could have been used to represent emotions.

3 *Wire Layout for temperature measurement*

In this task, the temperature should be read from the DS18B20 digital thermometer and displayed on one or more matrices.

3.1 Implementation

The sensor sends the temperature data to the Arduino via the 1-Wire protocol. For this purpose, it is necessary to integrate the "OneWire" library. The instance of “OneWire” is given the PIN that is to be used for communication. To read the data from the digital sensor, the "DallasTemperature" library was used. This first asks the thermometer for the temperature, blocking it. As soon as the data is available, the temperature can be read out.

The 1-Wire protocol enables power supply and data communication via a single cable. It is used, for example, in Apple's MagSafe. With the thermometer, there are two possibilities. An external power supply via the last PIN of the sensor or a parasitic power supply via 1-Wire. To prevent the sensor from breaking when using the latter power supply, a resistor of 4.7 kOhm is connected in between.

A concrete example for the application of such a sensor would be the use of devices that are used in areas that are subject to strong temperature fluctuations. For example, outside in the garden. Such a sensor could protect the Arduino from overheating or undercooling. Or it could be integrated into a Smarthome and, if the temperature in a room is too high, it could communicate with a home central, which in turn would lower the shutters and switch on the air conditioning.

3.2 Temperature reading

The temperature is read out in a loop every few hundred milliseconds according to a non-blocking timer. The query could also take place much less frequently, as temperatures rarely fluctuate strongly in most cases and change slowly.

To display the temperature data, two LED matrices were used, controlled by "MD_Parola". The difficulty at first was to find out the correct orientation of the matrices, as this is felt to be different every time. The function "displayText" makes it possible to define exactly how the text is to be displayed. It is also possible to display moving text. The text is read from a buffer that can always be written into.

As soon as the data from the sensor is available, it is copied into the buffer. By resetting the screen, the new text, more precisely the temperature, is displayed. The temperature is thus displayed in a scrolling manner. This makes it possible to display a lot of content with little space.

For the task, I took my cue from pharmacy signs, which often also use LED matrices and alternately display the time, the temperature or the pharmacy symbol.

I wasn’t sure if I should’ve added my code and proteus files. I added all my files and codes in my GitHub Repository:

https://github.com/jpkmiller/UbiCom/tree/master/Lab1

Konstanz, 29.04.2021

Josef Müller