Hochschule Konstanz
Department of Computer Science

Lukas Seehuber
Patrick Reichle
Ubiquitous Computing Lab
Degree: Applied Informatics (AIN)
Lecture: Ubiquitous Computing

Report for the Laboratory 1

Arduino

Task One: From blinking to communicate

As a setup for the following tasks we started a Proteus project with the Arduino 328 and added a LED and a button. Both need to be connected to a ground (Figure 1: Button and LED connected to Arduino bottom left) in order to let the energy flow properly and make the circuit complete. Note that they don't need to be connected to the same ground! The LED is connected with its anode to the Arduino and it's cathode to ground, it additionally needs a resistor in the circuit, to protect the LED of high current and thus protect it from burn-out.[1]

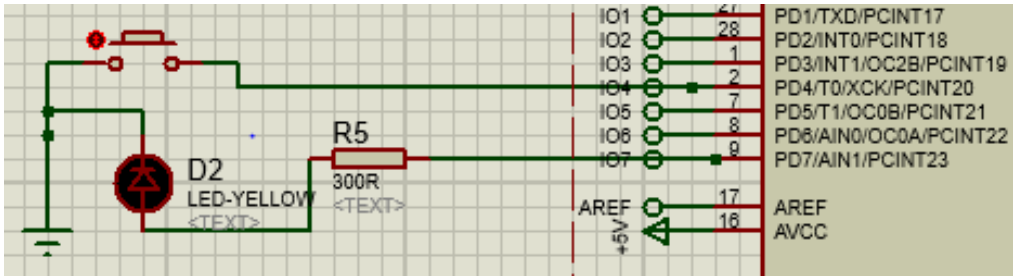

Figure 1: Button and LED connected to Arduino.

We read the state of the button (either *HIGH* or *LOW*) from pin 4 (PD4) and set the state of the LED to either *LOW* (0V) or *HIGH* (5V) correspondingly to the button[2]. Both actions can be utilized by ordinary digital pins, therefore we use the *digitalRead()* and *digitalWrite()* functions.

Press the button that you added bevor in the schematics. What happened? What can you observe?

As we press the button, the LED turns on. As soon as we let go of the button, the LED turns off again. That way we constructed a switch by simply looping through an if-statement, which sets the LED as described above.

Implement a Blinking LED

For this task we used the same setup as before in Figure 1: Button and LED connected to Arduino but the button is unused and could be omitted.

How does your LED Blink code work?

We simply use the function *digitalWrite(pin, led state)* to set the state of the LED *HIGH* or *LOW* (for on or off), then we delay for 1 second using *delay(1000)*. After that we set the LED to *LOW* (0V) and wait again using the *delay()* method.[3] Apart from that we tried another implementation (like Arduino IDE Example:

1 https://www.build-electronic-circuits.com/resistor-before-or-after-led/
2 https://www.arduino.cc/reference/en/language/functions/digital-io/digitalwrite/
3 https://www.arduino.cc/en/Tutorial/Blink

"BlinkWithoutDelay"[4]) that measures the time using the *millis()* and sets the LED state after a set interval. As in this case the program gives us the same result, we don't describe it any further, but the advantages are that we can continue looping and do other things, instead of putting the program to "sleep".

What is the difference between making a LED blink or fade? What are the technical limitations of each concept?

To make an LED fade, we need to use a pin that supports pulse width modulation (PWM) as we want to use the function *analogWrite()*, which "[turns] a digital pin on and off very quickly with different ratio […], to create a fading effect […]"[5], using a square wave. One such pin on the Arduino is pin 9, hence we alter the setup from Figure 1: Button and LED connected to Arduino by connecting the LED with pin 9 (note that we still need a resistor to protect the LED from high current).

In the code we can now use *analogWrite()* as described above and put different levels of brightness, by changing the width of a voltage simulating square wave from 0, where the ratio is none, and thus the LED is off, to 255 where the current equals to *HIGH* (5V) because the duty cycle is 100% (being always on).[6] In this implementation we again use the *delay()* method, to better visualize the different steps of fading and fade by steps of 5 (51 steps total).

Implement a Buzzer

In contrast to the implementation of blinking (where we set the voltage either *HIGH* or *LOW*) or fading (where we alter voltage by using PWM), we now use a method, namely *tone(),* that uses a square wave with a set duty cycle of 50% (width of 5V equals width of 0V), therefore we can use any digital pin (no PWM needed). As shown in Fehler: Referenz nicht gefunden, we use pin number 8.

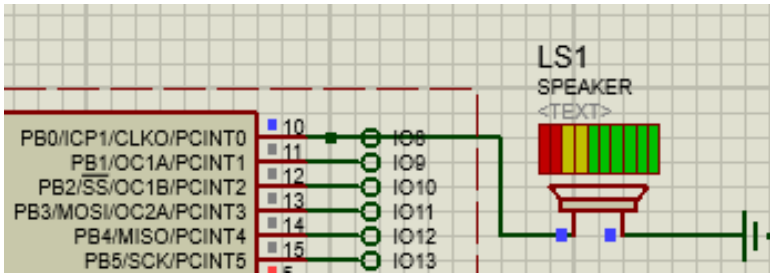

Figure 2: Speaker with visualization connected to Arduino. Currently playing.

To create different tones we alter the frequency of said square wave and thus the connected speaker creates a pitch by vibrating the air according to the frequency set (in the simulation it visualizes this process by showing a volume bar). We program that tone by using *tone(pin, frequency, duration)*. To play a proper melody we use the pre-defined *pitches.h*-file, which comes with the Arduino IDE Example "toneMelody"[7] and maps different pitches to their melodic tones. For a whole melody to play, we need to also set the proper tone length, by using the *duration*-parameter, and set pauses in between, again using the *delay()* method.

Merge previous ideas

The task is now to merge the above components. As shown Fehler: Referenz nicht gefunden we connect 3 LEDs to PWM pins (3, 9 and 10, we chose not to go with 5 and 6 as they have a different frequency than pin 3)[8] to be able to fade them. Then we have a button connected as in Figure 1: Button and LED connected to Arduino and a speaker connected like in Fehler: Referenz nicht gefunden.

4 BlinkWithoutDelay, https://www.arduino.cc/en/Tutorial/BlinkWithoutDelay Accessed 13 June 2020.
5 Fade, https://www.arduino.cc/en/Tutorial/Fade Accessed 13 June 2020.
6 PWM, https://www.arduino.cc/en/Tutorial/PWM Accessed 13 June 2020.
7 toneMelody, https://www.arduino.cc/en/Tutorial/toneMelody Accessed 13 June 2020.

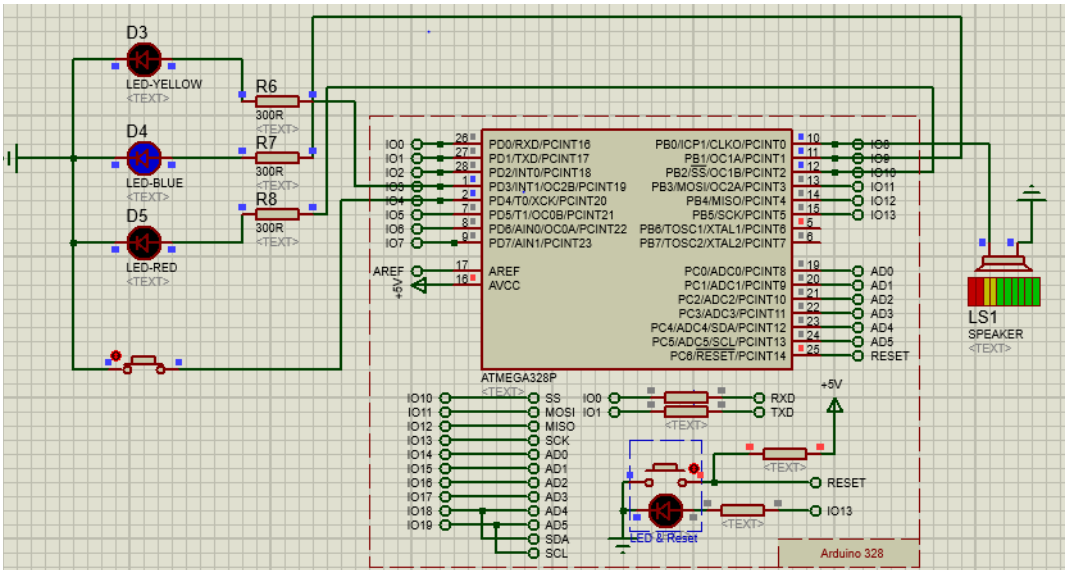

Figure 3: Arduino with multiple components interacting with each other.

We now let the LEDs fade on and turn them off after each other, at first the yellow one, then the blue one and finally the red LED fades on and turns off again. Done with the *analogWrite()* which chooses the right LED based on a value which indexes into an array of the above LEDs.

So far the LEDs fade after each other but to make it fun, the user can press the connected button and immediately the speaker will play a tone (as shown in the picture Fehler: Referenz nicht gefunden). However, the magic behind our implementation is that every colour is also represented by its special tone. This way yellow maps to the tone C, blue to F and red to G, implemented by using the same value as for the LED, but this time it is used to index an array of frequencies to match the tones (262 Hz, 349 Hz and 392 Hz) and is passed into the *tone()*-method to generate that using the speaker.

Which use case would you plan for a system with these components?

This implementation could be used as a reaction game, where you are only allowed to press the button while the right LED is lit; otherwise the played tone indicates that you pressed the button at the wrong moment.

Another probably more educational implementation could be achieved by extending the pure tone to a word, thus educating young children. For example by telling the directions of the LED like “left, middle, right” or even more intuitive the colour itself “yellow, blue, red”, when the child is using the button, extracting that information.

Matrix: How to work with LED matrix

To implement the LED matrix in Proteus were using the HK16K33 controller. The controller is connected to the Arduino via the SDA and SCL pins and the LED matrix is connected to the controller. The benefit of using a controller is that we only need to send the desired state of the matrix to the controller and then the controller takes care of switching on or off the LEDs. The Adafruit LEDMatrix library provides functions to manipulate the matrix and communicate with the controller. Therefore an instance of the Adafruit_8x16matrix is created. After the manipulation of the matrix is done the matrix is send to the controller with the writeDisplay() method.

8 analogWrite(), https://www.arduino.cc/reference/en/language/functions/analog-io/analogwrite/ Accessed 13 June 2020.

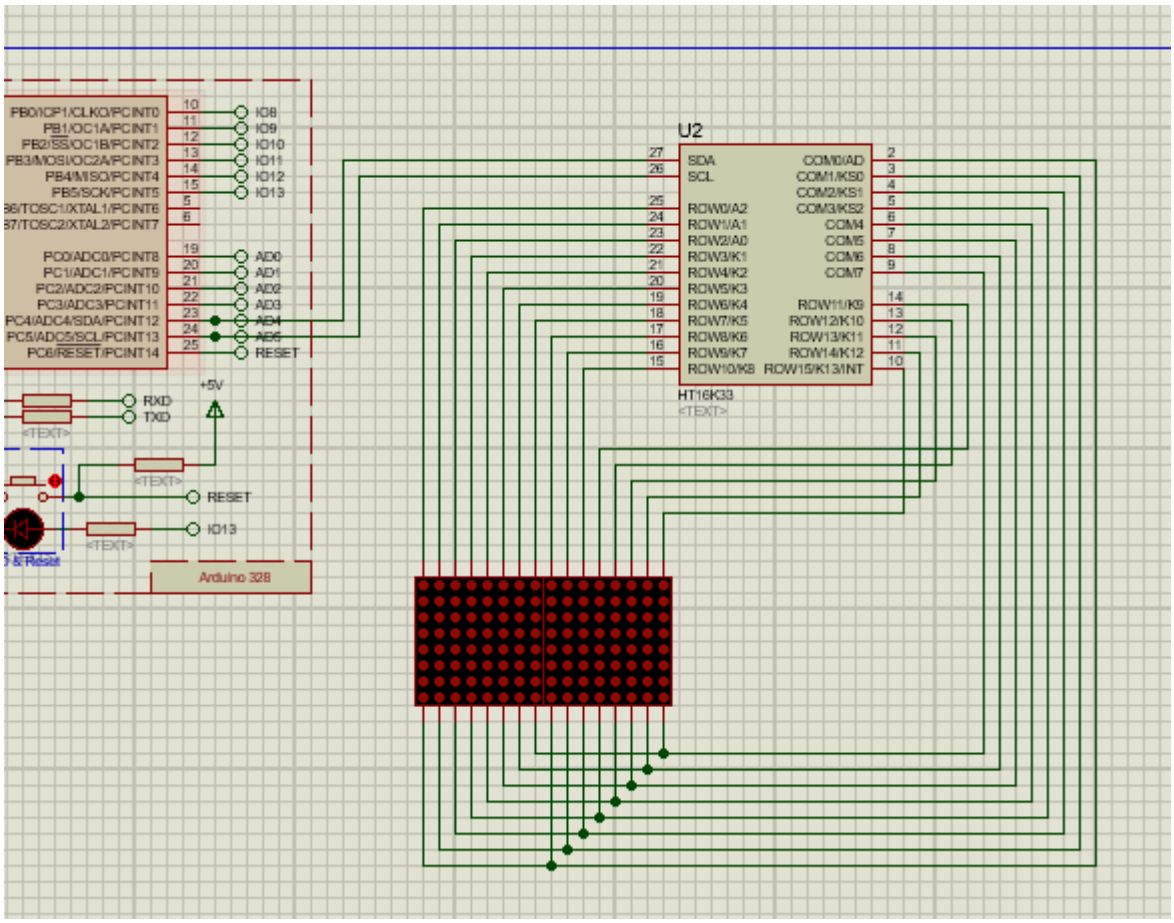

Figure 4: LED matrix with HT16K33 controller in Proteus

LED Matrix counter

We implemented a simple counter that displays a number by lighting up the corresponding number of LEDs. Therefore are counter is limited to numbers between 0 and 64. To light up one pixel the drawPixel(x,y,color) is used. X and y represent the coordinates of the pixel.

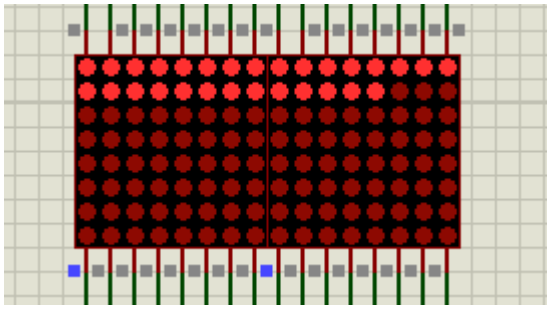

Figure 5: Simple counter

For each number N the amount of full rows to be switched on can be calculated by N / 16 (rounded down), the number of extra LEDs is N % 16. For example 20 is displayed by one full row of LEDs and 4 extra LEDs.

LED Matrix for output of numbers

To implement a counter that displays numbers as characters we used the print(str) function. This function prints any given ASCII characters on the matrix. Due to the space limitations of the matrix it can only display two digits at the same time. To display longer numbers you could use the setCursor(x,y) function. This function sets the coordinates where the first character is printed. By moving the cursor with a loop the text can be made to appear 'sliding' through the display.

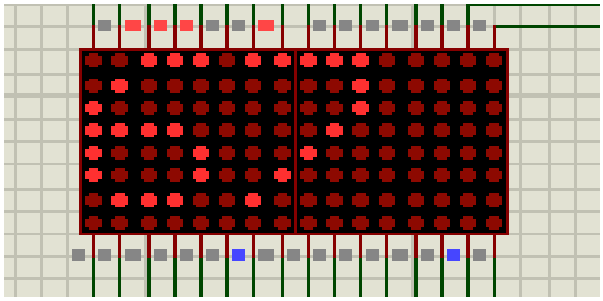

Figure 6: Counter that displays characters

Be creative and draw

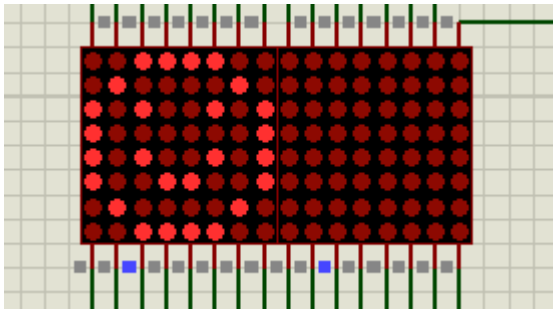

Figure 7: Display of a happy face

To display an icon with the drawBitmap() function we need the icon in a bitmap format. A bitmap stores the value of each pixel. The bitmap is stored as an array of bytes, where each 1 represents a switched on LED. The bitmap of the happy face looks like this:

00111100
01000010
10100101
10000001
10100101
01000010
00111100

If you look closely you can even recognize the happy smiley in there. The bitmap can be displayed with the drawBitmap function. The other parameters of the function are the starting position, where the top left pixel is drawn, the height and width of the bitmap and the color.

Expand the Matrix

We added a button and each time the button is pressed the happy smiley changes to a sad smiley or the other way around. Therefore we added the bitmap of a sad smiley.

LED Matrix as Terminal Output (Optional)

We use the Virtual Terminal in Proteus, it is able to send and receive data via serial connection. Make sure to connect the RX pin of the terminal to the TX pin of the Arduino and vice versa.

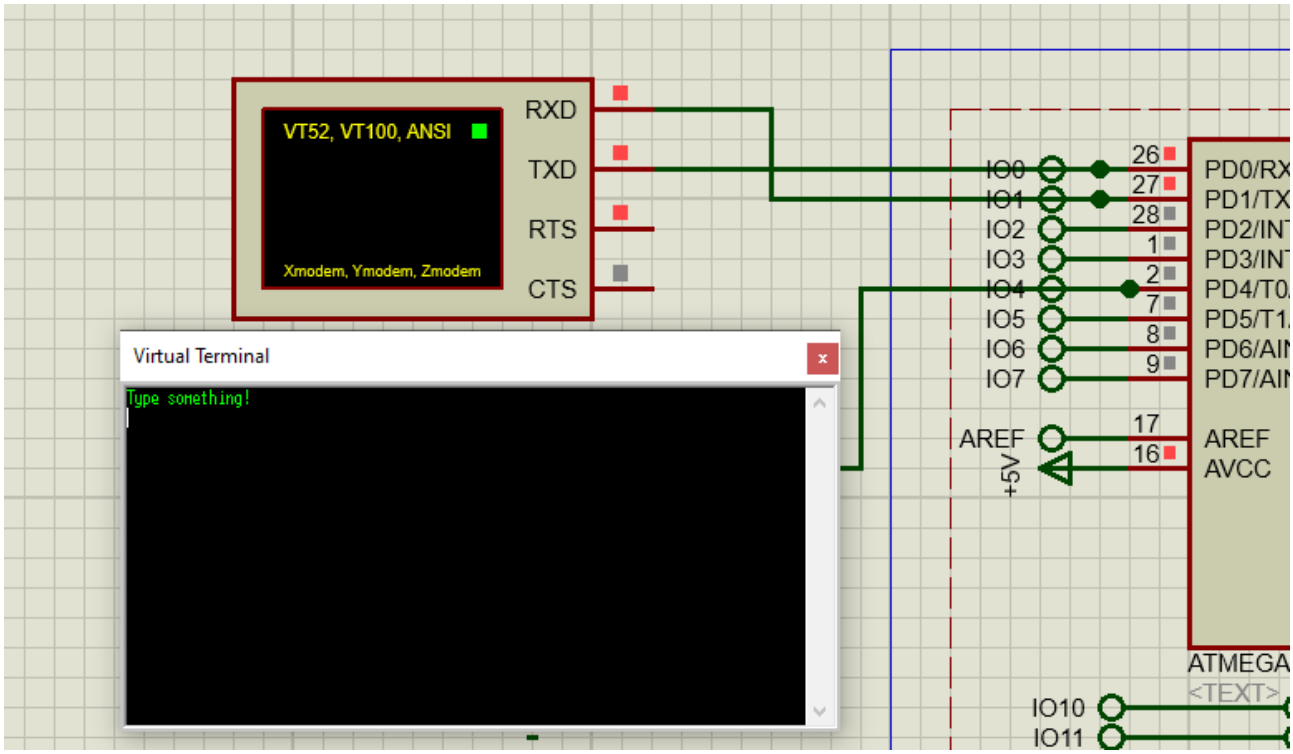

Figure 8: Virtual Terminal in Proteus

To establish a serial connection we use Serial.begin(9600) where 9600 is the BAUD-rate of the connection. The BAUD-rate is the number of transmitted symbols per second.

If there is new data on the serial connection to read, serial.available() returns true. The data can then be read by serial.read(). To send data via the serial connection we use serial.print(data). The sent data is then displayed on the Virtual Terminal.

Wire layout for temperature measurement (Arduino)

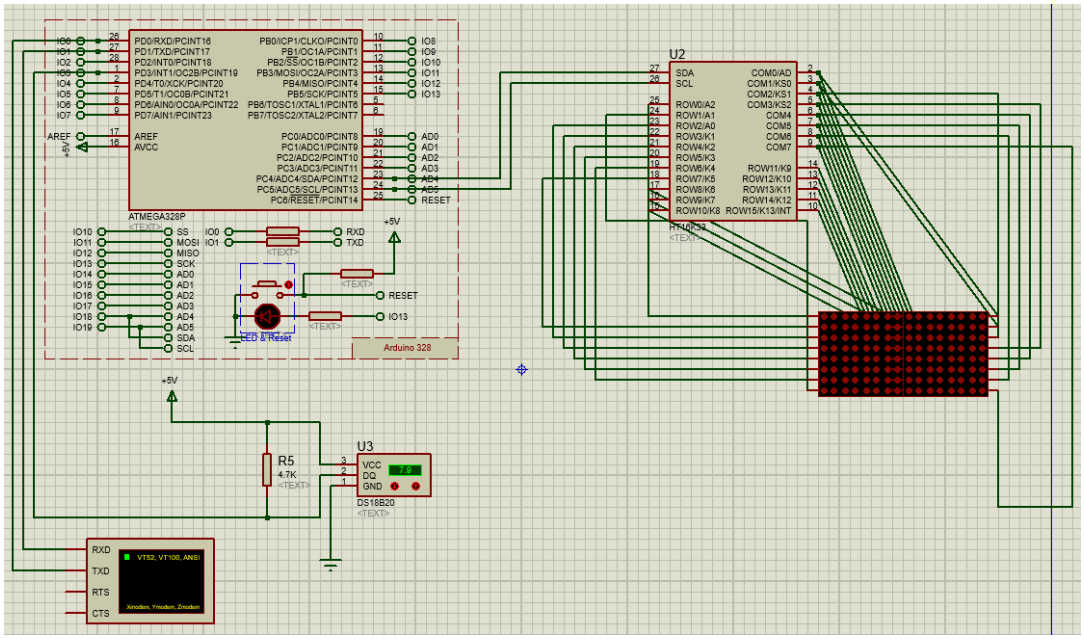

Figure 9: Complete Setup with temperature sensor (external supply mode), terminal and LED-matrix

Implementation

The sensor communicates using one control signal (hence it only requires one data line to the Arduino and one line to ground), implemented via the One-Wire-Protocol. We implement this protocol into the IDE as stated in the Lab description. “The control line requires a weak pullup resistor since all devices are linked to the bus via a 3- state or open-drain port (the DQ pin in the case of the DS18B20)”[9], hence we put the resistor as shown in Fehler: Referenz nicht gefunden. (You could add infinitely more sensors on that bus by theory)

We use the provided default 12-bit Celsius temperature measurements. It works between temperatures of -55°C (hex coded as 0xFC90) to +125°C (0x07D0) and is accurate to ±0.5°C between -10°C (0xFF5E) to +85°C (0x0550).

For check whether the sensor sends the right data to the Arduino, we connected a terminal to the Arduino as shown above (RX to TX and vice versa for Serial connection) and printed the output of the sensor on it. We used the “Code Example” on the OneWire Arduino Page[10]. Against the instructions we needed to implement the hex-conversion (described just below the Code Example on the same website) by ourselves, because the suggested Dallas Library for the temperature sensor didn’t work in the Proteus Simulation software.

Simply any measurements would be interesting, but as you can add more sensors simultaneously you can get quiet creative. For example in a Smart Home you could automatically have the windows opened to get some fresh air into your home, but to save energy the Arduino can check for how long you should keep your windows open determined by outside and inside temperature difference, and also suggest you to open the

9 Datasheet DS18B20, https://cdn-shop.adafruit.com/datasheets/DS18B20.pdf Accessed 17.06.2020
10 Dallas Semiconductor’s 1-Wire-Protocol, https://playground.arduino.cc/Learning/OneWire/ Accessed 17.06.2020

windows in hot summer periods if the inside temperature is higher than outside. With only one sensor you could still set an alarm if it gets to cold inside (especially during winter), or close the windows automatically.

Another great integration for more efficiency would be a regulated heater for an aquarium, where the water heater only turns on when the room temperature can't hold the water temperature in a specific range. Normal heaters just start heating as soon as the temperature drops to a pre-defined point and heat a bit beyond to pause again, which causes energy waste. If you can give up a few degrees in order to don't use the heater at all, because room temperature is enough, a dynamic system would be way more efficient

Temperature reading

As stated above we couldn't use the Dallas library and used the Code Example with added conversion instead.