Team Members: Louis Solovy & Hongyu Zeng

Professor Beichel

ECE:3360 Embedded Systems

Post-Predefined Project Report

**1. Introduction**

The goal of this project was to make a weather data monitor in C language that consisted of using a DHT11 sensor to track the temperature and humidity, a photoresistor as a light sensor to capture the brightness, a pushbutton for the user to press when they want to change the data being printed out, and a timer to allow for data to be outputted at user set intervals. For this project we also needed to use a user interface of our choosing, and we chose to use the serial monitor in the Arduino application. Our weather monitor had to have the ability to periodically take measurements specified by the user and allow for the user to configure the system and view the measurements within our interface, which is the serial monitor. The way that our weather monitor works is that the default interval will be set to 5 seconds and it will print out all measurements until the user presses the button which then will prompt the user to enter which measurement they want to take and for how long they want the intervals to be taken at.

**2. Description of provided functionality**

Our project has 4 main functionalities: DHT11, photoresistor, pushbutton, and a timer. The functionality of the DHT11 sensor is to get readings of the temperature and humidity from the location of setup. The photoresistor has the functionality of getting a reading for the brightness from the location setup using the ADC that we have learned from previous labs. The functionality of the pushbutton was to allow for the user to interact with our program that would allow for the user to set the time interval at which measurement specified they want to send to the serial monitor. The timer’s functionality is to implement the interval at which measurements will be intervalled in the serial monitor.

**3. Documentation of Design**

Overview

The overview of our design consisted of having a user interface that would display the measurements of DHT11 and the brightness of the photoresistor at given intervals described by the user. Since the user interface that we were using was the serial monitor, we needed to implement the serial communication device (USART) to allow for receiving and transmission of data. This would be our main functionality that would allow for messages and data to be printed to the serial monitor. Also since our project would be using interrupts to operate we also needed to integrate an interrupt service routine (ISR) that for our program would be using INT0 to generate interrupts. This service would be used for helping us to transmit data to the serial monitor. We also would be using a timer that needed to be implemented to allow for the user to print out data at their specified interval. The timer was simply created by allowing the user to pick the seconds, minutes, and hour they wanted the interval to be set at and then in our program we would take their input and calculate their interval by a function that would compute the interval in seconds and then print the data at the interval. This was done by using a delay base of 1000 ms that would just be looped over multiple times depending on their input. Another piece of our design was having a push button that the user could interact with that would allow for them to change their interval and which data was printed. The button was implemented using a software debounce to tell whether the button was pushed and a function to allow them to change their input if a press was detected. Overall, the overview of our design consisted of using the serial monitor to print the data and interval selected by the user. The data that could be selected could either be the temperature or humidity from the DHT11 or brightness by the photoresistor using the input signal of the ADC, or all three.

Hardware

Figure 1: Circuit Diagram

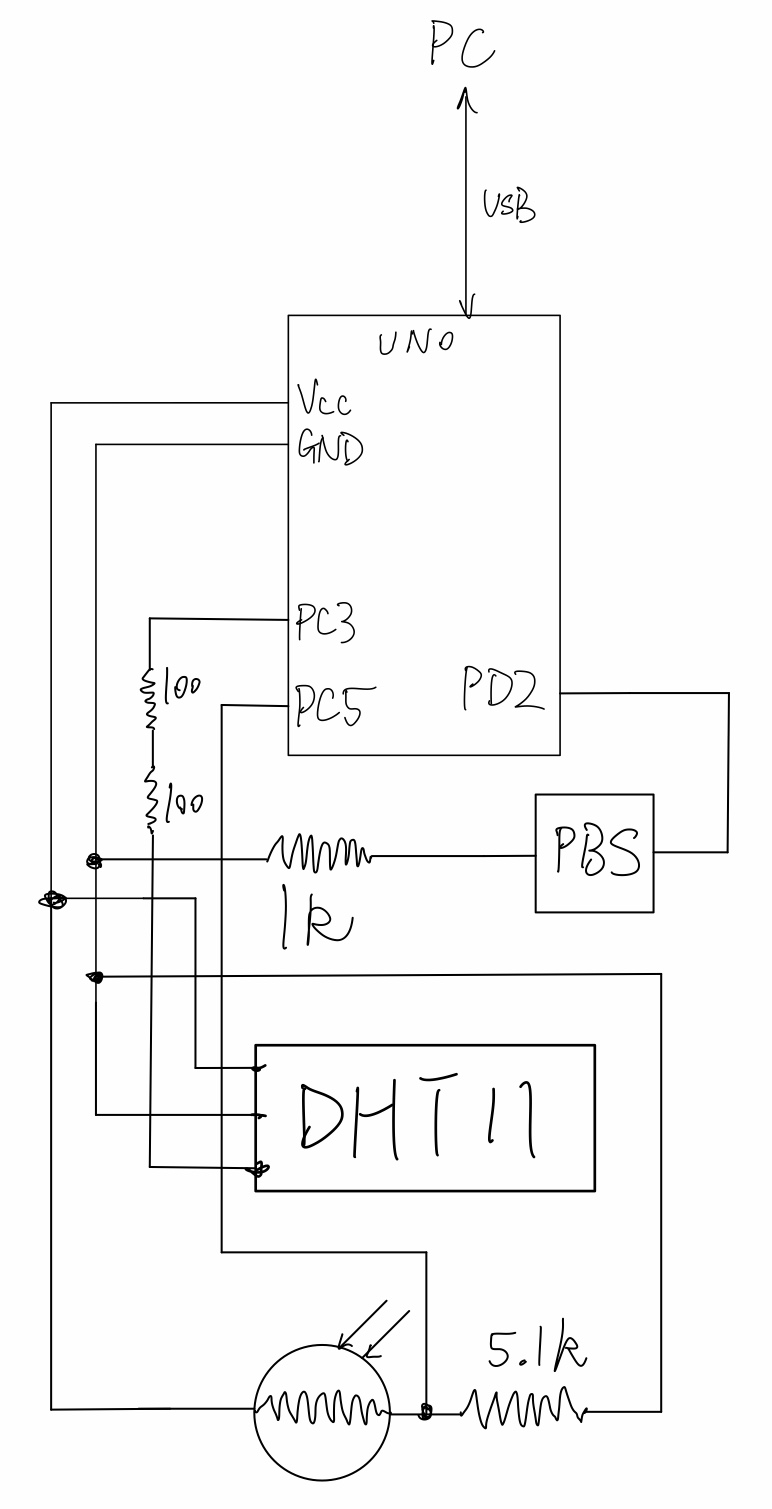
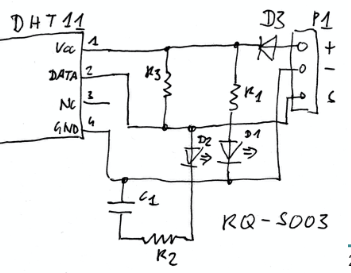
For our project, the hardware wasn’t too complicated considering that the only physically hardware pieces that were connected to the microcontroller were the DHT11 sensor, a photoresistor, and a pushbutton. The setup of the DHT11 consisted of having three wires connected to the microcontroller, one connected to GND, another connected to Vcc, and another one connected to a pin on the microcontroller to allow for the transfer of data. The wire connected to PC3 was the data input that consisted of being connected to two 100 ohm resistors. We got this calculation of using two 100 ohms resistors from the professor, he told us that this was just for protecting the I/O from electronic bus conflicts if the set up was incorrect. For the DHT11 to work, the PC3 output buffer should be turned off so no electrical bus conflict. To indicate the start of a reading, first use PC3 as output to send a 18ms low. Then use PC3 as Input to read the data. For the photoresistor, it consisted of being connected to Vcc, GND and to a 5.1k ohm resistor that would be connected to PC5. The reason for picking a 5.1k ohm resistor is because The photoresistor has a resistance between 500 to 2M. When it’s dark, 5.1k will not share a large voltage with the photoresistor and when it’s bright the 5.1k will take about 5100/5600 = 91% of the voltage. We believed that these numbers were desirable for our project. For reference, Figure 1 will show an image of our drawn circuit schematic and Figure 2 will show a drawn schematic for the setup of the DHT11 sensor. *Input Pins*: PD2, PC3, PC5, *Output Pins*: PC3 

Figure 2: DHT11 setup

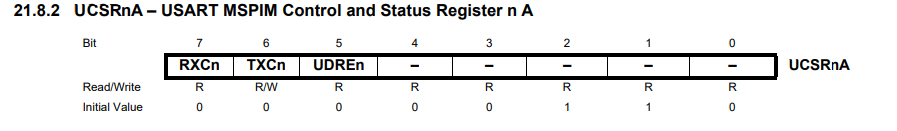


Software

For the software side of our project, there were many functionalities that needed to be implemented, some being easier than others. To start off we needed to successfully implement the USART to be able to transmit and receive data from the serial monitor. Lucky, we have done this in previous labs. This was done by creating a function called usart\_init(), that would set the baud rate along with enabling the receiver and transmitter using TXEN0 and RXEN0. The way we set the baud rate was using the USART baud rate registers UBRR0H (high) and UBRR0L (low). Since the baud rate was 12 bits we put the high to 4 bits and the low to 8 bits. Then we needed to set the character size of the format to 8 bits by setting UCSR0C bit 1 and 2 to 1; to enable receiver and transmitter we set the bit 3 and 4 on UCSR0B to 1.

Next was to implement the usart\_transmit(char arr[]) and usart\_receives(). For the transmit function we put each char in the char array in the UDR0 using polling style. We only transmit the char when the USART data register is empty. This is done by checking if the bit 5 on the UCSR0A is set to 1. For receiving char, we are taking one char at a time. This is done when the bit 7 on UCSR0A is set to 1 which means that receiving is complete (refer to figure 3).

Figure 3: UCSR0A



Next was to allow for the photoresistor to use the ADC for a reading that would be interpreted into a brightness value. We first needed to configure the ADC. For ADC, we first needed to configure it. We used the 5V Vcc for reference voltage by setting bit 6 to 1 on ADMUX (refer to figure 4). And since we are using PC5, we don’t need to change the bit 3 to bit 0 since its initial value is 0 which corresponds to ADC0. And to enable ADC, we set bit 7 on ADCSRA to 1 (refer to figure 5).The ADC value we have is a 10 bit resolution according to the datasheet we needed to have the input clock frequency between 50-200kHz. We needed to have a prescaler of 128 so our clock frequency was 125kHz, which is in the range. This was done by setting the bit 2-0 on ADCSRA to 1 (refer to figure 6).

.

Figure 4: ADMUX

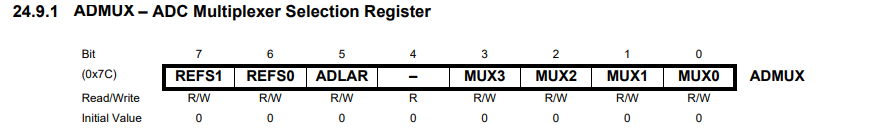


Figure 5: ADCSRA

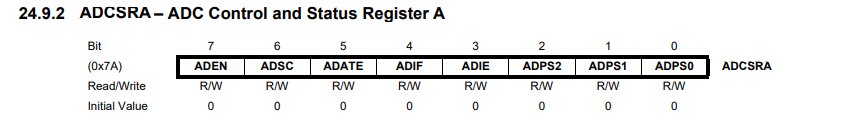
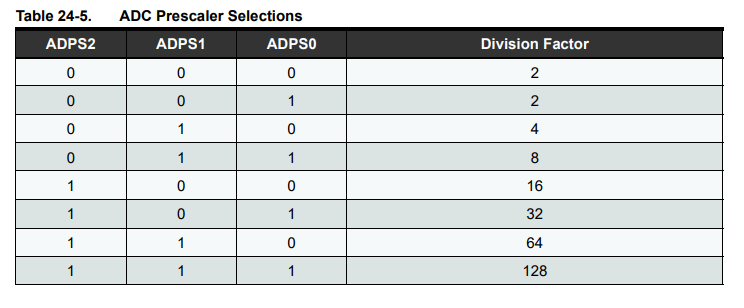


Figure 6: ADC Prescaler



To read ADC value, we set bit 6 on ADCSRA to 1 to start the conversion from analog (voltage input to PC5) to digital values (0-1023). When the conversion is finished (the bit 6 will return to 0 when the conversion completes), we read the value from ADC. Then we made a function called LightnessOutside() that would return the brightness in weather terms. For example if the value read from the ADC was between 301 and 699 it would display “Partially cloudy”. This function handled all values between 0-1023 and would output the brightness in weather terms what it would be like outside. The way that we got this data was by putting our photoresistor outside during different times in the day.

Next we decide to try and get our DHT11 to read the temperature and humidity. Although there were libraries that would help to get this to work, we took the route of making our own DHT reading procedure. This was done first by setting PC3 as an input and then delaying for 18 ms to initiate the DHT11 and then setting PC3 to an input. Then using four while loops we had to wait while it was rising and then had risen twice before we could start to get readings from the DHT11. After the rising state and 160 us of starting bits, we used a nested loop to read bits to an unsigned 8 bit digit called DHTbyte. Then using two integers for counting: one for counting how many bits we’ve processed for the current byte and one for how many bytes have been stored. The logic for determining bit 1 and bit 0 is the following: we wait till high (no longer in gaps between bits), wait for 35us so that if it’s bit 0 then then PD2 would be low, otherwise it’s high. And if it’s bit 1, then we have to wait till it goes to the gaps. Then we used the counter for counting the bits processed and if the counter equalled 8 we transferred the digit into the DHTresult. And finally when the counter for received was equal to 5 that meant that a reading for the DHT was done. But before using the USART to transmit the data was needed to check and make sure it was correct by checking the checksum digit. This was done by computing the addition of the first four digits in the DHTresult and making sure that it equaled the fifth digit in the result, and after making sure it equaled the checksum digit we could then transmit it to the serial monitor. We also decided to add another function that would tell how warm or cold it was by grabbing the temperature and for example if the temperature was less than 0 degrees Celsius it would display “BRRRRR its very cold out”, this was done in the function displayTemp().

Next was to implement a timer for the user to select an interval they want the data to be intervalled at. This was done by creating a function that would make three variables, one for seconds, minutes, and hours. Then based on the user input, it would then create a delay that would be their interval. This was done by having three separate while loops with a base of 1000ms delay, which is one seconds. For the seconds loop it would loop the amount of seconds inputted by the user and just use that base delay since 1000ms is equal to one second. For the minutes loop, it would loop the same way as the seconds while loop by also looping the delay 60 times for the base delay to be a minute. (1000ms \* 60 = 1 minute). And for the hours loop it would be the same but loop the base delay 3600 times ( 1000ms \*3600 = 1 hour). This was how we created our timer for our project, we didn’t need to use a RTC since this way worked perfectly fine. This process can be seen in our function delayOverall().

Finally was to implement our button to allow for the user to change the set interval whenever they want by just pressing the button. The first thing we had to do to implement the PBS was to create a software debounce function that would indicate when the button was pressed. This was done by using the software debounce in C that was given to us in class, where we would loop 9 times and keep track of the count of times PIND2 was equal to 0. And then if the count was equal or greater to 5 that meant the button was pressed. After creating the debounce we also decided to initiate an interrupt service routine (ISR) for the button. We used the External interrupt INT0 on PD2 with a falling edge trigger (the internal pull-up is enabled in PD2 so if the button is pressed it will go low). This was done by initializing registers EICRA and EIMSK. We set EICRA to 2 so that ISC01 was 1 and ISC00 was 0 to create the falling edge of INT0 that invokes the interrupt service (refer to figure 7 and 8). Then we needed to set EIMSK to 1 so that INT0 interrupt was enabled (refer to figure 9).

Figure 7: EICRA

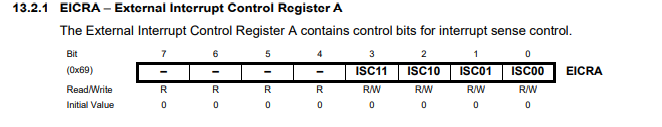


Figure 8: Interrupt 0 Sense Control

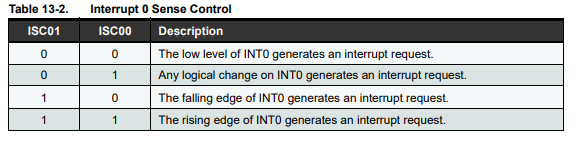
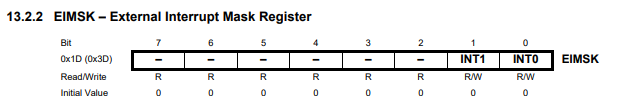


Figure 9: EIMSK



We then, for the interrupt service, needed to implement a function that everytime the button was pressed a message would be printed out to the serial monitor that would tell the user to enter the new intervale along with the data they wanted to see. This was done in the function called ISR that would take INT0\_vect so that an interrupt would be created. The first thing we did in this function was calling the debounce and then sending a message to the serial monitor that would prompt the user to enter in the new interval with the data they wanted enabled. It would then check to make sure that the inputs were all valid and then put their input in a buffer that would echo it to the main loop where the data would be then be handled.

Overall, our program will run in our main loop called main that is a void and in this function it will initialize our USART along with our ADC and set up the interrupt by calling sei(); to enable the interrupts. Then it will check which of the following data measurements that user selected and print them out according to the invertval they also selected. This was done by having variables in the ISR function to tell whether the selected data was enabled or not. The way our program is runned by a user is they will first start the program and by default a 5 second interval will be given along with enabling all data measurements to be printed out the serial monitor. Then if the user would like to change the default interval and data selected they will then push the button and enter the interval they want by hours, minutes, seconds, followed by the data they wanted to see. This was done using three integers, for example if they wanted to just see humidity they would type 100, and if they wanted humidity and temperature they would type 110. The first digit represented the humidity, the second was the temperature and the last digit was the brightness. So a valid input would be something like, 0,1,1,001 (SPACE) which would print out the brightness in intervals of 61 seconds (a minute and a second). Also some notes on our program is at the beginning it will define the baud rate to 9600 and the 16Mhz clock that we would be using for the microcontroller. We also needed to add the optimization -O1 along with in the AVR/GNU Linker, in General, select “use vprintf Library”; in Miscellaneous, “-lprintf\_flt -lm”.This is how a user will be able to use our weather monitoring application.

Libraries used:

#include <stdio.h>

#include <avr/io.h>

#include <util/delay.h>

#include <avr/interrupt.h>

**4. Description of Test Procedure**

For our testing procedure, we wanted to first make sure that all the requirements were working then we could check to see if our additional features were working properly. Although we were testing many functionalities throughout making our program we still wanted to give an overall test at the end of the project. Some of the components that we were going to need to test that were requirements for this project were the DHT11 readings, photoresistor reading using ADC, our functionally of the user interface and our implementation of our timer for the intervals. After finishing the project we decided to go about testing out functionality using a checklist to make sure that all of our functions were working properly. The first test we conducted was making sure that when the user runs the program and opens up the serial monitor, a default setting of having an interval of seconds will be set along with enabling all these data measurements. After making sure this was correct (refer to figure 10) the next we wanted to test was making sure that when the user pressed the button they would prompted with the following statement ("Please enter command for configs: (hour,minute,second,3 bits command for readings(example: 111)+SPACEBAR)"), that would ask them to decide an interval, and the measurements they wanted to collect. After making sure that this worked (refer to figure 11) we then needed to make sure that the input the user was giving was actually getting read. This was done by just checking if our program would read the humidity at an interval of 2 second. With reference to figure 12, one can see that this was working perfectly with input of (0,0,2,100). Since we tested for humidity from the DHT11 we also needed to test for the temperature from the DHT11 along with the brightness from the photoresistor using ADC. We decided to capture this in one test to also test the functionality that a user could pick more than one measurement to be selected. With reference to figure 13, one can see that with an input of (0,0,3,011) the temperature and brightness data will be enabled along with being printed out to the serial monitor every 3 seconds. To ensure that the sensors were working properly we also did some tests on them for instance to test the photoresistor we used our flashlight on our phone at different levels to make sure that it was working properly (refer to figure 14). For testing to make sure the temperature and humidity were working properly was a little more difficult but we found that either holding the DHT11 or breathing on the DHT would make the temperature increase along with the humidity (refer to figure 15).

Figure 10: Default Interval Setup

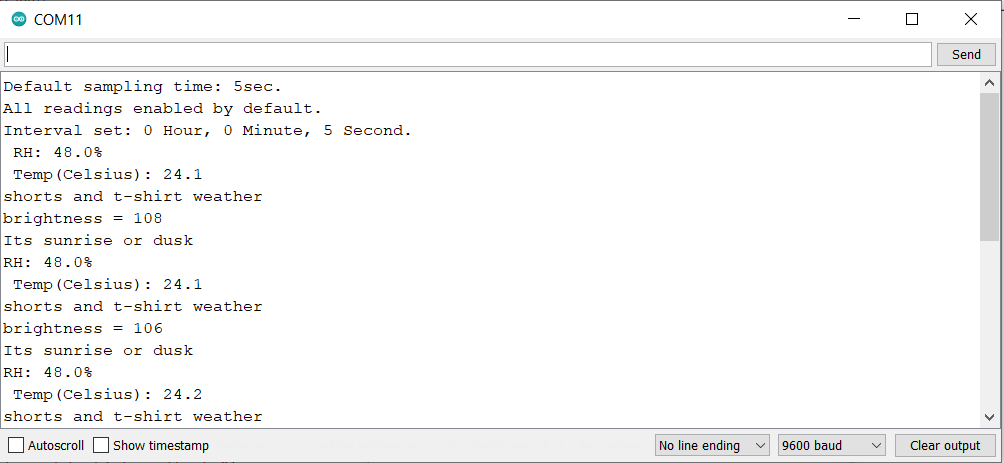


Figure 11: Button is pressed command



Figure 12: After entering 0,0,2,100 (Humidity)

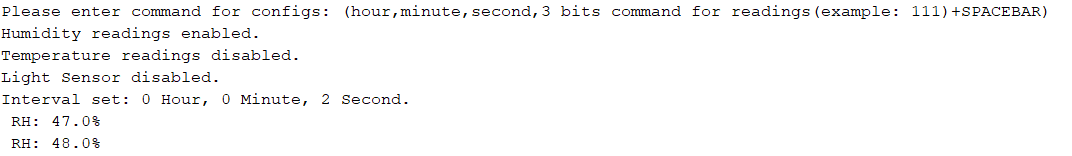


Figure 13: After entering 0,0,3,011 (Temperature and Brightness)

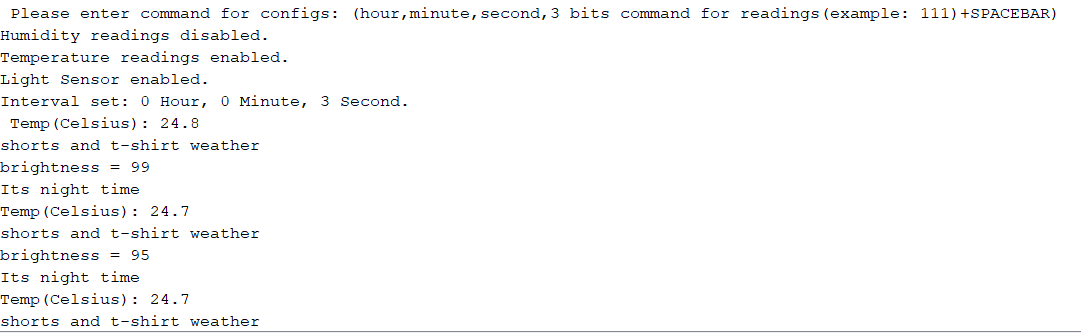


Figure 14: Testing Photoresistor

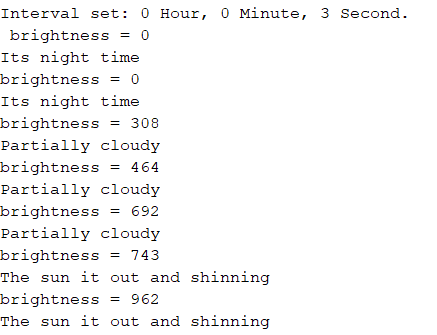
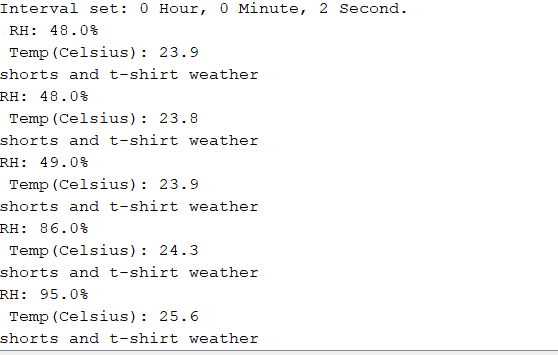


Figure 15: Testing DHT11



**5. Limitations and Future Designs**

Since our project had a time constraint on when it had to be done, there could have been many features that could have been added along with possible limitations that could have been adjusted. One being we tried to implement the interval constraints with time being limited but we thought that this was not the most important idea for our project so we did not perfect it. Another limitation that we saw was when the user wanted to print no data measurements, we didn’t fully handle this case since the user will most likely always be trying to get some sort of data. But for a future design, there could be an error message that will get printed out that handles this case. Also another limitation that I will talk about in future design is being able to handle a case of setting an interval to less than 1 second. We also had a limitation in our program that would assume that the values entered for the interval are positive, but this can have been fixed for future designs. Our design could have had many more implementations of functionality but since we were limited on time we will talk about what future designs could have been made to potentially make this product feasible. One idea that could have been made was making a case for the product so that if it is raining outside you will still be able to collect all the data measurements without damaging the systems. Another idea would be to have an interface that would be a web application so that the user doesn’t need to use a computer to run the program, for example creating a more user friendly interface for a phone app. Going off of a better user interface, we could have made our user interface be more user friendly by having it be more structured like adding modes. For example, we could have created three modes, one for humidity, temperature, and brightness, and then in each of the modes ask the user for the interval they want to interval them at, but instead our project would just take all of that as one input making it not as structured that it could have been. Another small future design that would have been interesting to add would be a display of the timer in the user interface. For example, in the user interface it will have a stopwatch that the user can see counting down for the interval at which they set the measurements to be displayed at. Adding more to the timer, could have been implementing a limit that would allow the user to select intervals less than 1 second if they desired. Lastly, we could have added a feature that would allow the user to select how many times the sensor is sampling in each sampling time and take the average so the output is more reliable. We could go on and on about things that could have been done to make this product better for future designs, but when it comes to our project with our time constraint, we can say that our project operates as intended for a weather monitor system that has the able to display the humidity, temperature, and brightness at intervals using the serial monitor.

**6. Conclusion**

After completing this project we can now say that we successfully made a weather data monitor in C language that consisted of using a DHT11 sensor to track the temperature and humidity, a photoresistor as a light sensor to capture the brightness, a pushbutton for the user to press when they want to change the data being printed out, and a timer to allow for data to be outputted at user set intervals. Throughout the process of making this program there were some difficulties that we ran into and had to solve. One of them being at first, for reading the photoresistor using ADC we kept getting 0. This was solved by making sure that we had set up the circuit correctly which at first our photoresistor was incorrectly set up. Another difficulty that we ran into was correctly setting up the DHT11. At first while we had to wait for the DHT11 to initialize before actually reading the input, we had our loops set up wrong. Since we had to wait for it to rise and then check to make sure we skip the start bits, we needed to implement four while loops. We kept on getting an error when making these while loops and then realized that we need parenthesis inside “while( (PINC&(1<<PINC3)) != 8 )” of the while. This was a small problem but it took us a little longer than other problems because the error was so tiny and hard to see. We did have some other small difficulties but the ones that I have talked about were some of our bigger problems we ran into. Overall, our weather monitor has the ability to periodically take measurements of the DHT11 and photoresistor specified by the user and allow for the user to configure the system and view the measurements within our interface.

**7. Appendix A: Source Code**

**;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;**

/\*

\* WeatherMonitor.c

\*

\* Created: 4/26/2021 8:59:35 PM

\* Author : HONGYU & LOUIS

\*/

// https://www.tutorialspoint.com/cprogramming/c\_passing\_arrays\_to\_functions.htm

// https://stackoverflow.com/questions/37487528/how-to-get-the-value-of-every-bit-of-an-unsigned-char

//

// define clock frequency and calculate the corresponding baudrate

#ifndef F\_CPU

#define F\_CPU 16000000UL

#endif

#define USART\_BAUDRATE 9600

#define UBRR\_VALUE (((F\_CPU / (USART\_BAUDRATE \* 16UL))) - 1)

#include <stdio.h>

#include <avr/io.h>

#include <util/delay.h>

#include <avr/interrupt.h>

void usart\_init();

void ADC\_init();

unsigned char usart\_receives();

void usart\_transmit(char arr[]);

void readDHT();

void displayDHT(char outstr[]);

void displayTemp();

void readBrightness();

void displayBrightness();

void LightnessOutside();

void delayOverall();

void debounce();

// global variables

int sec = 5;

int minute = 0;

int hour = 0;

uint8\_t DHTresult[5] = {0,0,0,0,0}; // DHT11 data array

char command[20];

int changed = 0; // if the button is pressed and command is given

int pressed = 0; // if the button is pressed

int brightness = 0;

int RHenable = 1; // Humidity data reading enables

int TEMPenable = 1; // Temp data reading enables

int BRenbale = 1; // light sensor data reading enables

int ReadingEnables = 111; //RH,Temp,Brightness

ISR (INT0\_vect) { // interrupt function to display msg to serial monitor along with receiving input

debounce();

if (pressed) {

int timebuffer = 0; // buffer for number extracted from command

// reset

char c;

char echo[100];

int i;

int valid = 1; // assume valid input at first, if invalid, clear to 0

usart\_transmit("Please enter command for configs: (hour,minute,second,3 bits command for readings(example: 111)+SPACEBAR)\n");

for (i=0;i<24;i++){

c = usart\_receives(); // Get character

command[i] = c;

if (c == ' ') {

break; // space bar indicate the end of a command

}

}

command[i] = '\0';

char \*pch; // points to the start of the next string fragment

int counter = 0;

pch = strtok(command, ",");

while (pch != NULL) { // split the M command to get sampling times and intervals

if (counter == 0) { // hour input

sscanf(pch, "%d", &timebuffer);

if (timebuffer >= 48) {

usart\_transmit("Invalid time settings.\n");

valid = 0;

break;

} else {

hour = timebuffer;

}

} else if (counter == 1) { // minute input

sscanf(pch, "%d", &timebuffer);

minute = timebuffer;

} else if (counter == 2) { // sec input

sscanf(pch, "%d", &timebuffer);

if (timebuffer <= 1 && hour == 0 && minute == 0) {

usart\_transmit("Invalid time settings.\n");

valid = 0;

break;

} else {

sec = timebuffer;

}

} else if (counter == 3) { // data enables input

sscanf(pch, "%d", &ReadingEnables);

} else {

usart\_transmit("Something went wrong.\n");

}

counter ++;

pch = strtok (NULL, ","); // memories the modifed string so NULL for no new input

}

if (valid) { // valid time settings

// modify enables

if (ReadingEnables == 0) {

usart\_transmit("Warning: No readings enabled.\n");

} else {

if ((ReadingEnables / 100) != 0) { // extract MSB for Humidity data reading enables

RHenable = 1;

usart\_transmit("Humidity readings enabled.\n");

ReadingEnables = ReadingEnables % 100;

} else {

RHenable = 0;

usart\_transmit("Humidity readings disabled.\n");

ReadingEnables = ReadingEnables % 100;

}

if ((ReadingEnables / 10) != 0) { // extract middle bit for Temp data reading enables

TEMPenable = 1;

usart\_transmit("Temperature readings enabled.\n");

ReadingEnables = ReadingEnables % 10 ;

} else {

TEMPenable = 0;

usart\_transmit("Temperature readings disabled.\n");

ReadingEnables = ReadingEnables % 10 ;

}

if (ReadingEnables != 0) { // LSB for light sensor data enables

BRenbale = 1;

usart\_transmit("Light Sensor enabled.\n");

} else {

BRenbale = 0;

usart\_transmit("Light Sensor disabled.\n");

}

}

// echo the command back to the screen

sprintf(echo, "Interval set: %d Hour, %d Minute, %d Second.\n ", hour, minute, sec);

usart\_transmit(echo);

changed = 1;

// hold if still pressed

while((PIND&(1<<PIND2)) == 0);

pressed = 0;

} else { // if command is invalid

usart\_transmit("Old settings used\n");

sprintf(echo, "Interval set: %d Hour, %d Minute, %d Second.\n ", hour, minute, sec);

usart\_transmit(echo);

if (ReadingEnables == 0) {

usart\_transmit("Warning: No readings enabled.\n");

} else {

if ((ReadingEnables / 100) != 0) {

RHenable = 1;

usart\_transmit("Humidity readings enabled.\n");

ReadingEnables = ReadingEnables % 100;

} else {

RHenable = 0;

usart\_transmit("Humidity readings disabled.\n");

ReadingEnables = ReadingEnables % 100;

}

if ((ReadingEnables / 10) != 0) {

TEMPenable = 1;

usart\_transmit("Temperature readings enabled.\n");

ReadingEnables = ReadingEnables % 10 ;

} else {

TEMPenable = 0;

usart\_transmit("Temperature readings disabled.\n");

ReadingEnables = ReadingEnables % 10 ;

}

if (ReadingEnables != 0) {

BRenbale = 1;

usart\_transmit("Light Sensor enabled.\n");

} else {

BRenbale = 0;

usart\_transmit("Light Sensor disabled.\n");

}

}

changed = 1;

// hold if still pressed

while((PIND&(1<<PIND2)) == 0);

pressed = 0;

}

}

}

int main(void)

{

// holding string output

char outstr[100];

// initialize usart

usart\_init();

// initialize ADC

ADC\_init();

// PC3 output buffer set to 0

PORTC &= ~(1<<3);

// PD2 to input PBS

DDRD = 0;

PORTD = 0x04;

// setup interrupt

EICRA = 2; // falling edge on INT0

EIMSK = 1;

sei();

usart\_transmit("Default sampling time: 5sec.\n");

usart\_transmit("All readings enabled by default.\n");

sprintf(outstr, "Interval set: %d Hour, %d Minute, %d Second.\n ", hour, minute, sec);

usart\_transmit(outstr);

while(1) {

if (TEMPenable == 1 || RHenable == 1) {

readDHT();

displayDHT(outstr);

}

if (BRenbale == 1) {

readBrightness();

displayBrightness();

}

delayOverall();

}

}

void usart\_init() {

// Set baud rate

UBRR0H = (unsigned char)(UBRR\_VALUE>>8);

UBRR0L = (unsigned char)UBRR\_VALUE;

// Set frame format to 8 data bits, no parity, 1 stop bit

UCSR0C |= (1<<UCSZ01)|(1<<UCSZ00);

//enable transmission and reception, enable receive interrupt

UCSR0B |= (1<<RXCIE0)|(1<<RXEN0)|(1<<TXEN0);

}

unsigned char usart\_receives() {

// Wait for byte to be received

while(!(UCSR0A&(1<<RXC0))){};

// Return received data

return UDR0; // get char from buffer

}

void usart\_transmit(char arr[]) {

// Uses polling (and it blocks).

int j;

for (j = 0; j <= strlen(arr)-1; j++) {

while (!( UCSR0A & (1<<UDRE0))) {}; // until buffer empty

UDR0 = arr[j]; // put the char to the buffer

}

}

void readDHT() {

uint8\_t DHTbyte = 0;

DDRC |= 1<<3; // set PC3 as output

\_delay\_ms(18.0); // delay for 18ms to initiate the DHT

DDRC &= ~(1<<3); // set PC3 as input

while( (PINC&(1<<PINC3)) != 8 ) ; // while still rising

while(PINC&(1<<PINC3)) ; // while rised

while( (PINC&(1<<PINC3)) != 8 ) ; // while 0 at start

while(PINC&(1<<PINC3)) ; // while 1 at start

int received = 0; // number of bytes decoded

int counter = 0; // number of bits decoded

while(1) { // start decoding

while((PINC&(1<<PINC3)) != 8 ) ; // while in gap

\_delay\_us(35.0); // delay for 35 us

if (PINC&(1<<PINC3)) { // still 1

DHTbyte |= 1 << (7 - counter);

while(PINC&(1<<PINC3)) ; // wait till gap

}

counter++;

if (counter == 8) { // one byte complete

DHTresult[received] = DHTbyte;

DHTbyte = 0;

counter = 0;

received++;

}

if (received == 5) { // all five bytes completed

break;

}

}

}

void ADC\_init() {

ADMUX = (1 << REFS0); // AVCC with external capacitor at AREF pin

ADCSRA = (1 << ADEN) | (1 << ADPS2) | (1 << ADPS1) | (1 << ADPS0); // enable ADC and set prescalar to /128

}

void readBrightness() { //function to get the brightness (0-1023)

ADCSRA |= 1 << ADSC; // start conversion

while (ADCSRA & (1 << ADSC)) { // until finished conversion

}

uint16\_t digitalV = ADC; // get ADC reading

//char temp[20];

//sprintf(temp, "%d", digitalV);

//usart\_transmit(temp);

brightness = digitalV;

}

void displayBrightness() { // will display brightness

if (BRenbale == 1) {

char outstr[30];

sprintf(outstr, "brightness = %d\n", brightness);

usart\_transmit(outstr);

LightnessOutside();

}

}

void LightnessOutside() { // will display message based on brightness

if(brightness < 100){

usart\_transmit("Its night time\n");

}

else if(brightness > 100 && brightness < 300){

usart\_transmit("Its sunrise or dusk\n");

}

else if(brightness > 300 && brightness < 700){

usart\_transmit("Partially cloudy\n");

}

else{

usart\_transmit("The sun it out and shinning\n");

}

}

void displayDHT(char outstr[]) {

// check if result is valid (checksum digit)

if (DHTresult[4] != (DHTresult[0] + DHTresult[1] + DHTresult[2] + DHTresult[3])) {

usart\_transmit("INTERFERENCE WITH DHT READING\n");

} else {

if (RHenable == 1) {

sprintf(outstr, "RH: %d.%d%% \n ", DHTresult[0], DHTresult[1]);

usart\_transmit(outstr);

}

if (TEMPenable == 1) {

sprintf(outstr, "Temp(Celsius): %d.%d \n", DHTresult[2], DHTresult[3]);

usart\_transmit(outstr);

displayTemp();

}

}

}

void displayTemp(){ //will display a message a based on DHT

if (DHTresult[2]< 0) {

usart\_transmit("BRRRR its very cold out\n");

}

else if(DHTresult[2]> 0 && DHTresult[2] < 11){

usart\_transmit("A light jacket would be advised\n");

}

else if(DHTresult[2] > 11 && DHTresult[2] < 20){

usart\_transmit("no need for a jacket enjoy the weather\n");

}

else if (DHTresult[2]> 20 && DHTresult[2] < 30){

usart\_transmit("shorts and t-shirt weather\n");

} else {

usart\_transmit("HOT, stay hydrated\n");

}

}

void delayOverall () { //delay function (timer)

int s = 0;

int m = 0;

int h = 0;

s = sec;

m = minute;

h = hour;

// sec

while (s > 0) {

\_delay\_ms(1000); // delay 1s

s--;

if (changed) {

changed = 0;

return;

}

}

// min

while (m > 0) {

for (int i = 0; i < 60; i++) {

\_delay\_ms(1000);

if (changed) {

changed = 0;

return;

}

}

m--;

}

// hour

while (h > 0) {

for (int i = 0; i < 3600; i++) {

\_delay\_ms(1000);

if (changed) {

changed = 0;

return;

}

}

h--;

}

}

void debounce() { // PBS debounce

int counts = 0;

for (int i = 0; i < 9; i++) {

if ((PIND&(1<<PIND2)) == 0) {

counts++;

}

\_delay\_ms(10.0);

}

if (counts >= 5) {

pressed = 1;

} else {

pressed = 0;

}

}

**8. Appendix B: References**

Atmel, “Atmel 8-bit AVR Microcontroller with 2/4/8K Bytes In-System Programmable Flash”, <<https://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-2586-AVR-8-bit-Microcontroller-ATtiny25-ATtiny45-ATtiny85_Datasheet.pdf>> August 2013.

Embeds, “Programming AVR USART with AVR-GCC”, <<https://embedds.com/programming-avr-usart-with-avr-gcc-part-1/>> January 2015.

LearnCpp, “Passing arguments by address” ,<<https://www.learncpp.com/cpp-tutorial/passing-arguments-by-address/>> December 2020.

MakerPro, “Use and AVR microcontroller incorporated ADC to take analog readings from your surrounding”, <<https://maker.pro/custom/tutorial/how-to-take-analog-readings-with-an-avr-microcontroller>> June 2018.

Stackoverflow, “Using floats with sprintf() in embedded C”, <<https://stackoverflow.com/questions/905928/using-floats-with-sprintf-in-embedded-c>> January 2020.

Stackoverflow, “ How does strtok() split the string into tokens in C”, <<https://stackoverflow.com/questions/3889992/how-does-strtok-split-the-string-into-tokens-in-c>> April 2020.

Tutorialspoint, “ C library function - sscanf()”, <<https://www.tutorialspoint.com/c_standard_library/c_function_sscanf.htm>> January 2021.