CPE301 – FALL 2020

MidTerm 1

Student Name: Jason Villanueva

Student #: 2001255420

Student Email: villaj21@unlv.nevada.edu

Primary Github address: <https://github.com/jasonvillanuevagit/submission_designAssignments->

Directory: <https://github.com/jasonvillanuevagit/submission_designAssignments-/tree/master/Midterm1> Submit the following for all Labs:

1. **COMPONENTS LIST AND CONNECTION BLOCK DIAGRAM w/ PINS**

Atmel Studio 7.0 Atmega328PB-Xmini PC LM-35

- Assembler

- Simulator

- Debugger

- Terminal

A screenshot of a computer

Description automatically generated

1. **INITIAL/MODIFIED/DEVELOPED CODE OF TASK 1/A**

MidTerm\_1 C Code

#define F\_CPU 16000000

#include <avr/io.h>

#include <avr/interrupt.h>

#include <stdio.h>

#define state uint8\_t

#define idleState 0x00U

#define helpState 0x01U

#define tempState 0x02U

#define ledControlState 0x03U

#define ledDelayState 0x04U

#define stringState 0x05U

#define BAUD 9600

#define BAUDPRESCALAR ((F\_CPU/(BAUD))/16 - 1)

#define ADC\_conversion (110.0/1024.0)

#define bool unsigned int

#define true 0x01

#define false 0x00

void initTimer1CTC();

void startTimer1CTC\_OC();

void stopTimer1CTC\_OC();

void waitForInput();

void initADC(uint8\_t adcCh, uint8\_t adcRef);

void adcStartConversion();

uint16\_t readADC();

void initUART(uint8\_t umsel, uint8\_t upm, uint8\_t usbs, uint8\_t ucsz);

float convert\_adc\_to\_c(uint16\_t adcValue);

float convert\_c\_to\_f(float celsius);

void convert\_float\_to\_string(float floatNum);

void sendString\_UART();

uint8\_t convertToState();

volatile static uint8\_t dataRegister;

volatile static uint16\_t adcRegister;

volatile static uint8\_t OC\_Second;

//declaring bools

volatile static bool flashingTOGGLE = true;

volatile static bool uartBUSY = false;

volatile static bool adcDATA\_READY = true;

volatile static bool adcBUSY = false;

volatile static bool flashingREADY = true;

volatile static bool newInput = false;

volatile static char\* dataString;

static char charConversion[20];

//main loop

int main(void){

volatile state currentState = idleState;

volatile state nextState = helpState;

volatile state stateRegister;

DDRB = 0xFF;

PORTB = 0xFF;

initADC(0x04, 0x03);

initTimer1CTC();

initUART(0x00, 0x00, 0x00, 0x03);

sei();

while (1){

currentState = nextState;

//hanndle input

if(currentState == idleState){

if(newInput == true){

newInput = false;

nextState = convertToState();

}

}

else if(currentState == helpState){

nextState = idleState;

//only way strings would output correctly

dataString = "help menu \nenter key \nh : help menu \nt : temperature in C \nT : temperature in F \no : turn on LED at port B2 \nO : turn off LED at port B2 \ns : send a string delay for LED at port B3 \ni : send an integer delay for LED at port B3 \n\r";

sendString\_UART();

}

else if(currentState == tempState){

nextState = idleState;

adcStartConversion();

if((char)dataRegister == 't'){

dataString = "\n";

sendString\_UART();

convert\_float\_to\_string(convert\_adc\_to\_c(readADC()));

sendString\_UART();

}

else if((char)dataRegister == 'T'){

dataString = "\n";

sendString\_UART();

convert\_float\_to\_string(convert\_c\_to\_f(convert\_adc\_to\_c(readADC())));

sendString\_UART();

}

}

else if(currentState == ledControlState){

nextState = idleState;

if((char)dataRegister == 'o'){

PORTB &= ~(1<<2);

}

else if((char)dataRegister == 'O'){

PORTB |= (1<<2);

}

}

else if(currentState == ledDelayState){

nextState = idleState;

waitForInput();

if((char)dataRegister == 's'){

OC\_Second = OC\_Second - 48;

if(OC\_Second == 0){

stopTimer1CTC\_OC();

PORTB |= (1U<<3);

}

else{

OCR1A = (uint16\_t)((F\_CPU)/(1024)\*OC\_Second - 1);

startTimer1CTC\_OC();

}

}

else if((char)dataRegister == 'i'){

if(OC\_Second == 0){

stopTimer1CTC\_OC();

PORTB |= (1U<<3);

}

else{

OCR1A = (uint16\_t)((F\_CPU)/(1024)\*OC\_Second - 1);

startTimer1CTC\_OC();

}

}

}

}

}

//state transitions

uint8\_t convertToState(){

if(dataRegister == 'O' || dataRegister == 'o' ){

return ledControlState;

}

else if(dataRegister == 'h'){

return helpState;

}

else if(dataRegister == 'i' || dataRegister == 's'){

return ledDelayState;

}

else if(dataRegister == 'T' || dataRegister == 't'){

return tempState;

}

else return idleState;

}

//initialize adc

void initADC(uint8\_t adcCh, uint8\_t adcRef){

ADMUX |= (adcCh << 0) | (adcRef << 6);

ADCSRA |= (0x07<<0);

}

//start conversion

void adcStartConversion(){

adcBUSY = true;

ADCSRA |= (1U<<7) | (1U<<6) | (1U<<3);

while(adcBUSY){};

}

//read adc

uint16\_t readADC(){

return (ADCL | (ADCH<<8));

}

//initialize UART

void initUART(uint8\_t umsel, uint8\_t upm, uint8\_t usbs, uint8\_t ucsz){

UBRR0H = (uint8\_t)(0x00);

UBRR0L = (uint8\_t)(BAUDPRESCALAR);

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

UCSR0C = (umsel << UMSEL00) | (upm << UPM00) | (usbs << USBS0) | (ucsz << UCSZ00);

}

//converting adc reading to float

float convert\_adc\_to\_c(uint16\_t adcValue){

return ((ADC\_conversion) \* (float)(adcValue));

}

//C to F

float convert\_c\_to\_f(float celsius){

return ((celsius \* 9.0/5.0) + 32);

}

//float to string

void convert\_float\_to\_string(float floatNum){

snprintf(charConversion, sizeof(charConversion), "%f\n\r", floatNum);

dataString = charConversion;

}

//send string

void sendString\_UART(){

UCSR0B |= ((1 << TXCIE0) |

(1 << UDRIE0) |

(1 << TXEN0)); //Enable interrupts and tx

}

//initialize timer 1 CTC

void initTimer1CTC(){

TCCR1B |= (1U<<3);

}

//start timer

void startTimer1CTC\_OC(){

TCNT1 = 0x00;

TIMSK1 |= (1U<<1);

TCCR1B |= (0x05<<0);

}

//stop timer

void stopTimer1CTC\_OC(){

TCCR1B &= ~(0x07<<0);

TIMSK1 &= ~(1U<<1);

}

//sub-idle in delay state

void waitForInput(){

dataString = "\n enter delay in seconds (1-4), 0 to remove delay. \n\r";

sendString\_UART();

flashingREADY = false;

while(!flashingREADY){};

}

//flash

ISR (TIMER1\_COMPA\_vect){

flashingTOGGLE = !flashingTOGGLE;

if(flashingTOGGLE == true) PORTB |= (1U<<3);

else PORTB &= ~(1U<<3);

}

ISR(ADC\_vect){

ADCSRA &= ~((1U<<3) | (1U<<7));

adcBUSY = false;

}

//wait to deactivate TX until buffer is empty

ISR(USART0\_TX\_vect){

UCSR0B &= ~((1 << TXCIE0) | (1 << TXEN0));

}

ISR(USART0\_RX\_vect){

if(!flashingREADY){

flashingREADY = true;

OC\_Second = UDR0;

}

else{

dataRegister = UDR0;

newInput = true;

}

}

//send date until array empty, increase string location, else disable interrupts

ISR(USART0\_UDRE\_vect){

if(\*dataString != '\0'){

UDR0 = \*dataString;

dataString++;

}

else{

UCSR0B &= ~(1 << UDRIE0);

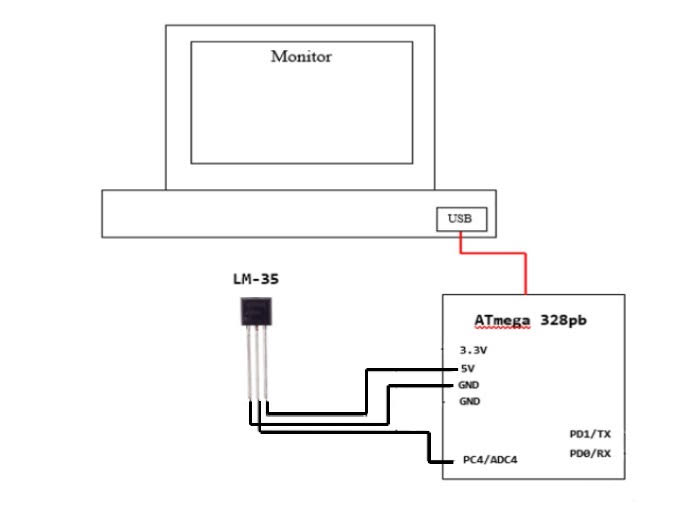
}

}

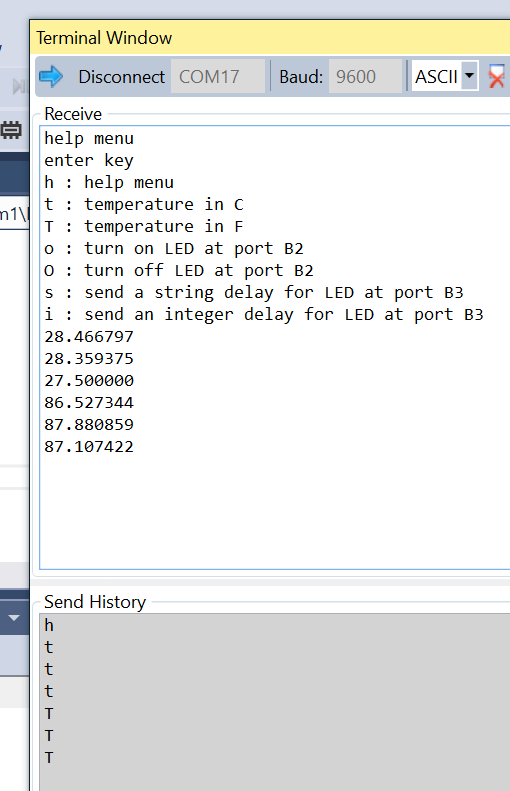
1. **DEVELOPED MODIFIED CODE OF TASK 2/A from TASK 1/A**

N/A

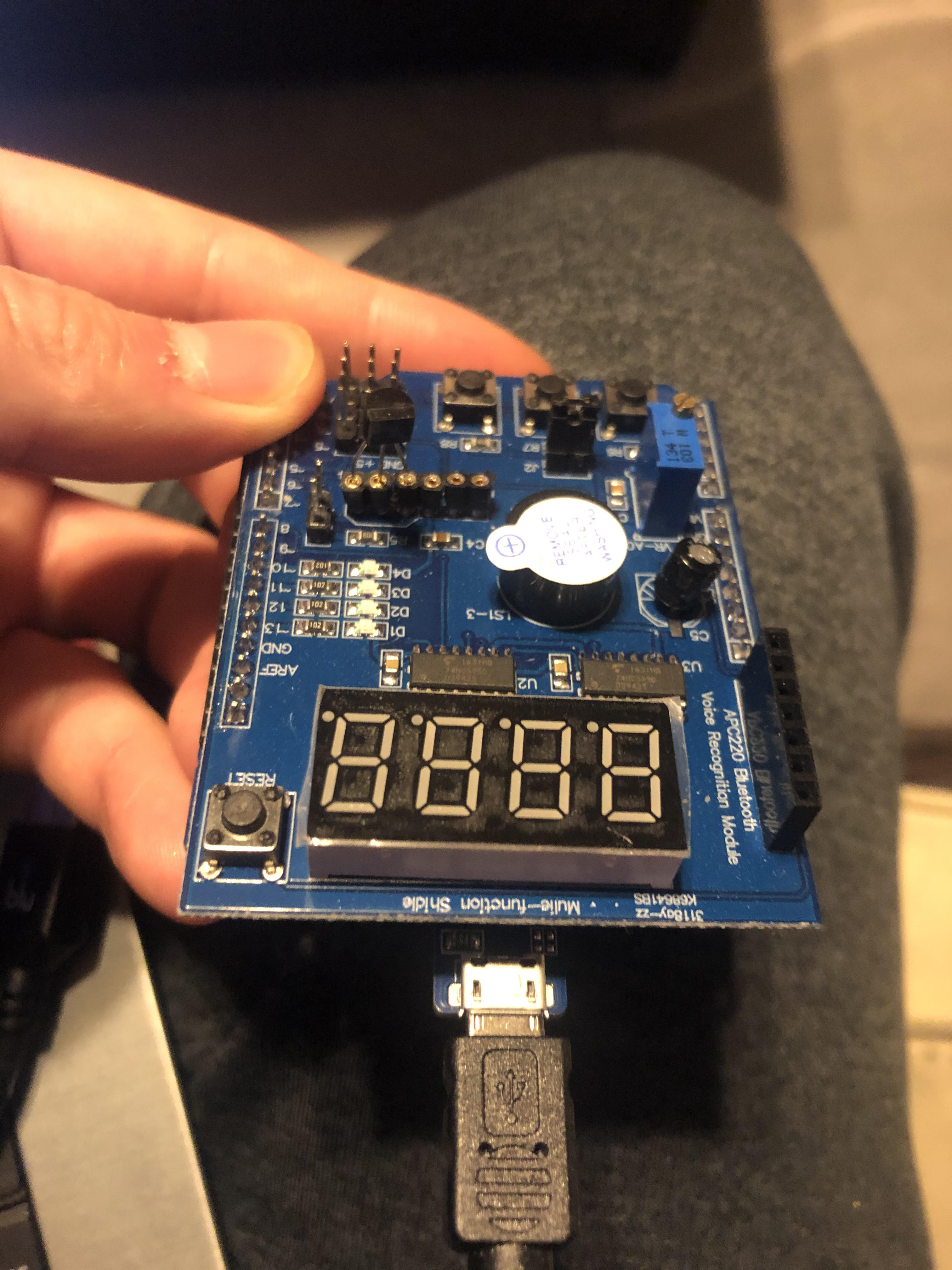
1. **SCHEMATICS**

****

1. **SCREENSHOTS OF EACH TASK OUTPUT (ATMEL STUDIO OUTPUT)**



1. **SCREENSHOT OF EACH DEMO (BOARD SETUP)**

****

1. **VIDEO LINKS OF EACH DEMO**

YouTube Demo Link

<https://youtu.be/4nBFtF5tDLE>

1. **GITHUB LINK OF THIS DA**

<https://github.com/jasonvillanuevagit/submission_designAssignments-/tree/master/Midterm1>

**Student Academic Misconduct Policy**

<http://studentconduct.unlv.edu/misconduct/policy.html>

“This assignment submission is my own, original work”.

**Jason Villanueva**