

Midterm 2

Student Name: David Nakasone

Student #: 2001646072

Student Email: nakasd3@unlv.nevada.edu

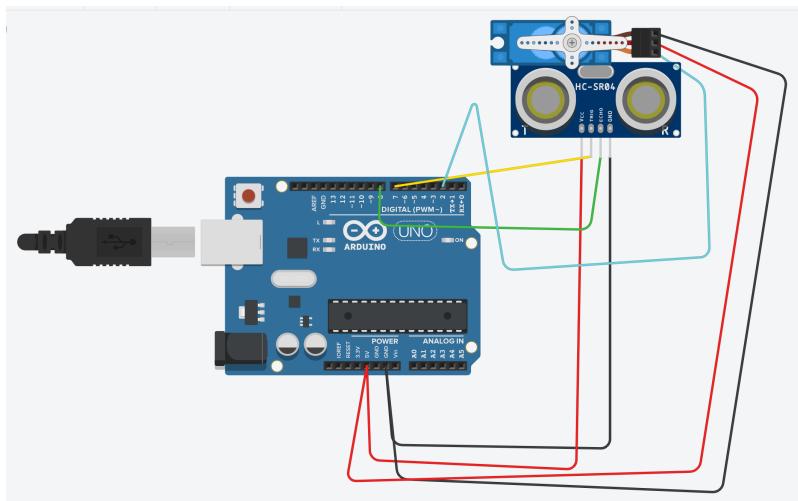
Primary Github address: <https://github.com/davenakasone>

Directory: https://github.com/davenakasone/cpe301_David_Nakasone

1. COMPONENTS LIST AND CONNECTION BLOCK DIAGRAM w/ PINS

List of Components used: ultra sonic sensor and servo motor

Block diagram with pins used in the Atmega328P:



Servo uses ground, $V_{CC} = 5V$ and PWM on PD2

Ultrasonic sensor uses ground, $V_{CC} = 5V$, PB0 echo, PD7 trigger

2. INITIAL/MODIFIED/DEVELOPED CODE, all tasks

The code covers all tasks.

The code was successful:

The screenshot shows the Atmel Studio interface with the main.c file open in the Data Visualizer window. The code defines pin mappings for a microcontroller, including servo, ultrasonic, and UART pins. Below the code editor is the Error List window, which displays 0 errors, 0 warnings, and 0 messages. The bottom half of the screen is the Output window, which shows the build log. The log indicates a successful build with no errors or warnings. A green checkmark is drawn over the 'Build succeeded' message in the output window.

```
/*  
 * cpe301, midterm2, all tasks are contained  
  
servo  
    Vcc -> external source  
    gnd -> joined to MCU, pick any  
    PWM -> PD2  
  
ultrasonic  
    Vcc      -> MCU "5V"  
    gnd      -> MCU "gnd"  
    echo     -> PB0  
    trigger  -> PD7  
  
uart0  
    tx -> PD0  
    rx -> PD1  
  
servo, HS 5065MG, f = 200Hz, min: 750us, max: 2250us  
using timer3, prescaled x8  
fast PWM mode 14  
PD2 is PWM generator (dedicated OC3B), keeps PD0 for UART  
  
100 %  
  
Error List  
Entire Solution < 0 Errors | 0 Warnings | 0 Messages | Build + IntelliSense -<br/>Description  
  
Output  
Show output from: Build  
Done executing task "RunCompilerTask".  
Task "RunOutputFileVerifyTask".  
Program Memory Usage : 4828 bytes 14.7 % Full  
Data Memory Usage : 212 bytes 10.4 % Full  
Warning: Memory usage estimation may not be accurate if there are sections other than .text sections in ELF file  
Done executing task "RunOutputFileVerifyTask".  
Done building target "CoreBuild" in project "GccApplication1.cproj".  
Target "PostBuildEvent" skipped, due to false condition: '$(PostBuildEvent)' != '' was evaluated as ('' != '').  
Target "Build" in file "C:\Program Files (x86)\Atmel\Studio\7.0\VsA\vs.common.targets" from project "C:\Users\uni\Desktop\GccApplication1\GccApplication1.cproj".  
Done building target "Build" in project "GccApplication1.cproj".  
Done building project "GccApplication1.cproj".  
  
Build succeeded.  
----- Build: 1 succeeded or up-to-date, 0 failed, 0 skipped -----
```

C code:

```
/*  
 * cpe301, midterm2, all tasks are contained  
  
servo  
    Vcc -> external source  
    gnd -> joined to MCU, pick any  
    PWM -> PD2  
  
ultrasonic  
    Vcc      -> MCU "5V"  
    gnd      -> MCU "gnd"  
    echo     -> PB0  
    trigger  -> PD7  
  
uart0  
    tx -> PD0
```

```

rx -> PD1

servo, HS 5065MG, f = 200Hz, min: 750us, max: 2250us
using timer3, prescaled x8
fast PWM mode 14
PD2 is PWM generator (dedicated OC3B), keeps PD0 for UART
share the common ground, but use external power just in case

HC-SR04 ultra-sonic sensor
PD7 for trigger
PB0 for echo, must be use since timer1 has ICP1 on this pin
sound will probably at 344 m/s where you use this
for each tik, (1 / f_cpu) * prescale = T
the sound travels to the target and back
distance = (tiks / 2) * 344 meters, about (1/16e6)*(344*100)/2 = 930 , the const for distance
in cm, must tune it
"distance = rate * time ... m = m/s * s"

start processing 4
orcFont = loadFont("OCRAExtended-30.vlw"); is in the folder
myPort = new Serial(this,"COM4", 9600); // starts the serial communication check com port
...set the macro, processing uses different data types

1500:33:4470 == 90 steps
0:2:180 == 90 steps
*/
#define PROCESSING 11 // toggle here
#ifndef PROCESSING
#define PROCESSING_STOP 180
#define PROCESSING_START 0
#define PROCESSING_STEP 2
#endif

#define F_CPU 16000000UL
#define BAUD 9600

```

```

#define BAUD_PRESCALE (((F_CPU / (BAUD * 16UL))) - 1)

#define SERVO_WIDTH 10000 // f_pwm = 200 Hz, 200 = f_cpu / (N * (TOP+1) -> TOP = 9999
#define SERVO_START 1500 // 750us min, f = f_cpu / N = 2e6, T = 1/f = 0.5us -> 750/0.5 =
1500...don't max out
#define SERVO_STOP 4470 // 2250us max, " " 2250/0.5 =
4500...don't max out
#define SERVO_HANG 100 // ms, change based on step size, the wait time
#define SERVO_STEP 33 // change as needed, give resolution
#define SERVO_RETURN 500 // give it enough time to return from a sweep

#define TRIGGER_WIDTH_US 10 // duration of the trigger pulse in micro seconds
#define TIME_OUT 10 // overflows allowed to occur before continuing program
#define TUNER 1024 // should be around 930, see calculation above

#define PWM_PIN 2 // PD2 has OC3B
#define ECHO_PIN 0 // on PORTB, must be on PB0 for ICR1
#define TRIGGER_PIN 7 // on PORTD

#define HELP_BUF 128
#define TRUE 888
#define FALSE 999

#include <avr/io.h>
#include <avr/interrupt.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <util/delay.h>
#include <util/atomic.h>

uint16_t servo_counter;
volatile char helper[HELP_BUF];
volatile int timer_overflows;

```

```

volatile double distance;
volatile long tik_count;
int read_attempt;

void read_distance();
void usart_putc (char send_char);
void usart_puts (const char* send_str);

int main()
{
    #ifdef PROCESSING
        uint16_t temp_a;
        uint16_t temp_b;
        uint16_t process_step = 0;
        uint16_t process_distance;
    #endif

    DDRD |= (1 << TRIGGER_PIN);      // trigger signal as output
    PORTB |= (1 << ECHO_PIN);       // turn on pull up resistor for echo capture as input

    UCSR0B |= (1 << RXEN0) | (1 << TXEN0);      // turn on USART module, receive and transmit enabled
    UCSR0C |= (1 << UCSZ00) | (1 << UCSZ01);      // configure: asynchronous, 8-bit data, 1-bit stop
    UBRR0H |= (BAUD_PRESCALE >> 8);              // sets baud rate
    UBRR0L |= BAUD_PRESCALE;

    TIMSK1 |= (1 << TOIE1);      // enable timer1 overflow interrupt
    TCCR1A = 0;                      // normal operation

    DDRD |= (1 << PWM_PIN);          // dedicated PWM pin for timer 3
    PORTD |= (1 << PWM_PIN);          // this must be done to disable OC4B, the
pin is shared
    ICR3 = SERVO_WIDTH;                // the "TOP", makes the time period, OCR3A
makes duty cycle
    TCCR3A |= (1 << COM3B1) | (1 << WGM31);      // non inverting, mode14
    TCCR3B |= (1 << WGM33) | (1 << WGM32) | (1 << CS31); // mode 14, prescale 8
    servo_counter = SERVO_START;
}

```

```
uart_puts("\r\n");
uart_puts("initialized, program begins...\r\n");
sei();

while(1)
{
    while (servo_counter < SERVO_STOP)
    {
        read_attempt = 0;
        OCR3B = servo_counter;
        read_distance();
        _delay_ms(SERVO_HANG);
        while (distance == 0 && read_attempt < TIME_OUT)
        {
            read_distance();
            read_attempt++;
            _delay_ms(SERVO_HANG);
        }
        #ifdef PROCESSING
        //
        process_step = process_step + PROCESSING_STEP;
        sprintf(helper, "%d", process_step);
        uart_puts(helper);
        uart_puts(",");
        sprintf(helper, "%d", (uint16_t)distance);
        uart_puts(helper);
        uart_puts(".");
        if(process_step > PROCESSING_STOP) {process_step = PROCESSING_START;}
        _delay_ms(SERVO_HANG);
        //
        #else
        uart_puts("\r\n");
        sprintf(helper, HELP_BUF-1, "tik#: %d , distance: %0.3lf cm",
servo_counter, distance);
        #endif
    }
}
```

```

        usart_puts(helper);
        usart_puts("\r\n");
        _delay_ms(SERVO_HANG);

#endif

servo_counter = servo_counter + SERVO_STEP;

}

while (servo_counter > SERVO_START)
{
    read_attempt = 0;
    OCR3B = servo_counter;
    read_distance();
    _delay_ms(SERVO_HANG);
    while (distance == 0 && read_attempt < TIME_OUT)
    {
        read_distance();
        read_attempt++;
        _delay_ms(SERVO_HANG);
    }
    #ifdef PROCESSING
    //
    process_step = process_step - PROCESSING_STEP;
    sprintf(helper, "%d", process_step);
    usart_puts(helper);
    usart_puts(",");
    sprintf(helper, "%d", (uint16_t)distance);
    usart_puts(helper);
    usart_puts(".");
    if(process_step > PROCESSING_STOP) {process_step = PROCESSING_START;}
    _delay_ms(SERVO_HANG);
    //
    #else
        usart_puts("\r\n");
        snprintf(helper, HELP_BUF-1, "tik#: %d , distance: %0.3lf cm",
servo_counter, distance);
        usart_puts(helper);
    #

```

```

        usart_puts("\r\n");

#endif

servo_counter = servo_counter - SERVO_STEP;

}

if (servo_counter < SERVO_START-5 ||

    servo_counter > SERVO_STOP+5    )

{

    servo_counter = SERVO_START;

    _delay_ms(SERVO_RETURN);

}

}

return EXIT_FAILURE;
}

////~~~~

void read_distance()

{
    memset(helper, '\0', HELP_BUF);

    timer_overflows = 0;

    distance = 0;

    tik_count = 0;

    PORTD |= (1 << TRIGGER_PIN);           // pulse begin

    _delay_ms(TRIGGER_WIDTH_US);             // pulse high, duration in us

    PORTD &= ~(1 << TRIGGER_PIN);         // pulse end

    TCNT1 = 0;                            // clear timer1 counter

    TCCR1B = (1 << ICES1) | (1 << CS10); // capture rising edge, no prescaling

    if (TIFR1 & (1 << ICF1)) {TIFR1 |= (1 << ICF1);} // clear input capture flag

    timer_overflows = 0;                  // overflows reset

    while ((TIFR1 & (1 << ICF1)) == 0)      // echo begins -> wait for rising edge

    {

        if (timer_overflows > TIME_OUT) {break;} // time out causes break

    }

    timer_overflows = 0;                  // echo high, timer_overflows must reset
}

```

```

TCNT1 = 0;                                // clear timer1 counter
TCCR1B = (1 << CS10);                   // falling edge capture, no prescaling
if (TIFR1 & (1 << ICF1)) {TIFR1 |= (1 << ICF1);} // clear input capture flag
while ((TIFR1 & (1 << ICF1)) == 0)          // echo ends -> wait for falling edge
{
    if (timer_overflows > TIME_OUT) {break;} // time out causes break
}
if (timer_overflows < TIME_OUT)             // a valid measurement occurred
{
    tik_count = ICR1 + 65535 * timer_overflows;
    distance = (double)tik_count / TUNER;
}
}

////~~~~

void usart_putc (char send_char)
{
    while ((UCSR0A & (1 << UDRE0)) == 0) {}
    UDR0 = send_char;
}

////~~~~

void usart_puts (const char* send_str)
{
    while (*send_str)
    {
        usart_putc(*send_str++);
    }
}

```

```

/////~~~~

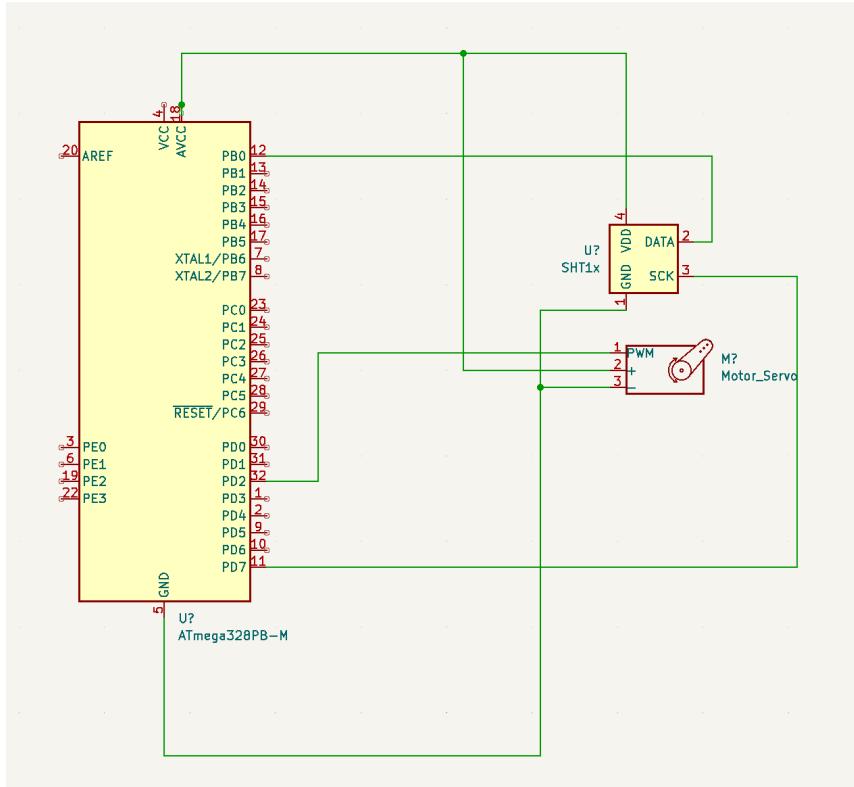
ISR(TIMER1_OVF_vect)
{
    timer_overflows++;

    if (timer_overflows > TIME_OUT*TIME_OUT ||
        timer_overflows < 0)
    {
        timer_overflows = 0;
    }
}

/////////~~~~~~END> main.c

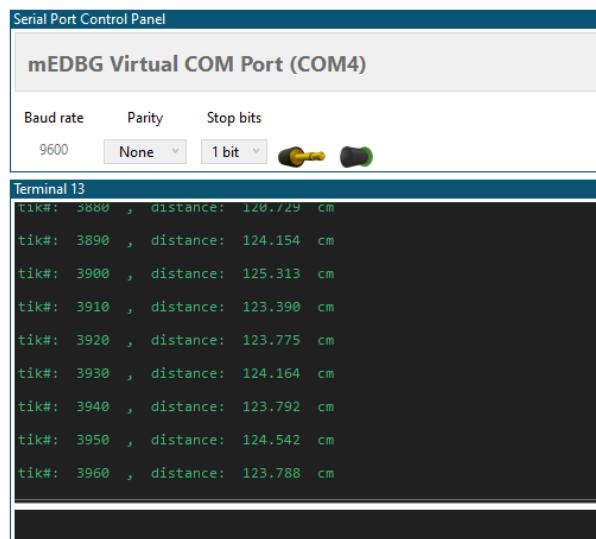
```

3. SCHEMATICS



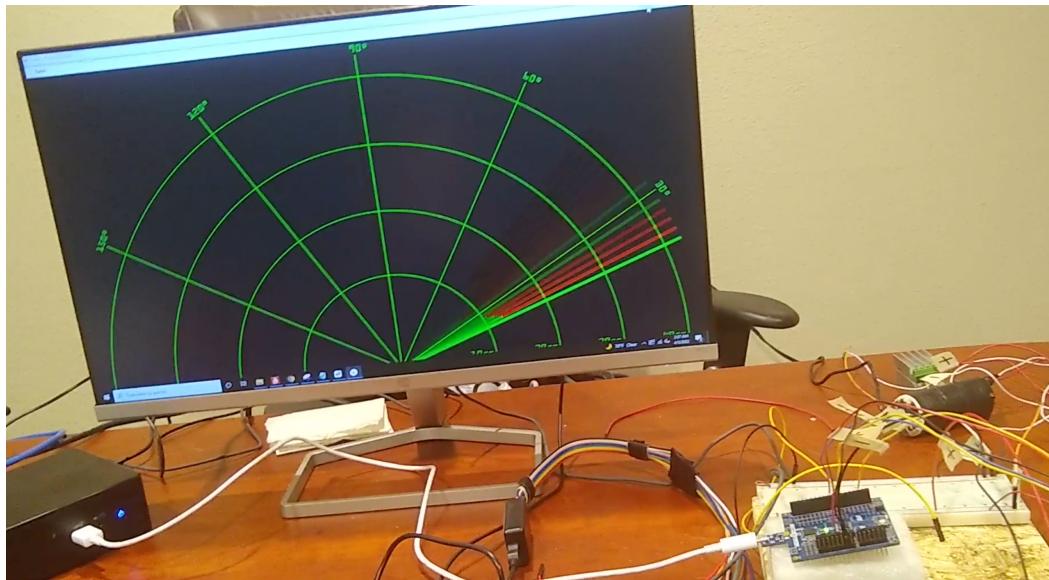
4. SCREENSHOTS OF EACH TASK OUTPUT (ATMEL STUDIO OUTPUT)

task 1, raw data:



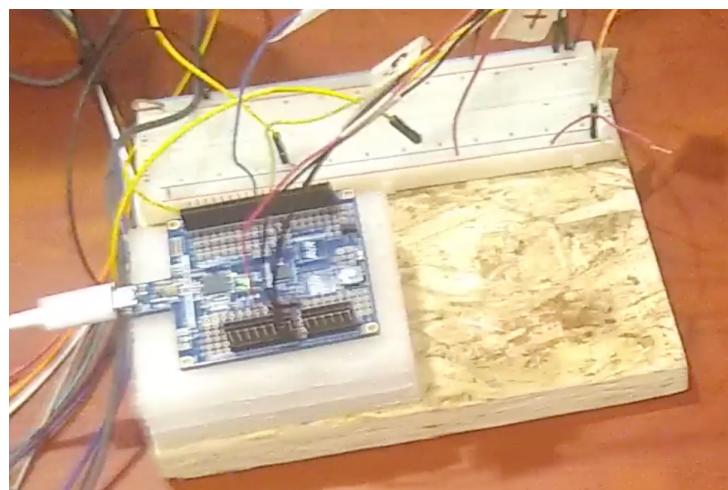
Serial Port Control Panel
mEDBG Virtual COM Port (COM4)
Baud rate: 9600 Parity: None Stop bits: 1 bit
Terminal 13
tik#: 3880 , distance: 120.729 cm
tik#: 3890 , distance: 124.154 cm
tik#: 3900 , distance: 125.313 cm
tik#: 3910 , distance: 123.390 cm
tik#: 3920 , distance: 123.775 cm
tik#: 3930 , distance: 124.164 cm
tik#: 3940 , distance: 123.792 cm
tik#: 3950 , distance: 124.542 cm
tik#: 3960 , distance: 123.788 cm

task 2, radar:



5. Screenshot of each demo (board setup)

The board, for both tasks:



The module:



6. VIDEO LINKS OF EACH DEMO

task1, raw demonstration <https://youtu.be/j68XFYX07bM>

task2, using "processing" <https://youtu.be/Ds2jTorLmts>

7. GITHUB LINK OF THIS DA

https://github.com/davenakasone/cpe301_David_Nakasone/tree/main/Midtermz/Midterm2

Student Academic Misconduct Policy

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

"This assignment submission is my own, original work".

David Nakasone