# MIDDLE EAST TECHNICAL UNIVERSITY

## ELECTRICAL-ELECTRONICS ENGINEERING DEPARTMENT

## EE400 SUMMER PRACTICE REPORT


**Student Name:** Elmas SOYAK

**Student ID:** 1626605


**SP Company Name:** TAI-TUSAŞ

**Company Division:** Space Systems

**SP Date:** 23.01.2012-17.02.2012

**Submission Date:** 14.03.2012

## TABLE OF CONTENTS

# 1. INTRODUCTION

I have performed my summer practice in TAI-TUSAŞ (Turkish Aerospace Industries-Türkiye Uzay ve Havacılık Anonim Şirketi). It is the center of technology in design, development, manufacture, integration of aerospace systems, modernization and after sales support in Turkey. My internship lasted 20 days. Celal Metehan Aydın, who is an electronics engineer in TAI was our supervisor and he arranged our internship program. I preferred TAI since I performed my previous SP in TAI and it was very efficient for me. During that internship, I learned many things about electronics engineering and observed the facilities of the company. Furthermore, I knew that TAI was following the advancements in electronics closely. In my second internship, I was charged with some projects and I completed them. The projects are mainly about the embedded system design. First week, I studied PIC family of microprocessors and started programming in PIC C language. I was given a project about PIC microprocessors. Then I continued writing firmware in Assembly language which helped me better understand the internal structure of the microcontroller. Then I made a circuit design for a brake box. This project required to know the electronics well. I had to learn about the electronic components, circuit design and draw a PCB design on the computer. The last mission was to receive analog data coming from pedals, and send it to a computer. This data controls the actions of the brake system. I used PIC microcontrollers for this project, too. All these missions provided a good progression for me. At the end of the SP, I had got invaluable experiences and knowledge about the electronics. I learned programming PIC microcontrollers, writing in CCS C and Assembly language. I used CCS C Compilers and MPLAB for compiling the codes. Moreover, I learned making circuit designs and drawing its PCB in Altium Designer.

In this report, I start with introducing the company. After that, I reported what I have performed and learned during SP. I have given detailed information about what I have done in projects: codes, simulations, circuitry, etc. At this part, I used many figures and photographs to better explain my works. After that, I summarized my report in “Conclusion” part. I included the sources of the documents from which I took help in the “References” part.

# 2. DESCRIPTION OF THE COMPANY

## 2.1. COMPANY NAME

TAI-TUSAŞ (Turkey Aerospace Industries Inc. - Türkiye Havacılık ve Uzay Anonim Şirketi)

## 2.2. COMPANY LOCATION

**Address-1:** Fethiye Mahallesi, Havacılık Bulvarı No: 17 06980 Kazan-ANKARA / TÜRKİYE

**Address-2:** Teknokent ODTÜ/ANKARA

**Phone:** +90 312 811 1800

**Fax:** +90 312 811 1425

## 2.3. GENERAL DESCRIPTION OF THE COMPANY

TAI was established on 15 May 1984 and is located in Akıncı-Ankara. TAI's modern aircraft facilities, which cover an area of 5 million square meters with an industrial facility of over 200,000 square meters under roof, is furnished with high technology machinery and equipment that provide extensive manufacturing capabilities. Furthermore, in order to develop a ground for defence aerospace skills, and create synergy among small to medium enterprises and universities, a significant part of engineering-based activities are conducted at the Techno Park located in the Middle East Technical University. The quality system of the Company meets the stringent world standards including NATO AQAP-2110, ISO-9001:2000 and AS EN 9100. TAI has a total of 3000 employees, 1200 of whom are engineers. The organizational chart of the company is shown in Figure 1.

The shareholders of the company are; the Turkish Armed Forces Foundation, the Undersecretariat for Defense Industries and Turkish Aeronautical Association. TAI, which participates in global-scale design and development programs, is also engaged in the design and manufacturing of structural components with leading international aerospace companies. With its proven experience, TAI is a uniquely qualified supplier for Aermacchi, AgustaWestland, Airbus, Boeing, EADS CASA, Eurocopter, Lockheed Martin, Northrop Grumman, MDHI, Sikorsky and many more. Since its establishment, TAI's employees, baring in mind their target to develop not only Turkey's national power, but also the technological capacity that will support the military capacity, have been carrying out their activities to develop the necessary capabilities and products in order to meet the aerospace requirements of the Turkish Armed Forces with "indigenous" systems. To this end, TAI, in line with its vision and mission statements, has established a modern aerospace facility, and successfully realized the co-production of F-16 fighters, CN-235 light transport/maritime patrol/surveillance aircraft, SF-260 trainers, Cougar AS-532 general purpose helicopters. With its proven experience and know-how, TAI has improved its capabilities in the fields of design, production, modernization, modification and systems integration of fixed and rotary wing air platforms, unmanned aerial vehicles and satellite.

Being the main contractor of ATAK - Attack/Tactical Reconnaissance Helicopters Program, TAI will not only customize, but also produce and provide integrated logistics support of the helicopter in accordance with the user needs. TAI, which is the prime contractor of the Turkish Unmanned Aerial Vehicle (MALE) production program, is engaged in design and development of Primary & Basic Trainer (HÜRKUŞ) Aircraft. Baring in mind its target to provide


**Figure 1: Organizational Structure of TAI**

the Turkish Armed Forces with indigenous systems, TAI continues its activities regarding the design and production of Turkish Primary and Basic Trainer Aircraft (HÜRKUŞ) and Turkish Indigenous Medium Altitude Long Endurance (MALE) Unmanned Aerial Vehicle (TIHA). TAI, which actively participates in the custom satellite development program of Turkey, will be the

local integrator company for the International Satellite Acquisition Programs. To this end, a new Satellite Assembly and Integration Test Facility will be built.

In addition to indigenous programs, TAI's core business also includes modernization, modification and systems integration programs and after sales support of both fixed and rotary wing military and commercial aircraft that are in the inventory of Turkey and friendly countries. TAI is the prime contractor of the avionic modernization programs of the C-130 transport aircraft and T-38 11 TAI participates, as a partner, in the global scale Joint Strike Fighter (JSF/F-35) and A400M6 design and development programs.

Being the shareholder of Airbus Military S.L., as National Industrial Institution, TAI has been participating in the design and development activities of A400M with the leading European aerospace companies namely; Airbus, EADS and FLABEL. TAI, which also keeps on developing its capabilities in commercial aviation, has also become a full risk-sharing partner of Airbus in the A350XWB program for the aileron work package. Furthermore, by utilizing capabilities of the Turkish Armed Forces' Maintenance Centers, TAI also gives maintenance, repair and overhaul services to its customers.

Determined to keep abreast of global technological developments and secure its place among the major aerospace companies, TAI aims to lead Turkey to new horizons in aviation in the 21st century.

## 3. FIRST PROJECT

I started my summer practice in TAI in 23rd January in Akıncı. First day, I attended an orientation program. We were informed about the facility, job safety, and information safety, product description and FOD (Foreign Object Damage). Then we were appointed to the related division of the company. For the rest of my internship, I worked at Hardware Engineering department which is working for Space Systems. Firstly, I was expected to learn using microcontrollers for specific purposes. The first one was to design a capacitive sensing circuit with a PIC microcontroller. A conducting cable connected to a pin of the microcontroller serves as a sensor. When something or somebody (a dielectric material) touches the cable, the microcontroller must sense and give an appropriate output. PIC 16F1937, the one I used, is a new product of Intel company and has a built-in capacitive sensing hardware. Before explaining my source code, I will give brief information about capacitive sensing:

**Capacitive Sensing Mechanism:** Each material in the space has dielectric constant and it creates a capacitive effect for the material in various values. This is the base of capacitive sensing mechanism. The circuits naturally have a parasitic capacitance to the ground. When the finger of the person approaches to the circuit, it creates an additional parallel capacitance. Therefore, total capacitance increases and it needs to be measured in some way. An oscillator is

used for this purpose. By applying a voltage to the capacitive end and measuring how many times it has been charged/ discharged in a fixed time interval, the relative capacitance can be measured. At each charging/discharging, the oscillator oscillates and the positive/negative edges are counted. If there is a dielectric material connected, then the counted number is less than the nominal value since the capacitive material increases the charging/discharging time.

**PIC 16F1937 Capacitive Sensing Module:** This microcontroller has a built-in Capacitive Sensing Module. Capacitive sensing operations can be done in 16 channels of the microcontroller. In the module, there exists a capacitive sensing oscillator to sense the charging/discharging of the material. The number of oscillations can be stored in Timer 1. The fixed time interval can be chosen as one overflow length of Timer 0. Timer 0 starts counting from 0 to 255 (it is an 8-bit register), during this interval Timer 1 counts the oscillations. After Timer 0 overflows (i.e. after it reaches 255, it becomes 0 again), an interrupt occurs and the comparison of the Timer 1 value is achieved there. Timer 1 value is compared with the nominal value during the interrupt. If it is less than the nominal value, it means that the selected channel of microcontroller is connected to a capacitive material.

**Timers and Interrupts:** As mentioned above, Timer 0 is chosen to provide the fixed time base. The oscillations are counted while Timer 0 goes from 0 to 255 (in decimal). When it becomes 255, Timer 0 overflow interrupt is called and comparison is performed there. Timer 1 is used to count the oscillations due to the capacitive charging/discharging of the material. To perform this function, some initializations needed to be done on the Timer 0 and Timer 1 registers. The initializations are explained in detail in the code. Before returning from interrupt service routine, Timer 0 and Timer 1 are cleared to zero, a new Timer 0 interrupt is enabled and a new count starts. Timer 1 gate is needed when capacitive sensing module is activated. Due to the design of the logic gates and flip-flops, Timer 1 Gate Value Bit (T1GVAL) must equal 1 for Timer 1 to count. The value of this bit cannot be modified in software. If toggles only when Timer 0 overflows. While writing the code, I took the precautions in order not to read Timer 1 value while T1GVAL is 0.

The code written in PIC C language is below:

```
#include <16F1937.h>

#use delay (clock=16000000)          //This is done to arrange delay operations.


//Addresses of the registers are defined.

#byte OPTION_REG = 0x95              //Timer 0 configurations.

#byte T1CON = 0x18                   //Timer 1 configurations.
```

```
#byte T1GCON = 0x19                 //Timer 1 gate configurations.
#byte CPSCON0 = 0x1A                // capacitive sensing module configurations are done with this
                                    //register.
#byte CPSCON1 = 0x1F                //Capacitive sensing channel is chosen with this register.
#byte TMR0 = 0x15                   // Timer 0 value is stored in this register.
#byte TMR1L = 0x16                  //Lower byte of Timer 1 value is stored in this register.
#byte TMR1H = 0x17                  //Higher byte of Timer 1 value is stored in this register.
#byte INTCON=0x0B                   //Interrupt configurations are done with this register.

unsigned int T1_value;              //This variable keeps the value of Timer1.

//Bits of the registers are defined.
#bit T1GVAL=T1GCON.2
#bit TMR1ON=T1CON.0
#bit TMR0IF=INTCON.2
#bit TMR0IE=INTCON.5

#define LED PIN_B1              //The connections of LEDs are specified.
#define LEDKNT PIN_B2
#define KRMZI PIN_B3

#define TSHLD 0x0001

#INT_TIMER0                     //Interrupt is defined.
void TIMER0_isr()               //Interrupt Service Routine
{
TMR0IE=0;               //Timer0 overflow interrupt is deactivated.
```

```
TMR1ON=0;              //Timer1 stops counting.

if(T1GVAL==1)
{
TMR0=0;
TMR1L=0;
TMR1H=0;
TMR1ON=1;

while(T1GVAL==1);                    //The progam waits until T1GVAL becomes 0.
}

  T1_value=TMR1L+(unsigned int)(TMR1H<<8);          //Timer1 value is assigned to a variable.
  if((T1_value==0))                                 //
  {
output_low(KRMZI);
 delay_ms(10);
  }
  else
  {
  output_high(KRMZI);
  delay_ms(10);
  }
TMR0=0;                              // Before returning from the interrupt, timers are set to 0.
TMR1L=0;                             // Then, counting operation can be done with no mistake.
TMR1H=0;
```

```
TMR1ON=1;
TMR0IE=1;                          // Timer 0 overflow interrupt is activated.
}

void init_cps(){
CPSCON0=140;
CPSCON1=0;
OPTION_REG=197;
T1CON=196;
T1GCON=225;
}

void main(){                                          //program starts from here
setup_oscillator(OSC_16MHZ|OSC_INTRC|OSC_PLL_OFF,0);  //internal oscillator  is defined with
                                                      //16MHz frequency

SET_TRIS_B(0x01);                  //B0 is defined as input. Other pins of B port are output.

setup_adc(ADC_OFF);                //Analog-to-digital conversion cancelled
enable_interrupts(INT_TIMER0);     //Timer0 overflow iinterrupt is enabled

init_cps();                        //necessary configurations for capacitive sensing are
                                   //done in this function.
output_low(LED);          //LED is initially turned off.

TMR0=0;                   //Timers are initialized as 0.
```

```
TMR1L=0;

TMR1H=0;

TMR1ON=1;                    //After making this bit 1, Timer1 starts counting.


enable_interrupts(GLOBAL);   //Enabled interrupts are activated by this instruction.


while(TRUE)                  // The system waits continuously for the interrupts.
{}
}                            //End of main program.
```

## 4. SECOND PROJECT

For my second project, I had to improve the previous one so that it would be used for a 7x4 capacitive keypad system. Each key and column of the keypad is connected to a 22pF capacitor. When a key is pressed, the capacitance is doubled there (since two capacitors are in parallel). Then, capacitive sensing method can be used to find out which key is pressed. I improved an algorithm for this. I obtain the average value of the Timer 1 when a 22pF capacitor is connected. Similarly, another average value is compared with 44pF connection. Then Timer 1 value is compared continuously with these two averages and it can be found out whether the key is pressed. After some trials, I observed that average values are not reliable, i.e. Timer 1 value can be easily affected from the peripheral. Our working environment was not ideal for that circuitry. Then I completed only initial parts of the project and it can be improved in the future. However, I learned assembly language in this project and it was very useful for me because assembly language needs more microcontroller hardware knowledge. It helped me better see the inside of PIC. My source code is written below:

```
list      p=16f1937                    ;  processor is defined.

#include          <p16f1937.inc>          ; processor library

  __CONFIG _CONFIG1, _FOSC_INTOSC & _WDTE_OFF & _PWRTE_OFF & _MCLRE_ON & _CP_OFF &
_CPD_OFF & _BOREN_OFF & _CLKOUTEN_ON & _IESO_OFF & _FCMEN_OFF

  __CONFIG _CONFIG2, _WRT_OFF & _VCAPEN_OFF & _PLLEN_OFF & _STVREN_OFF & _BORV_19 &
_LVP_OFF

org    0

goto    Main

; Interrupt Service Routine
```

```
org    100
movlb  0x00                ;select bank 0.
btfss  intcon,tmr0if       ;check if whether is a timer0 overflow interrupt or not.
goto   end_int             ;if not,return from interrupt
bcf    intcon,gie          ;if it is a timer 0 overflow interrupt, disable all the interrupts
                           ;and execute interrupt
btfsc  t1gcon,t1gval       ;check whether t1gval bit is zero or net.
goto   end_int2                  ;if t1gval=1,clear the timers and return from interrupt
                                 ;if t1gval=0,count the timer1 value.
bcf    t1con,tmr1on        ;stop timer1
movf   TMR1L,w             ;check whether timer1=0.
xorlw  0x00
btfss  status,z
goto   int010
movf   TMR1H,w
xorlw  0x00
btfsc  status,z
goto   end_int2        ;if timer1=0, go to end_int2

int010                 ;if timer1 is different than 0, go on
movf   TMR1H,w         ;check high byte of timer1 with the threshold
sublw  0x0f
btfsc  status,z
goto   end_int3        ;if high bytes are equal, check low bytes.
btfsc  status,c
goto   end_int4        ;if timer1 high byte is greater, turn off the leds
```

```
goto end_int1          ;if smaller, turn on the leds

end_int3               ;check low bytes
movf    TMR1L,w
sublw   0x18           ;if low byte is greater, turn on the leds
btfsc   status,c       ;if smaller, turn on the leds.
goto    end_int1

end_int4               ;turn off the leds
movlw   0x80
andwf   PORTC,1
goto    end_int2

end_int1               ;turn on the leds
movlw   0x7F
iorwf   PORTC,1

end_int2
clrf    TMR0                    ; Clear TMR0 register AND prescaller
CLRF    TMR1L                   ; Clear TMR1 High and Low registers
CLRF    TMR1H                   ;

end_int
bsf   t1con,tmr1on          ;start timer1
bcf             intcon,tmr0if       ;enable timer0 overflow interrupt
bsf             intcon,gie                  ;enable global interrupts
```

```
retfie                          ;return from interrupt
; Main Program
Main
clrf PORTA                      ; Clear all ports
clrf PORTB
clrf PORTC
clrf PORTD
movlb 0x03
clrf ANSELA                     ;disable A/D conversions
clrf ANSELB                     ;for all channels
clrf ANSELD
clrf ANSELE
movlb 0x01                      ; go to bank 1
movlw b'11111111'
movwf TRISA                     ; Set the port pin types of the RA
movwf TRISB                     ; Set the port pin types of the RB
movwf TRISD                     ; Set the port pin types of the RD
clrf TRISC                      ; Set the port pin types of the RC
movlw b'01111000'
movwf OSCCON                    ;oscillator freq=16MHz
movlb 0x00                      ; Go to bank 0
CLRF INTCON                     ; Disable all interrupts
bsf intcon,TMR0IE               ; Enable only the TMR0 overflow interrupt

movlw b'10001100'               ; Setup capacitance module.
movwf CPSCON0                   ;
```

```
movlw b'00000000'          ; Channel 0 (CPS0) will be checked.
movwf CPSCON1

movlw b'11000010'          ;configure timer0.
OPTION                     ;timer0 prescaller rate is 1:64

movlw b'11000100'          ; timer1 source is Capacitive Sensing Oscillator
movwf T1CON                ;Prescaller is 1:8 and timer1 is stopped

movlw b'11100001'          ; Setup Timer 1 Gate module
movwf T1GCON               ; Bits <1:0>: Gate source is Timer0 overflow

CLRF TMR1L                 ; Clear timer1
CLRF TMR1H
CLRF TMR0                  ; Clear timer0
BSF T1CON,TMR1ON           ; start timer1
bsf intcon,gie             ; enable global interrupts

Mainloop    goto MainLoop  ;wait for interrupt
end         ; end of program
```

## 5. THIRD PROJECT

I was charged with designing a brake box to be used for brake tests. The regulations and expectations are:

- The box must drive a motor to loose or tighten a brake.

- There is a double-pole double-throw (DPDT) switch on the box. When switch is neutral (i.e. not pressed), motor does not rotate. It starts rotating when switch is pressed, forward or backward when other two sides are selected.
- There is a limit switch outside the box. This switch is added to the system in order to protect the brake system. If the brake system is tightened or loosened extremely, the switch is pressed mechanically. The motor must stop rotating as long as the switch is activated. There are 2 LEDs on the box, one of them (RED) will turn on when limit switch is pressed. The other one (GREEN) will be on as long as the power is supplied to the box.
- There are no limitations about the size of the box.
- The box is supplied with 28V DC voltage.
- The motor must draw approximately 1000mA.

For this assignment, I started designing a box. First of all, a motor driver circuitry is needed and it must be fed with a Pulse-Width-Modulation (PWM) system. Since an intelligent system is not advised by the engineers, I decided to use a 555 timer circuit. The details are explained below:

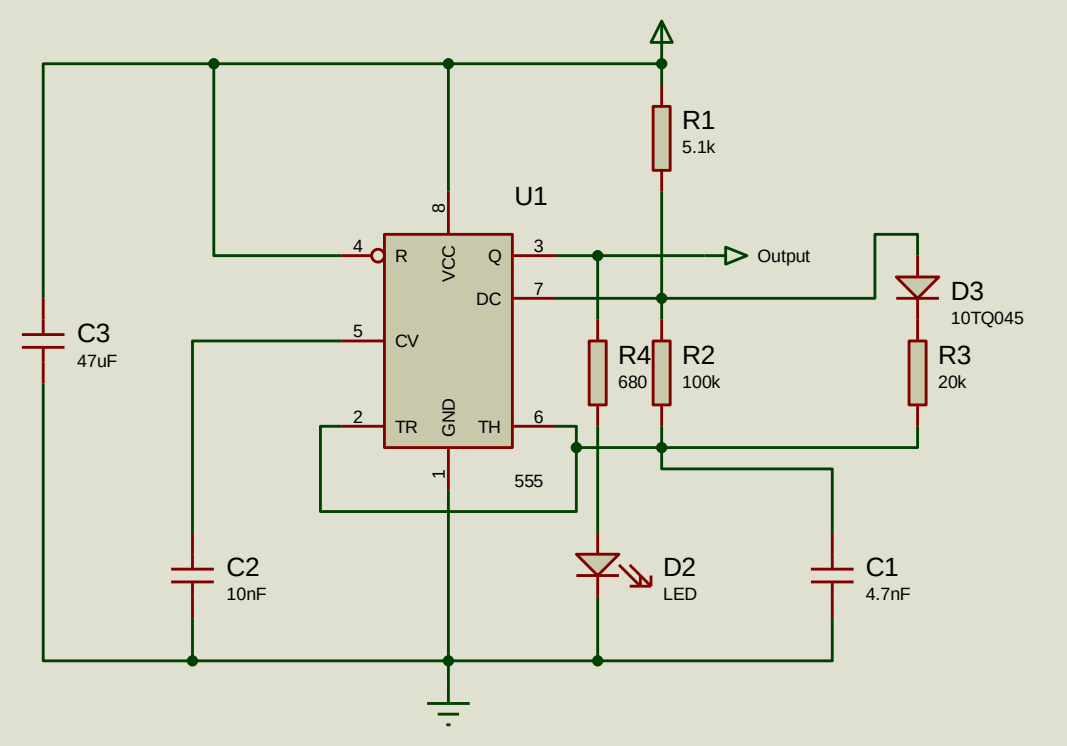


**Figure 2: 555 Timer Circuit**

**555 Timer:** According to the calculations, a square wave with 20% duty cycle and 3 KHz frequency is necessary. To this end, a circuit as in Figure 2 was constructed. R4 and LED are added to observe the output. Firstly, I used only $R_1$ and $R_2$ for duty cycle adjustment. The duty cycle formula is

Duty Cycle=100*($R_1$+$R_2$)/ ($R_1$+2$R_2$)

which is always higher than 50%. Then, I added diode ($D_3$) and $R_3$ to reduce the duty cycle and the new formula is

Duty Cycle=100*($R_1$+$R_2$//$R_3$)/ ($R_1$+$R_2$+$R_2$//$R_3$)

Finally, $R_1$, $R_2$, $R_3$ and $C_1$ are used to arrange the frequency. The frequency formula is

Frequency=1/[0.69*$C_1$*($R_1$+$R_2$+$R_2$//$R_3$)]

After doing some calculations, the resistance and capacitance values are chosen as $R_1$=5.1 Kohms, $R_2$=100 Kohms, $R_3$=20 Kohms and $C_1$=4.7 nF. The duty cycle is calculated as 18% and frequency is 2.5 KHz. After that, simulations in ISIS program are performed.

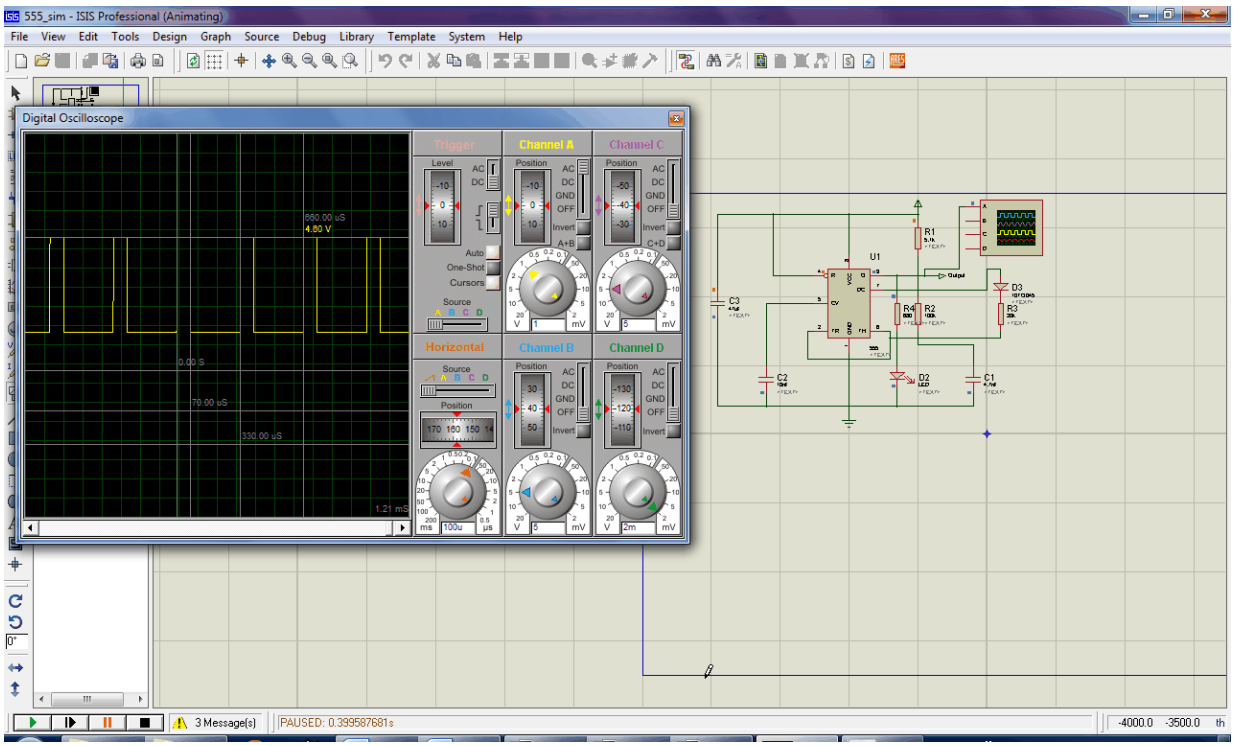


**Figure 3: 555 Timer Circuit Output Waveform**

As can be seen from Figure 4, I obtained a 4.6$V_{PP}$ square wave in simulation. The frequency and duty cycle are calculated in the following:

Frequency=1/(330us)=3.03KHz

Duty Cycle=100*(70us)/330us)=21.2% which are very close to the expectations.

After that, circuitry was constructed on the breadboard and then holey copper board (also known as pertinax). The values were as expected. In the following steps, $R_4$ and $D_2$ LED are

emitted for simplicity of the circuit. After that, a motor driver is integrated to 555 timer circuit. The details of motor driver circuit are explained below:

**Motor Driver:** The timer cannot supply the voltage and current necessary to drive the motor so we need to use motor driver. LMD18200 motor driver is chosen since it can operate in our voltage and current values. Since this model is not available in ISIS, I was not able to make simulations. The output of 555 timer is connected to PWM input of LMD18200.

Brake pin is used to stop the motor while DPDT switch is not pressed. In the datasheet, it is said that PWM must be 1 to enable brake pin. Although brake function worked properly without this modification, I added a pull-up resistor to PWM input of the motor driver for any case.

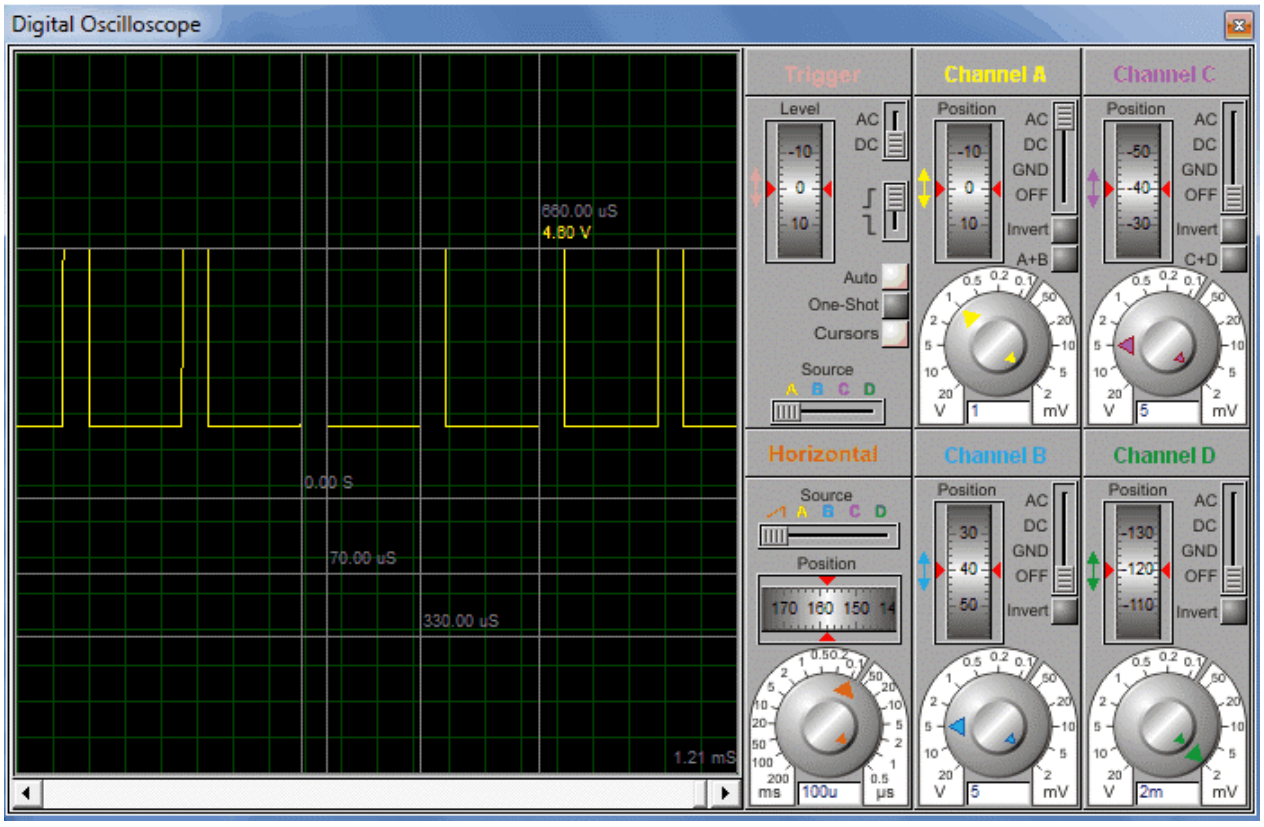


**Figure 4: Simulation Results**

Direction pin is used to decide if we want to loosen or tighten the brake. The pin is connected to one of the outputs of the switch. When the switch is pressed for forward, direction pin will be high, otherwise low.

Current sense function of the motor driver is not enabled since the motor draws relatively small current.

**Voltage Regulator:** A voltage regulator is used to convert 28V DC to 5 V DC. It is necessary that 555 timer and pull-up resistors be supplied with 5V. LM78L05 voltage regulator was appropriate for our circuit since 555 timer and pull-up resistors do not draw more than 100 mA. The capacitors are connected to the input and output to prevent instant voltage changes. The circuit can be seen from Figure 5.

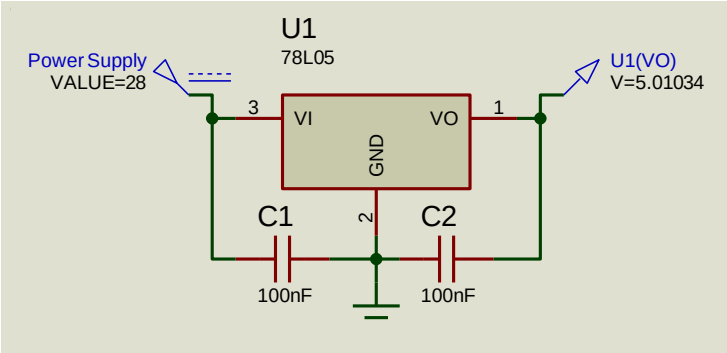


**Figure 5: Voltage Regulator Circuit**

**LEDs:** The red LED must stay on as long as the limit switch is pressed, then a resistor-transistor circuit was constructed as can be seen in Figure 7. In this circuit, npn type low-power bipolar junction transistor is used. If limit switch is not pressed, BE junction of the transistor is ON and current flows through red LED. If pressed, BE junction is OFF since $V_{BE}$=0 and no current flows through LED.

The green LED is placed on the box to show that power is supplied to the box. For this purpose, I placed it to the output of 78L05. A 2.7-Kohm-resistor is connected in series with the LED. As long as 28V is supplied to the box, voltage regulation will be performed and green LED will stay ON.

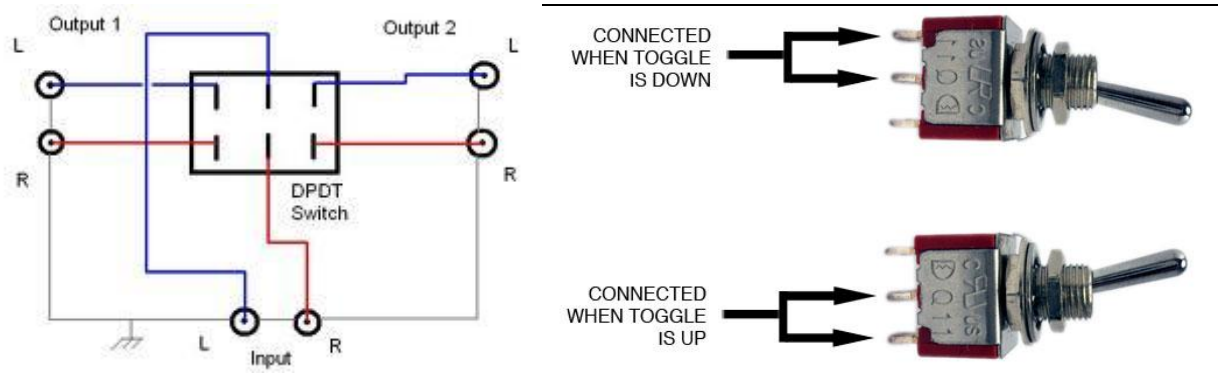


**Figure 6: Double-Pole Double -Throw Switch**

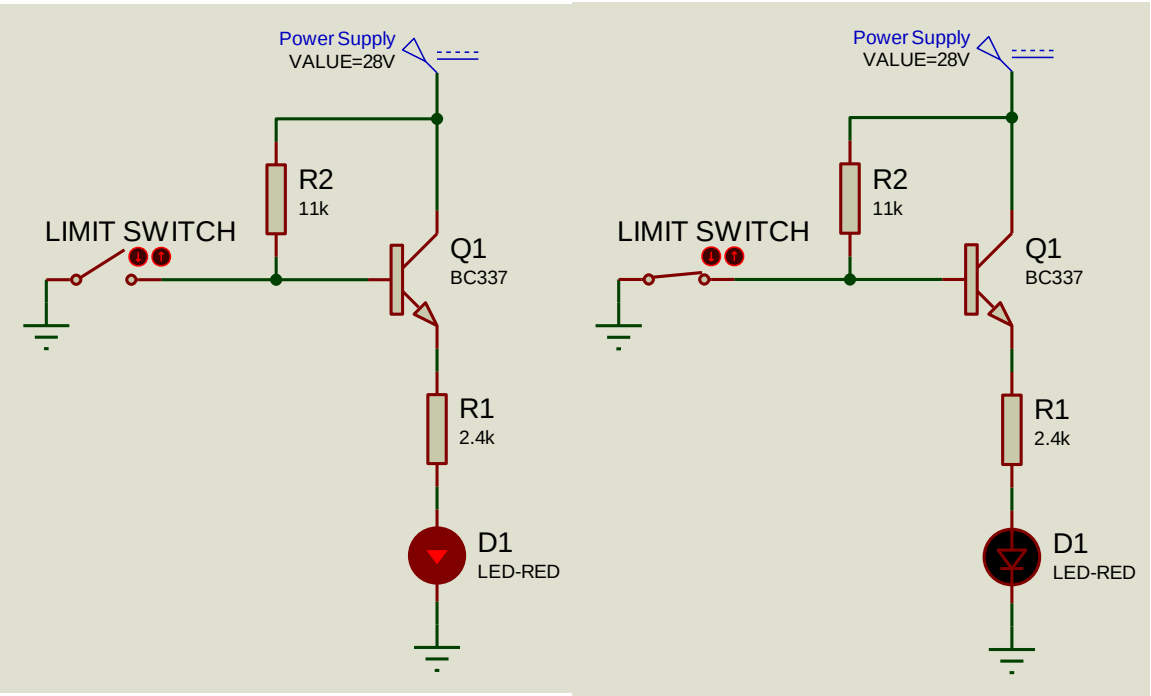


**Figure 7: Limit Switch Activates and Deactivates LED**

**Switch:** The switch on the box must affect the actions of the motor. The motor stops, moves in clockwise or counterclockwise directions according to the status of the switch. The type is double-pole double-throw switch. As can be seen in the Figure 6, the switch has 2 independent outputs. None of the three pins in the upper half is connected to the ones in the lower half whether it is pressed or not. When switch is pressed, the middle pin is internally connected to the one in the left or the right depending on which side is pressed. On the lower half, the pin on the left is connected to the supply ground. The middle in in lower half is connected to the Direction input. When switch is pressed to that side, Direction pin will become low and motor will rotate in reverse direction. If the switch is pressed to opposite side, the direction pin will get no signal from the switch. Due to the pull-up resistor, Direction pin will be high and motor will rotate in usual direction.

**Limit Switch:** This switch is placed to protect the brake from being broken due to being tightened more than necessary. If the motor rotates excessively in forward direction, there exists a possibility of breaking the brake. This limit switch is automatically pressed in the case that the brake is tightened more than the critical rate. According to my design, the switch behaves as a fuse. When switch is pressed, 555 timer and motor driver are disconnected from the power supply. Therefore, no output to drive the motor is generated. I did this by connecting to the negative ends of 555 and motor driver to the limit switch. While the limit switch is not pressed, it is ground. When it is pressed, it becomes floating. Similarly, when limit switch is pressed, 555 timer and motor driver do not work and motor makes no rotations.

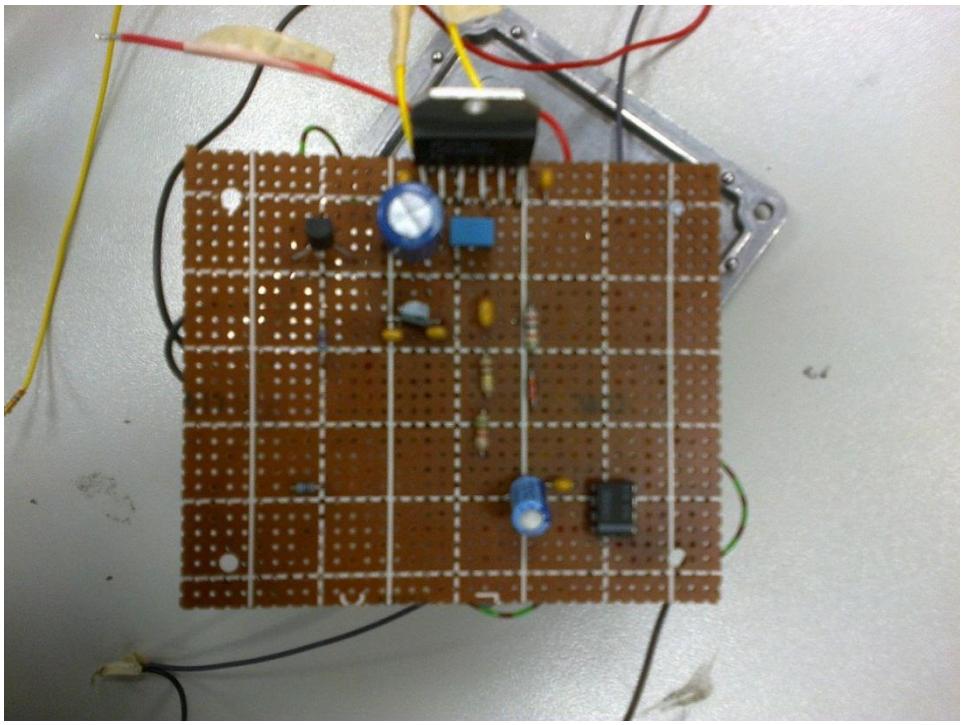

Figure 8: The circuit and components

**PCB:** The last thing to do was to construct the circuit in Altium Designer and draw a Printed Circuit Board (PCB) for this design. Firstly, I must give information about this program. Altium is a multi-functional electronics program. The user can design embedded systems, make its simulations and draw PCBs. I was not able to make simulations in this program because some components were not defined in library for simulation. To be clearer, the electronic materials are defined in libraries so that user sees how many I/O pins it has, how it responds to the signals, what is the size of the component etc. It was not applicable to make simulations for some of the components I used. While designing the schematic, components are chosen from the library (each company has its own library for its products) and replaced to the field. I chose all of them through-hole, not surface-mounted.

I prepared a schematic of the circuit and then converted it to a PCB. During this, I applied to the Altium PCB Design Tutorial many times. The schematic can be seen in Figure 9. The red crosses in the figure indicate the pins which are not connected to anywhere. It is necessary to put them in the schematic because Altium compiles the schematics and unconnected pins cause errors. This is done to avoid any connection mistakes. The red circles indicate the design rules. The rules shape the PCB design. It can be set for various reasons. I put them in the schematic in order to make supply voltage routes thicker than other routes. Routes carrying 28V and 0V have

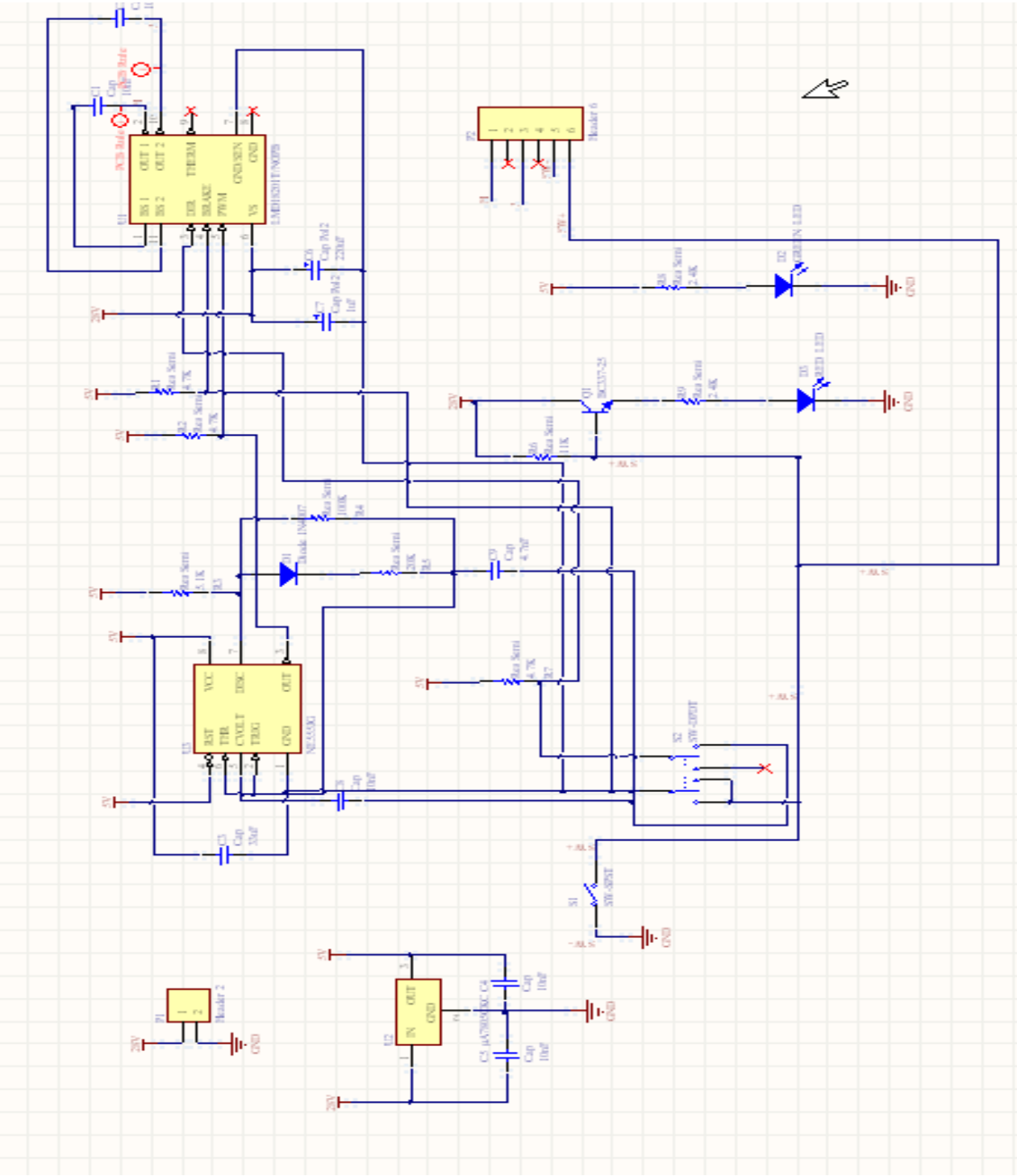

Figure 9: Schematic in Altium Designer

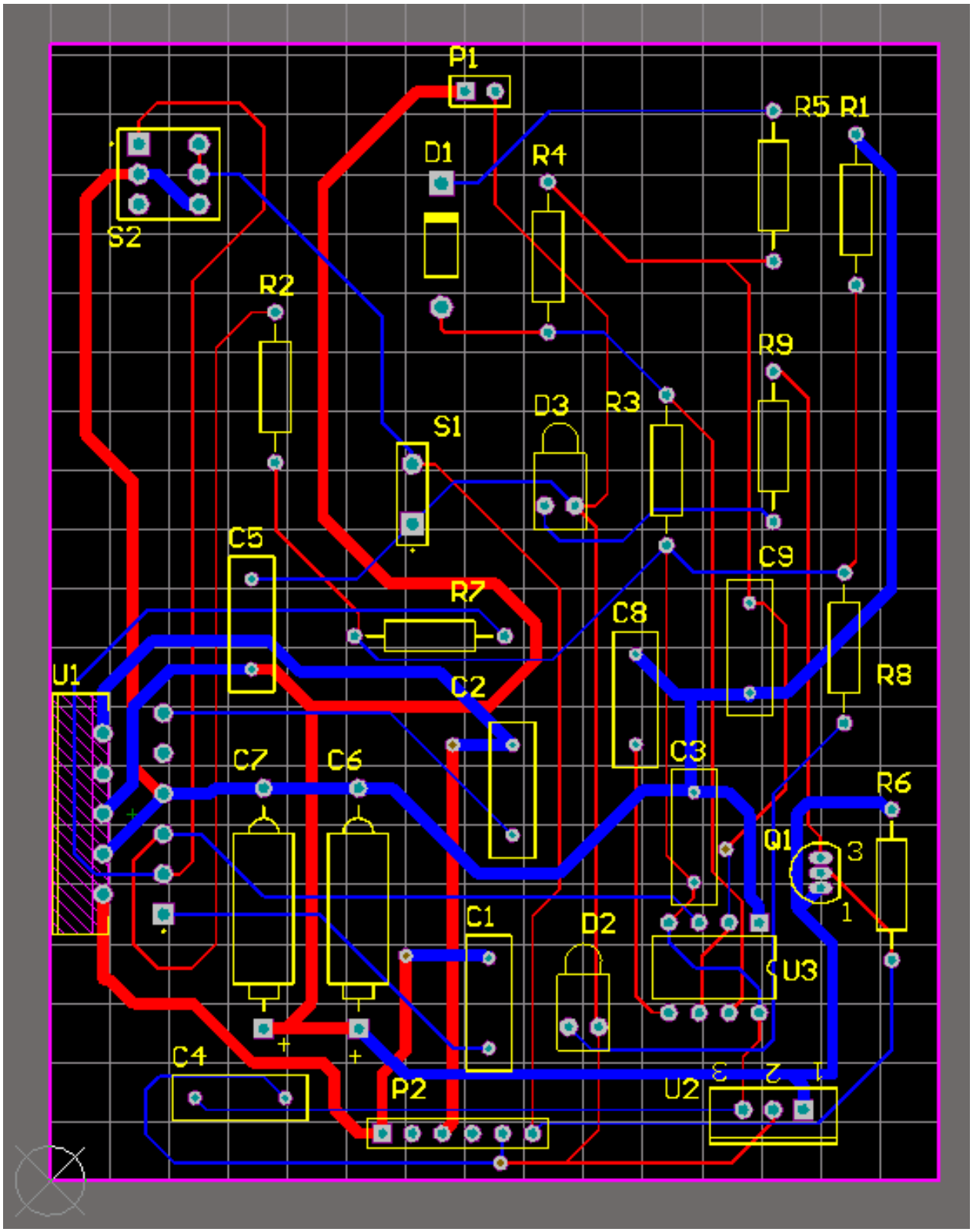


Figure 10: PCB Drawing in Altium Designer

1mm thickness while others have 0.254mm. After compiling and removing the errors in the schematic, I started drawing the PCB.

Initially, I defined a PCB size. It is 75mm*96mm which is appropriate for the box. Then all the components were carried automatically onto the PCB. I chose a 2-layered PCB design, i.e. the routes exist on both sides of the electronic card. The components are placed only to the top layer, though. I selected “Auto-routing” option and most of the routes were drawn. Then, I manually drew the remaining nodes and edited the routing errors. My prior rules were not to keep lines away from each other and to not make contact with wrong pins. After that, I tried to keep the routes with high voltage differences away from each other. This is necessary to prevent voltage jumps between two routes. One of my concerns is to prevent silkscreens from vanishing. The silkscreens are the shapes and letters on the PCB that describe the component. They must not cross with top-layer routes (since components are placed at the top, the silk screens are also printed to the top layer). Following all the routes, I have drawn the PCB in Figure 10. In this figure, red routes are at the top layer, the blues at the bottom layer. Some of the routes have both of these colors. These routes are connected to the both side with a conducting hole(called via). When this method is used, a circle is used to show this transition.

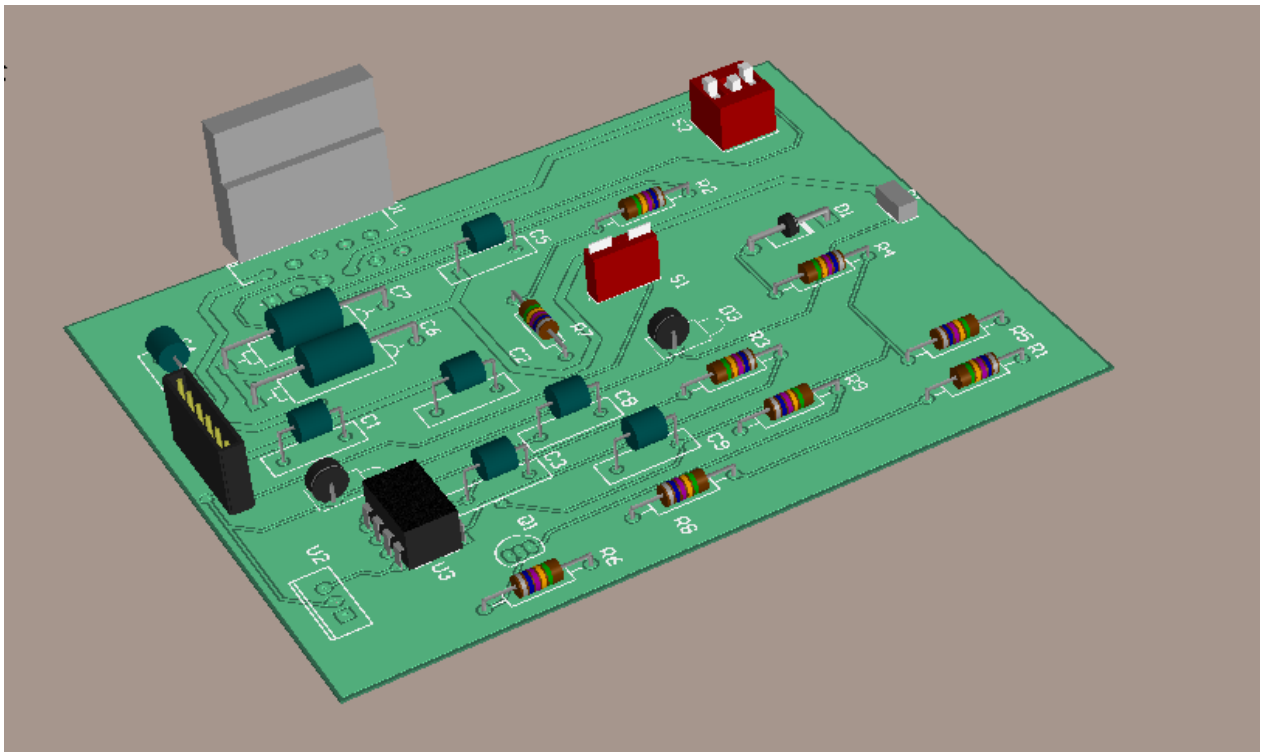

Figure 11: Top Surface of the Circuit in Altium Designer

Finally, I surrounded the whole circuit with grounds. The empty spaces are filled with conductor connected to the ground (of the power supply). This is done to minimize the ground

impedance and solve the EMC (Electromagnetic Compatibility) problems. The 3D view of the PCB with components placed on can be seen in Figure 11. Figure 12 shows the bottom layer of the PCB where only the bottom routes and pins of the components can be seen.

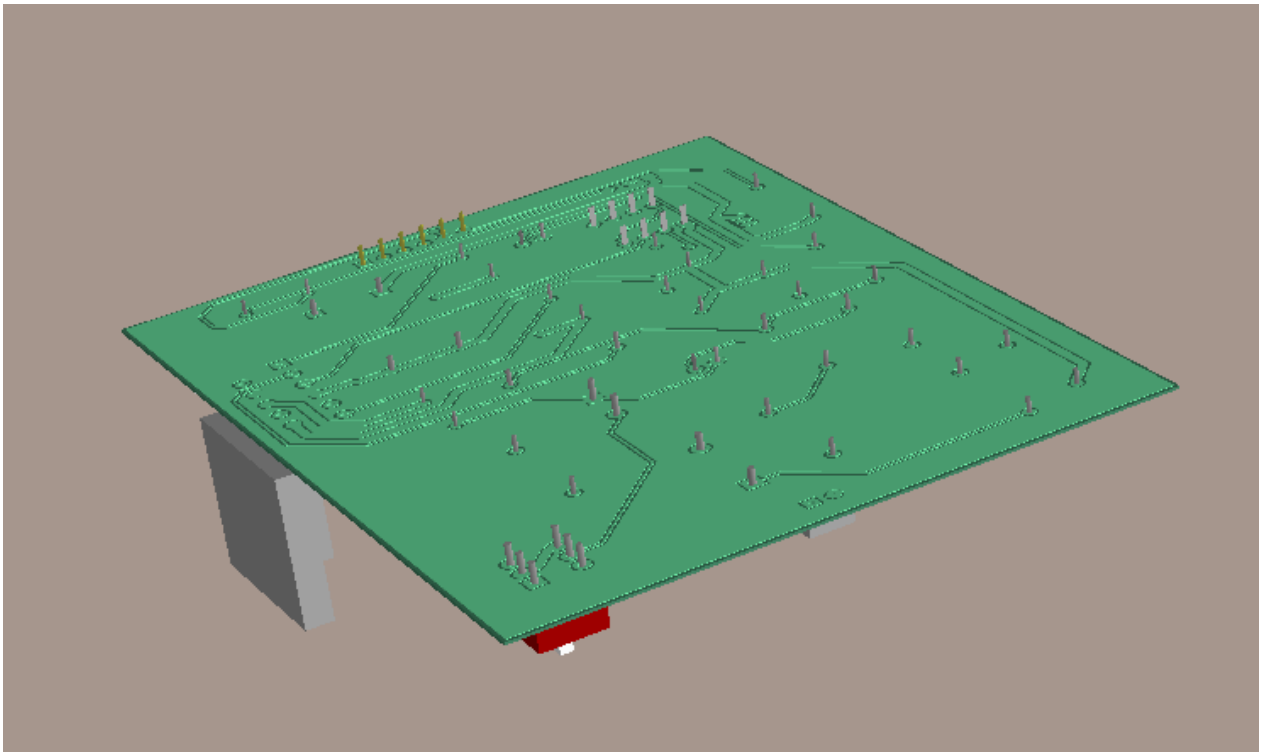

Figure 12: Bottom Surface of the Circuit in Altium Designer

## 6. FOURTH PROJECT

At my last week in TAI, I was given a new job: measuring the status of 2 pedals and sending them to a computer for tests which can be seen in Figure 13. For this job, I used the potentiometer connected to the pedal. Two ends of the potentiometer are connected to 5 V DC supply. The analog voltage is measured from the middle pin of the potentiometer. PIC 16F1937 microprocessors are chosen for analog-to-digital-conversion. The results to be obtained from the conversions will be continuously sent via UART (Universal Asynchronous Receiver/Transmitter). RS232 serial communication protocol is used for sending data.

I started the job with analog-to-digital-conversion. Two analog voltages coming from potentiometers are converted to 8-bit numbers. Then, they are converted to 14-bit numbers since this format is necessary in some other programs. After that, the numbers must be sent via UART. However, UART can send at most 8-bit data at a time. Therefore, conversion results are sent in 2 parts. There are start and stop bytes in the data format sent to the computer. This is done to ensure the reliability of the sent data. Moreover, sum check method is used to confirm the correctness of the sent data. Four data byte are added to each other and the result is sent

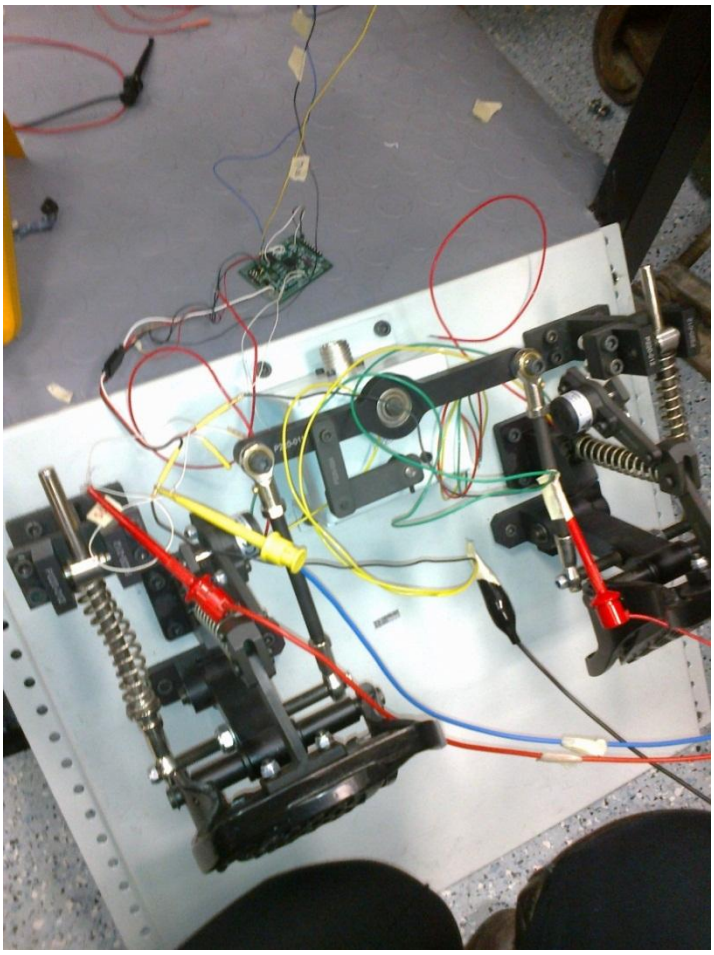

Figure 13: Pedals

to the computer as shown below:

Check Sum= (Pedal-1 Low byte) + (Pedal-1 High byte) + (Pedal-2 Low byte) + (Pedal-2 High byte)

The sum check is 8 bit, so there is a possibility of overflowing while summing the bytes. In such a case, carry is ignored. Then, at one time, 8 bytes are sent via UART to the computer. These bytes are 2 start bytes (AA and 55 in hexadecimal), Pedal-1 high byte, Pedal-1 low byte, Pedal-2 high byte, Pedal-2 low byte, sum check byte and stop byte (FF in hexadecimal), respectively. The bytes can be seen below:

| Start Byte (AA) | Start Byte (55) | Pedal-1 High Byte | Pedal-1 Low Byte | Pedal-2 High Byte | Pedal-2 Low Byte | Sum Check Byte | Stop Byte (FF) |
|---|---|---|---|---|---|---|---|

The signals coming from UART are in TTL level, i.e. they are in 0-5V range. However, computers cannot read data in these levels. It uses +12V, -12V as low and high, respectively. Then, I need to convert the signals. I used a MAX233 driver chip for this conversion. Generally, MAX232 driver is preferred, however this chip has an advantage over MAX232. There is no need to use external capacitors with our model. The connections of the driver are indicated in Figure 14 which is constructed in ISIS.

After writing the code and compiling, I used a potentiometer (not the one on the pedal) and observed the bytes on Realterm terminal program. The potentiometer resistance was very sensitive and the values seen on the screen were fluctuating even though there was no change in resistance of the potentiometer. Then I decided to make sampling of ADC (Analog-to-Digital-Conversion) results. Every time, 8 A/D conversion results are summed and the average is calculated. This operation is performed for both channels. Since the ADC results need to be sent in 14 bit format, the average values are multiplied by 64. After that, I connected my circuit to the pedal and tried to observe the outputs. I noticed that potentiometer is not operating as I thought it would. At the minimum level, it has 1 V and at the maximum level it has 2.9 V. For the project, it is important to know that if the pedal is fully pressed or not pressed. Then these

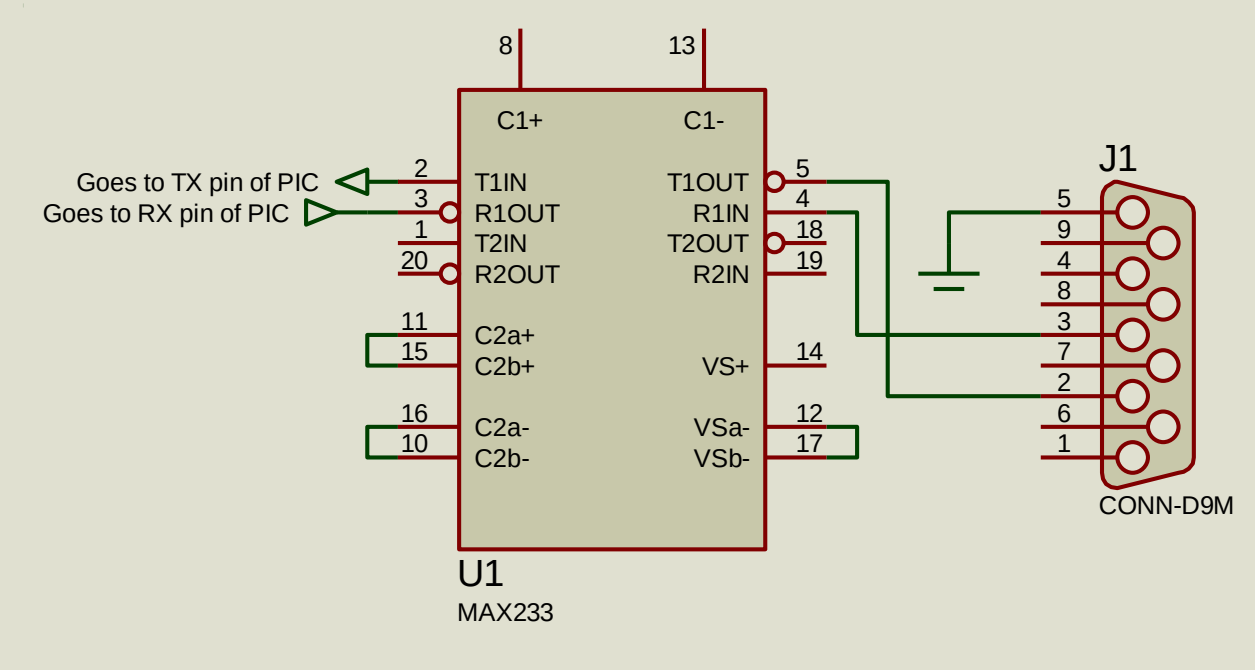


**Figure 14: MAX233 Driver Connection**

voltages are taken as reference voltages. If less than 0.5 V is read from a channel, it is taken to be 0. Similarly, if more than 2.9 V is read, it is taken to be 5V (which equals 3FFF as 16-bit hexadecimal number). If the ADC result is between these numbers, it is sent directly to the computer. The calculations are shown below:

[(0.5V)/(5V)]*255*64=1632

[(2.9V)/(5V)]*255*64=9466

The code is written in C language and shown below:

```
#include <16F1937.h>
#device adc=8
#FUSES NOWDT                 //No Watch Dog Timer
#FUSES INTRC_IO              //Internal RC Osc, no CLKOUT
#FUSES NOPUT                 //No Power Up Timer
#FUSES NOPROTECT             //Code not protected from reading
#FUSES MCLR                  //Master Clear pin enabled
#FUSES NOCPD                 //No EE protection
#FUSES NOBROWNOUT            //No brownout reset
#FUSES IESO                  //Internal External Switch Over mode enabled
#FUSES FCMEN                 //Fail-safe clock monitor enabled
#FUSES WDT_SW
#FUSES CLKOUT                //Output clock on OSC2
#FUSES NOWRT                 //Program memory not write protected
#FUSES NOVCAP
#FUSES PLL
#FUSES STVREN                //Stack full/underflow will cause reset
#FUSES BORV19
#FUSES NOLVP                 //No low voltage programming, B3(PIC16) or B5(PIC18) used for I/O
#FUSES NODEBUG               //No Debug mode for ICD
```

```
#use delay( int=16000000)     //Needed for delay operations.
//rs232 configurations: Baud rate=38400bps. No parity bit. 8 bit data is sent.
#use rs232(baud=38400,parity=N, xmit=PIN_C6,rcv=PIN_C7,bits=8)


void main() {
unsigned long value_1,value_2, average_1,average_2;
unsigned char check_sum=0;
unsigned long MSB_1_temp, MSB_2_temp;
int j;
unsigned char MSB_1,LSB_1,MSB_2,LSB_2;
  setup_adc_ports(sAn6 | sAn5 | VSS_VDD);           //AN5 and AN6 channels are used for ADC
  setup_adc(ADC_CLOCK_DIV_4);     // sampling time 1us
  setup_spi(SPI_SS_DISABLED);
  setup_lcd(LCD_DISABLED);
  setup_timer_0(RTCC_INTERNAL|RTCC_DIV_1);
  setup_timer_1(T1_DISABLED);
  setup_timer_2(T2_DISABLED,0,1);
  setup_ccp1(CCP_OFF);
  setup_comparator(NC_NC_NC_NC);
  setup_oscillator(OSC_16MHZ|OSC_TIMER1|OSC_PLL_OFF,0); //internal oscillator with
                                                        // 16MHz is chosen
while(1) {
average_1=0;
average_2=0;
for (j=0;j<8;j++){                  //8 successive ADC results are sampled.
```

```
set_adc_channel(5);          //Enable channel 5 for ADC.
delay_ms(10);                //Some delay is necessary for ADC.
value_1=read_adc();          //ADC result of 1st pedal is stored in this variable.

set_adc_channel(6);          //Enable channel 5 for ADC.
delay_ms(10);
value_2=read_adc();          //ADC result of 2nd  pedal is stored in this variable.

(average_1)=(average_1)+value_1;          //8 ADC results are summed
(average_2)=(average_2)+value_2;
}
(average_1)=(average_1)*8;                //Average of them is calculated and then average is
(average_2)=(average_2)*8;                //multiplied by 64 to get an 14-bit data.

 if(average_1<1632)                  //if voltage<1V, take it as 0V.
 average_1=0;

 if(average_2<1632)
 average_2=0;

 if(average_1>9466)          //if voltage>2.9V, take it as 5V.
 average_1=0x3FFF;

 if(average_2>9466)
```

```
 average_2=0x3FFF;


MSB_1_temp=average_1>>8;
MSB_1=(unsigned char)(0x00FF) & MSB_1_temp;       //high byte of average 1st pedal ADC result
MSB_2_temp=average_2>>8;
MSB_2=(unsigned char)(0x00FF) & MSB_2_temp;    //high byte of average 2nd pedal ADC result

LSB_1=(unsigned char)(0x00FF) & average_1;          //low byte of average 1st pedal ADC result
LSB_2=(unsigned char)(0x00FF) & average_2;          //low byte of average 2nd pedal ADC result

check_sum=MSB_1+MSB_2+LSB_1+LSB_2;
putc(0xAA);
putc(0x55);
putc(MSB_1);
putc(LSB_1);
putc(MSB_2);
putc(LSB_2);
putc(check_sum);
putc(0xFF);
check_sum=0;
}         }
```

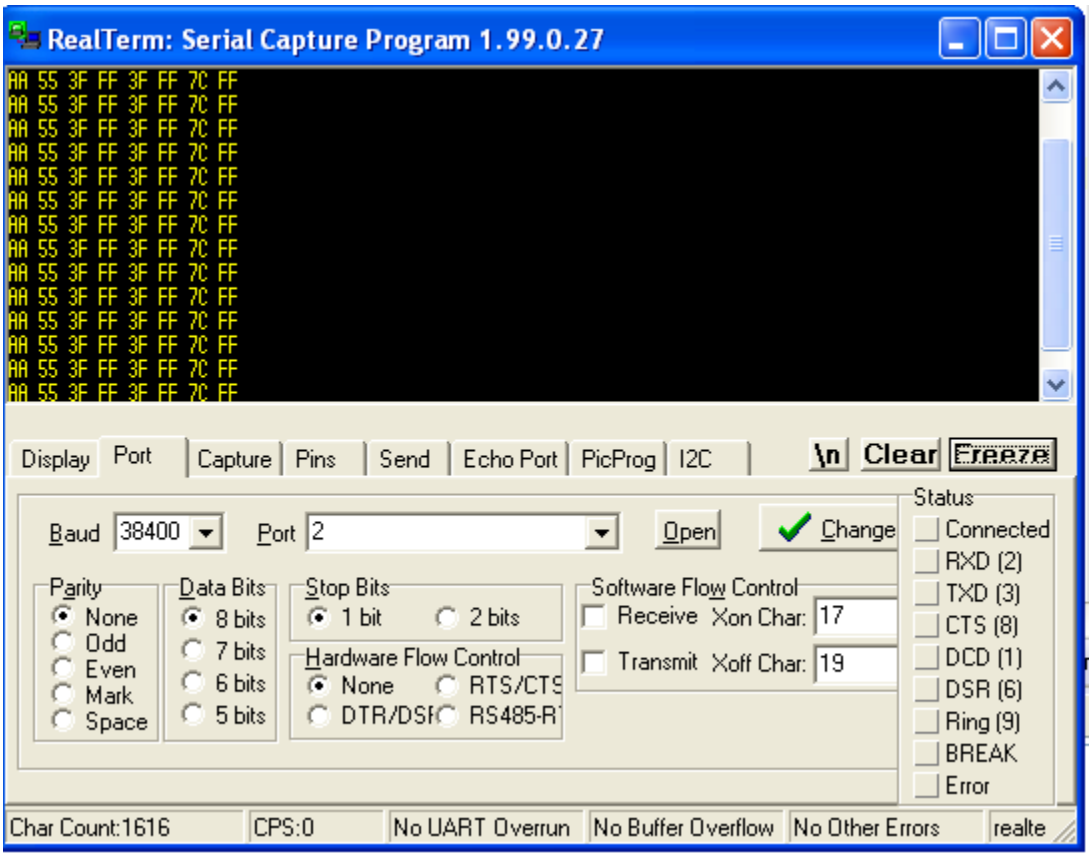


Figure 15: Data Sent to the Computer

## 7. CONCLUSION

I completed my internship at TAI in Ankara. I believe that it has been a good experience for me. It has several reasons. Firstly, I improved my technical knowledge of electronics. I gained many experiences in embedded circuit design. I worked with different electronics programs such as MPLAB, Altium. Since TAI is following the latest technologies, I could benefit from this, too. Secondly, TAI was a well-organized and professional company. I was able to observe how design, implementation and test steps of a project are achieved in such a company. Moreover, during this internship I had chance to see how engineers achieve division of labor, team work in a laboratory and how big systems are divided into smaller systems and accomplished.

I performed my previous SP in a different department of TAI, Engineering Reinforcement (Mühendislik Destek). The projects I was charged with were not directly utilized by the company. However, in this internship, I was charged with the designs to be used in the tests or the products. For this reason, I needed to think how to make the system simpler so that end user can easily work with it. The brake protection circuit is designed to be used in the tests. The pedal data transmission design is also constructed to test whether the pedals are functioning properly. Additionally, during this SP, I had chance to do implementation rather than design and

simulate it in PC. This helped me improve methods for solving problems encountered during the implementation process such as finding the defected component or poorly-made soldering connections.

Finally, I recommend my SP location to other students. I had many work experiences in technical and nontechnical manner. I believe that all of these will help me in my work life. I used the facilities of the company and had spent my time efficiently during the internship to be useful for TAI.

## 8. REFERENCES


[1]PIC 16F1937 Microcontroller Datasheet
http://ww1.microchip.com/downloads/en/DeviceDoc/41364E.pdf

[2]CA555E Timer Datasheet

http://www.datasheetcatalog.org/datasheets/90/53593_DS.pdf

[3]LMD18200 Motor Driver Datasheet

http://www.ti.com/lit/ds/symlink/lmd18200.pdf

[4]LM78L05 Voltage Regulator Datasheet

http://www.ti.com/lit/ds/symlink/lm78l05.pdf

[5]BC337 Npn Transistor Datasheet

http://www.fairchildsemi.com/ds/BC/BC337.pdf

[6]MAX233 Driver Datasheet

http://datasheets.maxim-ic.com/en/ds/MAX220-MAX249.pdf

[7]Microchip Application Notes AN1101 Introduction to Capacitive Sensing

http://ww1.microchip.com/downloads/en/AppNotes/01101a.pdf

[8]Microchip Application Notes AN1103Software Handling for Capacitive Sensing

http://ww1.microchip.com/downloads/en/AppNotes/01103a.pdf

[9]Microchip Application Notes AN1171 How to Use the Capacitive Sensing Module

http://ww1.microchip.com/downloads/en/AppNotes/01171C.pdf

[10]Altium Designer PCB Design Tutorial

http://wiki.altium.com/display/ADOH/Tutorial+-+Getting+Started+with+PCB+Design