EEC 195 Final Report

Team Chucksgon

Instructor: Lance Halsted

Jiang, Zheng Zhang

Lai, Cheuk Chung

Lin, Lejin

Lu, Zixin (Yin Yin)

Fall - Winter Quarter 2014 - 2015

Executive Summary

This report presents the design of an autonomous race car that embeds the FRDM-KL25Z microcontroller for two NATCAR competitions. The aim of this project was to design and construct an autonomous race car that can detect both edge lanes in a plastic track and be able to complete the racecourse in the shortest possible time. This design is able to handle different road conditions, including turns, straights, chicanes, bumps and hills. Because the two NATCAR competitions adopted different racing systems, this report includes the discussion of TFC shield control for Competition I and the design of custom motor control board for Competition II. In addition to presenting significant hardware and software designs in the project, this report summarizes the operating performance and the safety design that estimates and handles unforeseen circumstances.

Below is a summary of each team member's contribution to this project:

	Task	% effort	Signature
Jiang, Zheng Zhang	Camera Stand/ Chassis Design	25	
Lai, Cheuk Chung	Programming - Lab	25	
Lin, Lejin	PCB Board/ Chassis Design	25	
Lu, Zixin	Programming - Servo/Motor	25	

Detailed Technical Reports: Coding

Motor and Servo Controls

This part is written by Lu,Zixin

Part 1: Motor Control

By analyzing the current feedback, I can change the power delivered to the motor based on the track course. Current feedback is hard to use because it is not directly related to the speed of the car, the same current feedback can be reflecting to different types of track. I intended to implement speed boost on uphill and slow down on downhill, but unfortunately I never got the reliable value of current feedback when doing downhill. Therefore, the car easily run off track where there is a turn after the hill.

I believe motor control is the reason behind smooth turns. By implementing the motor properly with servo, the car can perform better than using servo only. When our car turns, it sets one of the motor's PW to zero. It is crucial to our car's turning mechanism because the PW can affect the speed. For further implementation, there will be a 2 turning cases. The first case is when the car is slightly close to the edge of track, in this situation the car will not change the motor's PW. Another case is when the car gets to the edge and the error value is huge, then one motor's PW will be set to zero while the other one's PW will be decreased as well.

Part 2: Servo Control

I implemented a simple servo control algorithm for efficiency. Our car only has three conditions that turns the servo: left, right, and straight. I implemented Proportional for our servo controls.

Instead of implementing PW1 = (Error*Pg), as demonstrated in class, we used the expression (Error * Pg)^N. The Proportional has minor effects on the straight line performance especially in chicanes. I set the exponential power N to be 4 or 5 to make rapid change in shape turns while keeping the car to go as straight as possible in the straight track. This is probably the reason why our car is able to run straight in chicanes.

Often time the resulting PW is a lot greater than what the servo can accomplish. Therefore, I made an important implementation to protect our fragile servo: In order to avoid breaking our servo, I used the following expressions to limit the PW:

```
PW3 = 4500 - (error2 * pg)^4;
if(PW3 <= 3600){ PW3 = 3600;}
```

The purpose of these expressions is simply stop the servo from overturning the wheels and hence preventing the wheels to hit the car's chassis. I have discovered the overturning issue before Competition and I have found this issue caused large damage to car's structure. In addition to the damage, the overturning issue caused the servo unable to turn the wheels back to straight position unless it is adjusted manually. After testing many possible solutions, I have found these expressions modify the PW accordingly.

For intersection handling, our car generally has three conditions that require to turn the servo: left, right and straight. Because in each angle of the intersection, the 90-degree black track edge is easily to mislead the car and the car would make turn before going into the intersection. I designed a straight algorithm in the else statement so that our car can run straight most of the

time. Because of this important change in the algorithm, our car can easily go across the intersection without any issue since the Checkpoint 2.

Detailed Technical Reports: Coding

This part is written by Lai,Cheuk Chung

Camera Analyzing

To increase efficiency and uniformity, I decided to use same set of code for analyzing data from both cameras. This decision also makes me assume both side of camera behaves similarly. Since I only apply proportional servo adjustment, Errotolo1 and 2 , the error thresholds that determines the track's black and white, are always the same value because of the assumption I made above, As I proceed through the spring quarter, I found out that the two cameras behave quite differently and have to declare some variables that are dependent on only one camera. Our right camera is significantly weaker than the left one because it is simply less sensitive with the same parameters set. With this acknowledged, I decided to implement more to compensate the shortcoming.

One of the implementation is black counter. In order to ensure the car is able to make a sharp turn and run within the racecourse, we have implemented black counter to control the wheels by conditions. The cameras read the data from the track then specifically keep the last error value in record. This value is significant because when the car is about to run off track, the system is able to obtain the last error value and use this value into the P method to make that turn. The right camera always read too many black so the servo is not turning as desired.

There is 128 values in the array `valPing[i]/valPong[i]`

```
if(valPing[i] < 0x30){
                              valPingBinaryVT[i] = 0;
                              blackCounter1++;
```

Black counter increments one every time it receives a valPing[i] lower than the threshold (0x30)

The threshold is another factor that should be set uniquely for each camera.

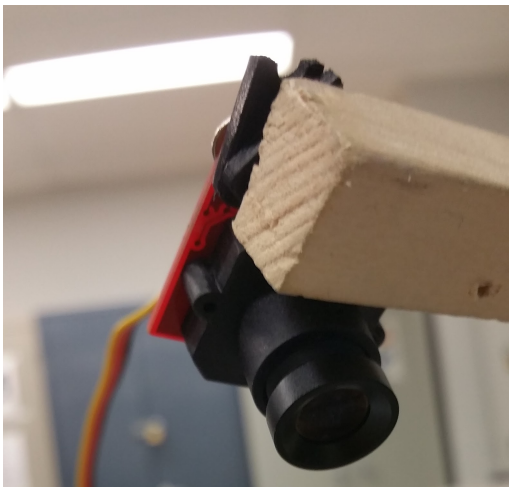
```
if(blackCounter1 >= 110){
                    blackCounter1 = 0;
                    error1 = error1;   //all black then keep last error}
```

The last error is kept so that there is a good error that keeps the servo turning consistent. The last error is the last known error before the car goes out of the track.

The threshold is gathered by trial and error. Thankfully the light in 2147 is very similar to the light in track room. I only need to test a few times to find out the appropriate threshold for the room.

Camera Focus

In a time between competition 1 and 2, the focus of the camera determines whether the car makes through the track or not. When I twist the lens outward, the camera sees more of the edge,



thus increasing the sensitivity of the camera. This is another way to improve the weak right camera. The line is actually blurred by the focus, this might make the intersection more puzzling in the view of camera.

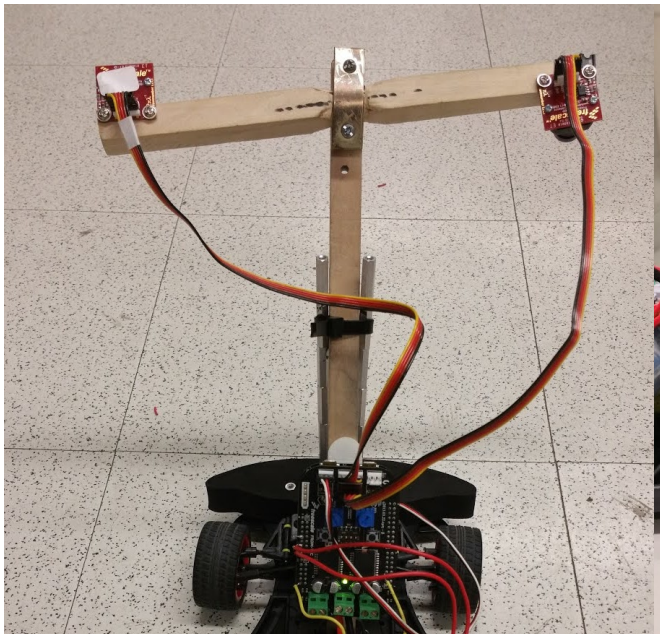
Detailed Technical Report: Hardware

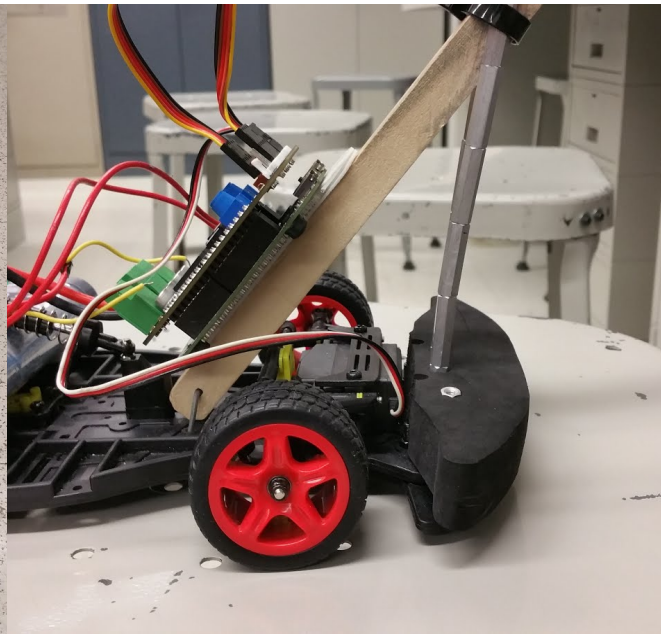
This part is written by Jiang, Zheng Zhang

Cameras and Camera Stand

The upside down camera is one of the most noticeable features of our design. I originally wrote an algorithm for both cameras. In order to ensure both cameras read data correctly, I put one of the cameras upside down. In this way, this camera flips the data left to right and hence the program can process the data from both camera in the same way. In addition, because one camera is flipped, there is a height difference between two cameras. Hence, I tilted the horizontal bar of the camera stand to set them to be at the same height for both cameras.

I decided to make a widely adjustable camera stand with minimum tools for adjustments, So I chose wood and aluminum as the main materials. Our T-shaped camera stand allows the cameras to be 360 degree turnable so that we can easily adjust how far the cameras are looking ahead of the car. The vertical part of the camera stand can adjust the height of cameras as well as how forward or backward the cameras can view. By measurement, the stand can be adjusted approximately 45 degree vertically. More information about the drawn car design is discussed in the **Tool-free Modular Car** section**.**





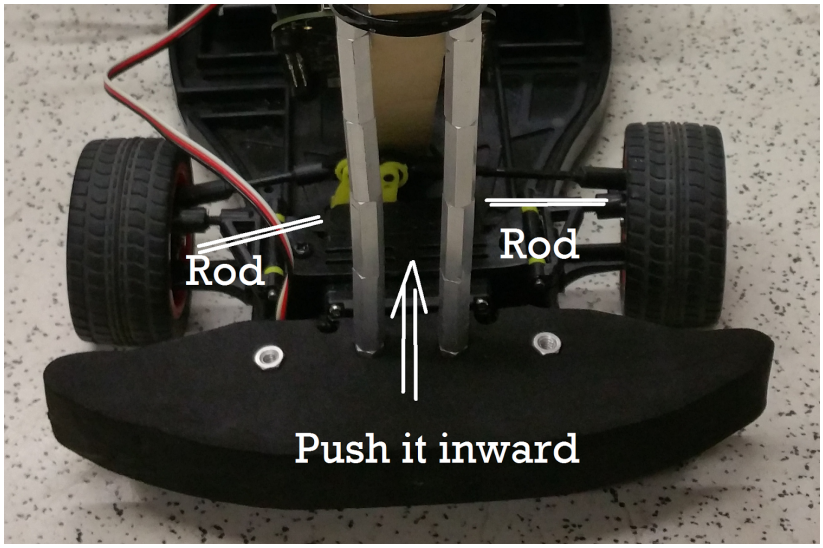
Upside down camera tilt adjustment **Triangle Configuration**

The T-shaped wooden camera stand attaches to a vertical aluminum nuts column to form a triangle for rigid configuration. The aluminum and wood are tied with releasable cable tie. The tie is flexible so that it reduces the impact when the camera stand hits the walls.

In fact, the adjustable camera stand makes trouble when the car hits some obstacles and changes the camera's orientation. As a result, I need a more advanced method to fix the camera orientation more tightly to avoid this issue. Avoiding collision with the camera stand is also an easy solution. (See section **Safety** for more information)

Servo adjustment

As discussed in the **Servo Control** section, the chassis blocks the wheels from turning to the desired position. The servo position also reduces the wheels' maximum turning capacities. The rods that connects the wheel and servo hits the chassis and limits the wheel from turning back to straight condition. The servo is moved slightly inward (~.5cm) to make sure the rods does not hit the chassis.

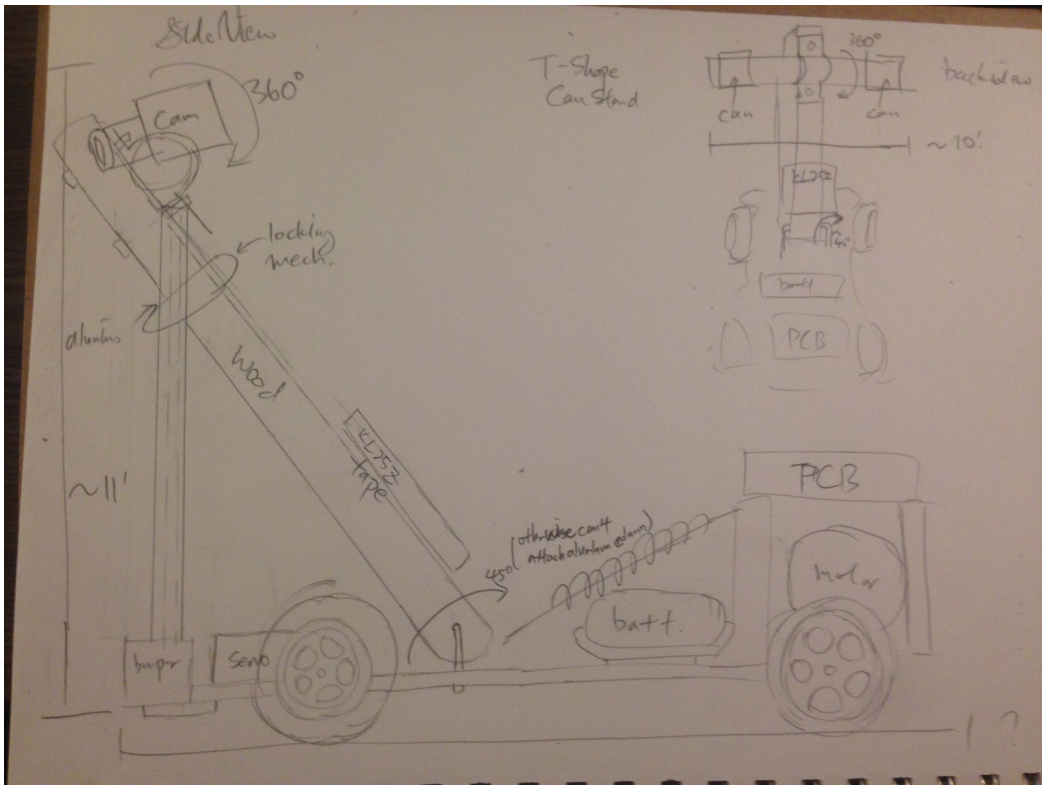


Detailed Technical Report: Hardware

This part is written by Lin, Lejin

Tool-free Modular Car

In hardware perspective, I tried to design our car as modular as possible. The flexibility in hardware enables the team to develop the design more efficiently. To build our car, I started off by drawing a blueprint, as shown below. In the blueprint, I also considered the dimensions of all important parts, including the horizontal length of the camera stand and the angle that the stand can be adjusted.

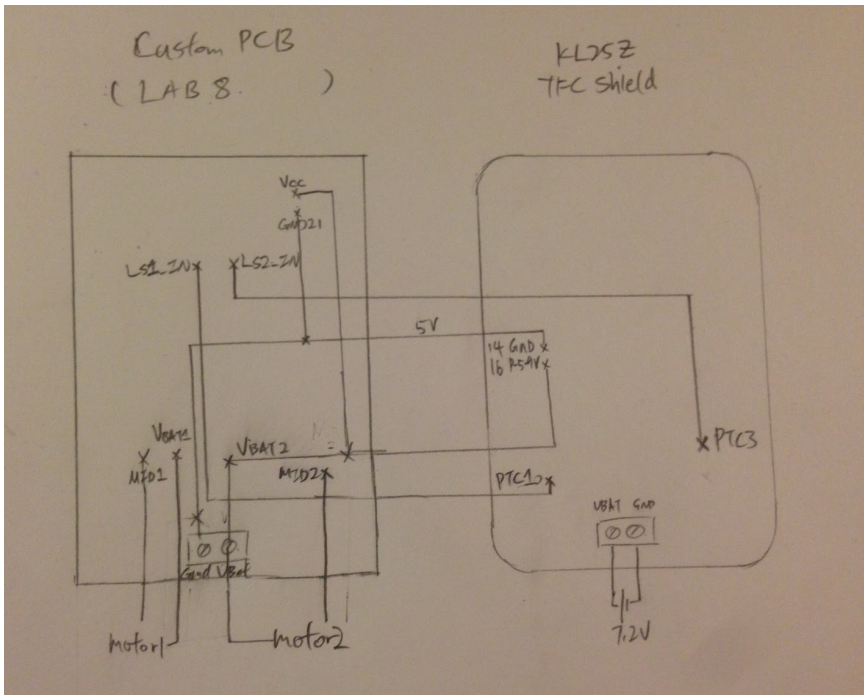


Car Design Blueprint

Tool-free: Component Attachment: I used Velcro tape to attach the Freescale KL25Z on the camera stand. I realized the Velcro moderately affects the chips' heat dissipation. However, I measured the temperature around this portion and its temperature range was around 30 degree Celsius. I considered this temperature is within an acceptable range and it will not burn the board or affect the performance. After I discussed with my team members, we decided to keep this configuration. I also used the Velcro tape on the custom PCB motor control board and stuck the board on the motors at the rear end of the car.

Custom motor control configuration

Instead of remaking the breadboard and the custom PCB which were done in Lab 8, I came up with a simpler approach. The PCB board controls the motor as required in the document. The TFC Shield still controls the servo and the cameras. The advantage of this approach is the design avoids using a breadboard. The following figure shows the connection between both boards.



Instead of using voltage regulator, I drawed 5V power from 16. P5-9V_VIN to deliver 5V to both motors. I designed this method because 7.2V is a large voltage for running motors. To protect the motors from burning, the microprocessor resets itself into the initial non-racing state automatically. In order to overcome the issue, I set the initial state to a race state to bypass this issue. The temperature of the wire reveals that the voltage causes problems. I found a way to draw 5V from TFC shield so that I did not have to use a breadboard for only a voltage regulator. In addition, I did not need to pay additional attention on noise reduction since the noisy motors are separated away from the camera and servo.

Design and Performance Summary

Camera Improvement: There are many way to improve the camera stand. If we would redesign the hardware again, we would use 3D printing to print the camera stand instead of use wooden. In this way, it would look more like a final product. We intended to make the adjustment much more flexible, so we would also separate the turning adjustment of each camera. After we got the best focus point for the cameras, we would fix the focus in position instead of manually look for the focus every time. In addition to fixing the two cameras on both sides, we would try to fix them vertically in different angles so that the cameras could see the track in broader view.

Safety

The safety mechanism we have implemented was the use of the last reasonable error value of the track. The purpose of this value is to force the car back to the racecourse if the car leaves the track. More discussions about the design of the last error value are in the **Camera Analysis** section. In addition to keep the last error value, we have also implemented the servo and the motors to make the car safely run across sharp turns. When the car is about to run outside of turns, one of the motor drops its speed to the lowest value so that the car actually spins in circle and forces itself back on track.

However, this safety method is not applicable for all cases. The method relies on the angles that the car enters into the turns. It is difficult to use only one function to determine how much the car should turn for all cases. In some situations, the car can not turn enough and thus run off track. In the spring quarter, we will improve the algorithm to have different solutions for different cases. So far, we have had an general idea on the software part. We will set a timer to count how long both cameras do not read any value then turn both motors off afterward. This implementation can be further refined so that the car stops when it finishes the track at the goal.

Appendixes

Appendix A: System Code

```
#include "MKL25Z4.h"
#include "mainprint.h"
#include "adc16.h"
#include "timers.h"

const uint32_t led_mask[] = {1UL << 18, 1UL << 19, 1UL << 1};

volatile unsigned short PW1 = 300;
volatile unsigned short PW2 = 300;
volatile unsigned short PW3 = 4500;

volatile unsigned char valIFB_A, valIFB_B, diffFB_A, diffFB_B;
volatile unsigned char IFB_Done = 0;
```

```
volatile char valPing[128], valPong[128];
volatile char valPing2[128], valPong2[128];
volatile unsigned int ind=0;
volatile char Done_Flag=0;

                                                                    // if Done_Flag == 0 : Pong buffer store, Ping buffter anaylze
volatile char Done_Ping=0;
volatile char Done_Pong=0;
volatile char camFlag = 0;

                                                                    // if camFlag == 1 : Read value from Camera #2
int blackCounter1 = 0;
int blackCounter2 = 0;
volatile unsigned short lastSixError[6];
int k = 0;
int hillUPCounter = 0, hillDOWNCounter = 0;

#define LED_RED    0
#define LED_GREEN  1
#define LED_BLUE     2
#define LED_A      0
#define LED_B      1
#define LED_C      2
#define LED_D      3
#define LED_CLK    4
#define tolerro1    28 //22 //32
#define tolerro2    20 //22 //42
#define pg                              7
#define PWstr                 200
#define PWright     0
#define PWleft                0
#define currentFB   14
#define maxSpeed    320


char uart0_getchar()
{
      /* Wait until character has been received */
      while (!(UART0->S1 & UART0_S1_RDRF_MASK));

      /* Return the 8-bit data from the receiver */
      return UART0->D;
}

void uart0_putchar (char ch)
{
      /* Wait until space is available in the FIFO */
      while(!(UART0->S1 & UART0_S1_TDRE_MASK));

      /* Send the character */
      UART0->D = (uint8_t)ch;

 }
#if UART_MODE == INTERRUPT
void UART0_IRQHandler (void)
{
  char c = 0;
  if (UART0->S1&UART_S1_RDRF_MASK)
  {
    c = UART0->D;

    if ((UART0->S1&UART_S1_TDRE_MASK)||(UART0->S1&UART_S1_TC_MASK))
    {
          UART0->D  = c;
    }
  }
}
#endif


/*****************************************************************
*****                      TPM Handler                         *****
*****************************************************************/
void TPM0_IRQHandler(void) {
//clear pending IRQ
          NVIC_ClearPendingIRQ(TPM0_IRQn);

// clear the overflow mask by writing 1 to CHF

          if (TPM0->CONTROLS[0].CnSC & TPM_CnSC_CHF_MASK)
                    TPM0->CONTROLS[0].CnSC |= TPM_CnSC_CHF_MASK;

          if (TPM0->CONTROLS[2].CnSC & TPM_CnSC_CHF_MASK)
                    TPM0->CONTROLS[2].CnSC |= TPM_CnSC_CHF_MASK;
}
```

```
void TPM1_IRQHandler(void) {
//clear pending IRQ
          NVIC_ClearPendingIRQ(TPM1_IRQn);

// clear the overflow mask by writing 1 to CHF

          if (TPM1->CONTROLS[0].CnSC & TPM_CnSC_CHF_MASK)
                    TPM1->CONTROLS[0].CnSC |= TPM_CnSC_CHF_MASK;
}

void put(char *ptr_str)
{
          while(*ptr_str)
                    uart0_putchar(*ptr_str++);
}

void printHexAscii(char value)
{
          char val1, val2;

          val1 = value & (0xF0);
          val1 = val1 >> 4;
          val2 = value & (0x0F);

          if(val1 < 10)
                    uart0_putchar('0' + val1);
          else
                    uart0_putchar('A' + val1 - 0x0A);

          if(val2 < 10)
                    uart0_putchar('0' + val2);
          else
                    uart0_putchar('A' + val2 - 0x0A);

}

/*----------------------------------------------------------------------------
 *      Main: Initialize
 *----------------------------------------------------------------------------*/
int main (void) {

          char valPingBinaryVT[128], valPongBinaryVT[128];   // Voltage Threshold Buffer of 1 (light) and 0 (dark)
          char valPingBinaryVT2[128], valPongBinaryVT2[128];   // Voltage Threshold Buffer of 1 (light) and 0 (dark)
          char centerPingIndexVT, centerPongIndexVT, centerPingIndexVT2, centerPongIndexVT2;
          char firstOne, lowZeroIndex, highZeroIndex;
          char max, min, diff, average;
          unsigned int sum=0;
          int error1, error2;

          char slopeThreshold = 0x09;
          char trackThreshold = 0x3D;

          unsigned int i=0;
          char key;
          int uart0_clk_khz;
          unsigned char valPOT1, valPOT2;
          char str[] = "\r\nEnter 'p' to print buffer\r\n";
  char str2[] = "\r\nEnter 'c' to continue or 'q' to quit\r\n";

          SystemCoreClockUpdate();

          /* Enable the pins for the selected UART */
          /* Enable the UART_TXD function on PTA1 */
          SIM->SCGC5 |= (SIM_SCGC5_PORTA_MASK
                                                                                | SIM_SCGC5_PORTB_MASK
                                                                                | SIM_SCGC5_PORTC_MASK
                                                                                | SIM_SCGC5_PORTD_MASK
                                                                                | SIM_SCGC5_PORTE_MASK );
          SIM->SOPT2 |= SIM_SOPT2_PLLFLLSEL_MASK; // set PLLFLLSEL to select the PLL for this clock source
          SIM->SOPT2 |= SIM_SOPT2_UART0SRC(1); // select the PLLFLLCLK as UART0 clock source

          PORTA->PCR[1] = PORT_PCR_MUX(0x2);               // Enable the UART0_RX function on PTA1
          PORTA->PCR[2] = PORT_PCR_MUX(0x2);               // Enable the UART0_TX function on PTA2

          uart0_clk_khz = (48000000 / 1000); // UART0 clock frequency will equal half the PLL frequency
          uart0_init (uart0_clk_khz, TERMINAL_BAUD);

  GPIO_Initialize();           /* Initialize the LEDs           */

          init_ADC0();                            // initialize and calibrate ADC0
          Init_PWM();

          Init_PIT(10000);
                                                                  // count-down period = 100 us - 10KHz
```

```
          put("Please turn on the power supply and press SW2...\r\n");
          while(!(FPTC->PDIR & (1UL << 17)));
// wait for SW2 pressed

          FPTE->PSOR |= (1UL << 21);
                              /* enable H-Bridge */

          put("Please set duty cycles using POT1 and POT2...\r\n");

          while(!(FPTC->PDIR & (1UL << 13)))
// while SW1 not pressed
          {
                    // read one value from ADC0 using software triggering and polling
                    ADC0->SC1[0] = (AIEN_ON | DIFF_SINGLE | ADC_SC1_ADCH(13)); // start conversion on channel SE13
                    while (!(ADC0->SC1[0] & ADC_SC1_COCO_MASK)) {     ; }                // wait for conversion to complete
(polling)
                    valPOT1 = ADC0->R[0];                                        // read result register
                    PW1 = (valPOT1 << 1) + 60;
                    TPM0->CONTROLS[2].CnV = PW2;
                    printHexAscii(valPOT1);

                    put("|");

                    ADC0->SC1[0] = (AIEN_ON | DIFF_SINGLE | ADC_SC1_ADCH(12)); // start conversion on channel SE13
                    while (!(ADC0->SC1[0] & ADC_SC1_COCO_MASK)) {     ; }                // wait for conversion to complete
(polling)
                    valPOT2 = ADC0->R[0];                                        // read result register
                    PW2 = (valPOT2 << 1) + 60;
                    TPM0->CONTROLS[0].CnV = PW2;
;
                    printHexAscii(valPOT2);

                    put("   ");

          }
          put("\r\nExit!\r\n");


          Start_PIT();

          /* Enable Interrupts */
          NVIC_SetPriority(ADC0_IRQn, 64); // 0, 64, 128 or 192
          NVIC_ClearPendingIRQ(ADC0_IRQn);
          NVIC_EnableIRQ(ADC0_IRQn);

          __enable_irq();

          TPM1->CONTROLS[0].CnV = PW3;

          put(str);


          // Main Loop
          while(1)
          {
                    // Parameter Initialization
                    max = 0;
                    min = 0xD8;
                    diff = 0;
                    sum = 0;

                    firstOne = 0;
                    lowZeroIndex = 0;
                    highZeroIndex = 0;

                    FPTB->PSOR   = led_mask[LED_RED];                /* Red LED Off*/
                    FPTB->PSOR   = led_mask[LED_GREEN];     /* Green LED Off*/
                    FPTD->PSOR   = led_mask[LED_BLUE];      /* Blue LED Off*/


                    // If Keyboard Input
                    if(uart0_getchar_present())
                    {
                              key = uart0_getchar();

                              while(key != 'p')
                              {
                                        put("\r\nWrong Command!\r\n");
                                        key = uart0_getchar();
                              }

                              Stop_PIT();

                              if(Done_Flag)
```

```
                              // Print Pong Buffers
{
          put("\r\nIFB Values:   ");
          printHexAscii(valIFB_A);
          put("|");
          printHexAscii(valIFB_B);

          put("\r\n\r\n#1 Camera Print Pong Buffer:\r\n");
          for(i=0; i<128; i++)
          {
                    printHexAscii(valPong[i]);
                    uart0_putchar(' ');
          }

          put("\r\n\r\nVoltage Threshold Buffer:\r\n");
          for(i=0; i<128; i++)
          {
                    printHexAscii(valPongBinaryVT[i]);
                    uart0_putchar(' ');
          }

          if(centerPongIndexVT)
          {
                    put("\r\nVT Center of Track Pixel Index: ");
                    printHexAscii(centerPongIndexVT);
                    put("\r\n");
          }
          else
                    put("\r\nVT Track Not Found\r\n");

          /*******************************************/

          put("\r\n\r\n#2 Camera Print Pong Buffer:\r\n");
          for(i=0; i<128; i++)
          {
                    printHexAscii(valPong2[i]);
                    uart0_putchar(' ');
          }
          put("\r\n\r\nVoltage Threshold Buffer:\r\n");
          for(i=0; i<128; i++)
          {
                    printHexAscii(valPongBinaryVT2[i]);
                    uart0_putchar(' ');
          }

          if(centerPongIndexVT2)
          {
                    put("\r\nVT Center of Track Pixel Index: ");
                    printHexAscii(centerPongIndexVT2);
                    put("\r\n");
          }
          else
                    put("\r\nVT Track Not Found\r\n");
}

/*********************************************
 *********************************************/

else

                    // Print Ping Buffers
{
          put("\r\nIFB Values:   ");
          printHexAscii(valIFB_A);
          put("|");
          printHexAscii(valIFB_B);

          put("\r\n\r\n#1 Camera Print Ping Buffer:\r\n");
          for(i=0; i<128; i++)
          {
                    printHexAscii(valPing[i]);
                    uart0_putchar(' ');
          }

          put("\r\n\r\nVoltage Threshold Buffer:\r\n");
          for(i=0; i<128; i++)
          {
                    printHexAscii(valPingBinaryVT[i]);
                    uart0_putchar(' ');
          }

          if(centerPingIndexVT)
          {
                    put("\r\nVT Center of Track Pixel Index: ");
                    printHexAscii(centerPingIndexVT);
                    put("\r\n");
```

```
                                        }
                                        else
                                                put("\r\nVT Track Not Found\r\n");

                                        /*****************************************/

                                        put("\r\n\r\n#2 Camera Print Ping Buffer:\r\n");
                                        for(i=0; i<128; i++)
                                        {
                                                printHexAscii(valPing2[i]);
                                                uart0_putchar(' ');
                                        }
                                        put("\r\n\r\nVoltage Threshold Buffer:\r\n");
                                        for(i=0; i<128; i++)
                                        {
                                                printHexAscii(valPingBinaryVT2[i]);
                                                uart0_putchar(' ');
                                        }

                                        if(centerPingIndexVT2)
                                        {
                                                put("\r\nVT Center of Track Pixel Index: ");
                                                printHexAscii(centerPingIndexVT2);
                                                put("\r\n");
                                        }
                                        else
                                                put("\r\nVT Track Not Found\r\n");

                                }

                                put(str2);

                                while(!uart0_getchar_present());
                                key = uart0_getchar();

                                while(key != 'c' && key != 'q')
                                {
                                        put("\r\nWrong Command!\r\n");
                                        key = uart0_getchar();
                                }
                                if(key == 'c')
                                {
                                        put("\r\nContinue.\r\n");
                                        Start_PIT();
                                }
                                else if(key == 'q')
                                {
                                        put("\r\nQuit.\r\n");
                                }
                        }

/*************************************************************************
 *************************************************************************/

                        // Main Analysis
                        else
                        {
                                if(Done_Ping || Done_Pong)
                                {
                                        if(Done_Flag && Done_Pong)
                                        {
//                                      Camera #1 Pong Buffer Analysis
                                                for(i=0;i<128;i++)
                                                {
                                                        if(valPong[i] < 0x30){
                                                                valPongBinaryVT[i] = 0;
                                                                blackCounter1++;
                                                        }
                                                        else
                                                        {
                                                                valPongBinaryVT[i] = 1;
                                                                if(!firstOne)
                                                                        lowZeroIndex = i;
                                                                firstOne = 1;
                                                        }


                                                // Calculate Cam 1 Voltage Binary Buffer

                                                        if(firstOne && valPongBinaryVT[i] == 0)
                                                        {
                                                                        highZeroIndex = (i-1);
                                                          firstOne = 0;
                                                        }

                                                }
```

```
                                        centerPongIndexVT = lowZeroIndex;

                                if(blackCounter1 >= 110){
                                                        blackCounter1 = 0;
                                                        error1 = error1;   //all black then keep last
error
                                                        //error2 = error2;
                                                }
                                        else {
                                                        error1 = centerPongIndexVT;
                                        //error2 = centerPongIndexVT2;
                                                        }



                                // Use VT to Find the Center of the Track
                                        // Parameter Initialization
                                        max = 0;
                                        min = 0xD8;
                                        diff = 0;
                                        sum = 0;

                                        firstOne = 0;
                                        lowZeroIndex = 0;
                                        highZeroIndex = 0;

//                                      Camera #2 Pong Buffer Analysis


                                        for(i=0;i<128;i++)
                                        {
                                                //for(k = 0; k <6; k++){
                                                if(valPong2[i] < 0x30){
                                                        valPongBinaryVT2[i] = 0;  //black
                                                        blackCounter2++;
                                                }
                                                else
                                                {
                                                        valPongBinaryVT2[i] = 1;  //white
                                                        if(!firstOne)
                                                                lowZeroIndex = i;
                                                        firstOne = 1;
                                                }


                                // Calculate Cam 2 Voltage Binary Buffer

                                                        if(firstOne && valPongBinaryVT2[i] == 0)
                                                        {
                                                                        highZeroIndex = (i-1);
                                                          firstOne = 0;
                                                        }

                                                        //lastSixError[k] = lowZeroIndex;
                                                //}
                                                //if(k == 5){k =0;}
                                        }

                                        centerPongIndexVT2 = lowZeroIndex;

                                        if (blackCounter2 >= 110){
                                                blackCounter2 = 0;
                                                //error1 = error1;   //all black then keep last error
                                                //TPM1->CONTROLS[0].CnV = 6000; // error2
                                                error2 = error2;
                                        }
                                        else {
                                                //error1 = centerPongIndexVT;
                                                error2 = centerPongIndexVT2;
                                        }

                                        //error2 = centerPongIndexVT2;



                                        if(error1 >= tolerro1 && error2 < tolerro2)  // turn right

                                        {
                                                //FPTB->PCOR  = led_mask[LED_RED];
                                                //PW3 = 4500 + pg*error1 ;
                                                PW3 = 4500 + (pg*error1)^4;
                                                if(PW3 >= 5300){ PW3 = 5300;} // 5500
                                                //if(error1 > (tolerro1 + 30)){
                                                TPM0->CONTROLS[0].CnV = PWstr;
```

```
                                                        TPM0->CONTROLS[2].CnV = PWleft;
                                                        //}
                                                        TPM1->CONTROLS[0].CnV = PW3; //servo

                                                }
                                                else
                                                        if(error2 >= tolerro2 && error1 < tolerro1 && centerPongIndexVT2 !=
0) // turn left
                                                        {
                                                                /*if (blackCounter2 >= 120){
                                                                        blackCounter2 = 0;
                                                                //error1 = error1;   //all black then keep last error
                                                                TPM1->CONTROLS[0].CnV = 3000; // error2

                                                                }*/
                                                                //else{
                                                                //FPTD->PCOR   = led_mask[LED_BLUE];
                                                                //PW3 = 4500 - pg *error2;
                                                                PW3 = 4500 - (pg *error2)^4;

                                                                if(PW3 <= 3600){ PW3 = 3600;}//3600
                                                                //if(error2 > (tolerro2 + 30))
                                                                //{
                                                                        TPM0->CONTROLS[0].CnV = PWright;
                                                                        TPM0->CONTROLS[2].CnV = PWstr;
                                                                //}
                                                                TPM1->CONTROLS[0].CnV = PW3;
                                                        //}
                                                        }
                                                        else
                                                                {
                                                                        /*if(valIFB_A < currentFB || valIFB_B < currentFB
||valIFB_A > currentFB || valIFB_B > currentFB){
                                                                                if(valIFB_A < currentFB || valIFB_B <
currentFB){
                                                                                        diffFB_A = (valIFB_A -
currentFB) *150; //current feedback
                                                                                        diffFB_B = (valIFB_B -
currentFB) *150;
                                                                                        //TPM0->CONTROLS[0].CnV =
100;
                                                                                        //TPM0->CONTROLS[2].CnV =
100;
                                                                                        TPM0->CONTROLS[0].CnV = PWstr
+ diffFB_A;
                                                                                        TPM0->CONTROLS[2].CnV = PWstr
+ diffFB_B;

                                                                                        FPTB->PCOR   =
led_mask[LED_RED];
                                                                                }
                                                                                else if(valIFB_A > currentFB ||
valIFB_B > currentFB){
                                                                                        diffFB_A = (valIFB_A -
currentFB);
                                                                                        diffFB_B = (valIFB_B -
currentFB);
                                                                                        //TPM0->CONTROLS[0].CnV =
PWstr + diffFB_A;
                                                                                        //TPM0->CONTROLS[2].CnV =
PWstr + diffFB_B;
                                                                                        TPM0->CONTROLS[0].CnV = PWstr
- 200;
                                                                                        TPM0->CONTROLS[2].CnV = PWstr
- 200;
                                                                                        while(hillUPCounter < 30){
                                                                                                hillUPCounter++;


TPM0->CONTROLS[0].CnV = 50;
                                                                                        hillUPCounter = 0;
TPM0->CONTROLS[2].CnV = 50;}
                                                                                        //while(hillDOWNCounter <
20){hillDOWNCounter++;}
                                                                                        //hillDOWNCounter = 0;
                                                                                        FPTB->PCOR   =
led_mask[LED_GREEN];
                                                                                }
                                                                                PW3 = 4500;
                                                                                //TPM0->CONTROLS[0].CnV = PWstr +
diffFB_A;
                                                                                //TPM0->CONTROLS[2].CnV = PWstr +
diffFB_B;
                                                                                TPM1->CONTROLS[0].CnV = PW3;
                                                                        }*/
```

```
                                                                        /*if(valIFB_A < currentFB || valIFB_B <
currentFB){
                                                                                valIFB_A = currentFB;
                                                                                valIFB_B = currentFB;
                                                                        }
                                                                        else if(valIFB_A > currentFB || valIFB_B >
currentFB){
                                                                                valIFB_A = currentFB;
                                                                                valIFB_B = currentFB;
                                                                        }*/
                                                                        //FPTB->PCOR   = led_mask[LED_GREEN];

                        // LED Change Color Based on ST Center of Track
                                                                        if(valIFB_A > currentFB || valIFB_B > currentFB){
                                                                                diffFB_A = (valIFB_A -
currentFB);
                                                                                diffFB_B = (valIFB_B -
currentFB);
                                                                                //TPM0->CONTROLS[0].CnV =
PWstr + diffFB_A;
                                                                                //TPM0->CONTROLS[2].CnV =
PWstr + diffFB_B;
                                                                                TPM0->CONTROLS[0].CnV = PWstr
- 50;
                                                                                TPM0->CONTROLS[2].CnV = PWstr
- 50;

                                                                                PW3 = 4500;

                                                                                TPM1->CONTROLS[0].CnV = PW3;
                                                                                }
                                                                        else{
                                                                                PW3 = 4500;
                                                                                TPM0->CONTROLS[0].CnV = PWstr - 10;
                                                                                TPM0->CONTROLS[2].CnV = PWstr - 10;
                                                                                TPM1->CONTROLS[0].CnV = PW3;
                                                                        }
                                                                        /*if(TPM0->CONTROLS[0].CnV >= maxSpeed ||
TPM0->CONTROLS[2].CnV >= maxSpeed){
                                                                                TPM0->CONTROLS[0].CnV = maxSpeed;
                                                                                TPM0->CONTROLS[2].CnV = maxSpeed;
                                                                        }*/
                                                                }
                                                        //Print Values onto terminal
                                                        printHexAscii(centerPongIndexVT);
                                                        uart0_putchar('/');
                                                        printHexAscii(centerPongIndexVT2);
                                                        uart0_putchar(' ');

                                                        Done_Pong = 0;
                                                }

                                /*********************************************/
                                                else
                                                        if(Done_Ping)
                                                        {
//                                                      Camera #1 Ping Buffer Analysis
                                                                for(i=0;i<128;i++)
                                                                {
                                                                        if(valPing[i] < 0x30){
                                                                                valPingBinaryVT[i] = 0;
                                                                                blackCounter1++;
                                                                        }
                                                                        else
                                                                        {
                                                                                valPingBinaryVT[i] = 1;
                                                                                if(!firstOne)
                                                                                        lowZeroIndex = i;
                                                                                firstOne = 1;
                                                                        }


                                                                // Calculate Cam 1 Voltage Binary Buffer

                                                                        if(firstOne && valPingBinaryVT[i] == 0)
                                                                        {
                                                                                highZeroIndex = (i-1);
                                                                                firstOne = 0;
                                                                        }

                                                                }

                                                                        centerPingIndexVT = lowZeroIndex;

                                                                if(blackCounter1 >= 110){
                                                                        blackCounter1 = 0;
```

```
                                                  error1 = error1;   //all black then keep last
error
                                                  //error2 = error2;
                                        }
                                        else {
                                                  error1 = centerPongIndexVT;
                                        //error2 = centerPongIndexVT2;
                                                  }

                                        // Parameter Initialization


//                                      Camera #2 Ping Buffer Analysis
                                    for(i=0;i<128;i++)
                                        {
                                                  if(valPing2[i] < 0x30){
                                                            valPingBinaryVT2[i] = 0;
                                                            blackCounter2++;
                                                  }
                                                  else
                                                  {
                                                            valPingBinaryVT2[i] = 1;
                                                            if(!firstOne)
                                                                      lowZeroIndex = i;
                                                            firstOne = 1;
                                                  }


                                        // Calculate Cam 2 Voltage Binary Buffer

                                                  if(firstOne && valPingBinaryVT2[i] == 0)
                                                  {
                                                                      highZeroIndex = (i-1);
                                                                      firstOne = 0;
                                                  }

                                        }

                                        centerPingIndexVT2 = lowZeroIndex;

                                        // Use VT to Find the Center of the Track

                                        if(blackCounter2 >= 110){
                                                  blackCounter2 = 0;
                                                  //error1 = error1;   //all black then keep last error
                                                  error2 = error2;
                                        }
                                        else {
                                                  //error1 = centerPongIndexVT;
                                                  error2 = centerPongIndexVT2;
                                        }
                                                  //error2 = centerPongIndexVT2;


                                                  if(error1 >= tolerro1 && error2 < tolerro2)//if(PW3 >= 5500){ PW3 =
5500;}(1st:22)
                                        {
                                                  //FPTB->PCOR   = led_mask[LED_RED];
                                                  //PW3 = 4500 + error1 * pg;
                                                  PW3 = 4500 + (error1 * pg)^4;
                                                  if(PW3 >= 5300){ PW3 = 5300;}
                                                  //if(error1 > (tolerro1 + 30)){
                                                  TPM0->CONTROLS[0].CnV = PWstr;
                                                  TPM0->CONTROLS[2].CnV = PWleft;
                                                  //}
                                                  TPM1->CONTROLS[0].CnV = PW3;
                                        }
                                        else
                                                  if(error2 >= tolerro2 && error1 < tolerro1 && centerPongIndexVT2 !=
0)
                                                  {
                                                  /*if (blackCounter2 >= 120){
                                                            blackCounter2 = 0;
                                                  //error1 = error1;   //all black then keep last error
                                                  TPM1->CONTROLS[0].CnV = 3000; // error2

                                                  }*/
                                                  //else{
                                                  //FPTD->PCOR   = led_mask[LED_BLUE];
                                                  //PW3 = 4500 - error2 * pg;
                                                  PW3 = 4500 - (error2 * pg)^4;
                                                  if(PW3 <= 3600){ PW3 = 3600;} // 3600
                                                  //if(error2 > (tolerro2 + 30))
                                                  //{
                                                            TPM0->CONTROLS[0].CnV = PWright;
                                                            TPM0->CONTROLS[2].CnV = PWstr;
```

```
||valIFB_A > currentFB || valIFB_B > currentFB){

currentFB){

currentFB) *150; //current feedback

currentFB) *150;

100;

100;

+ diffFB_A;

+ diffFB_B;


led_mask[LED_RED];


valIFB_B > currentFB){

currentFB);

currentFB);

led_mask[LED_GREEN];

PWstr + diffFB_A;

PWstr + diffFB_B;

- 200;

- 200;


TPM0->CONTROLS[0].CnV = 50;

TPM0->CONTROLS[2].CnV = 50;}



20){hillDOWNCounter++;}




diffFB_A;

diffFB_B;




currentFB){




currentFB){









currentFB);

currentFB);

PWstr + diffFB_A;

PWstr + diffFB_B;
```

```
                //}
                TPM1->CONTROLS[0].CnV = PW3 ;
        //}
        }
        else
                {
                        /*if(valIFB_A < currentFB || valIFB_B < currentFB
                                if(valIFB_A < currentFB || valIFB_B <
                                        diffFB_A = (valIFB_A -
                                        diffFB_B = (valIFB_B -
                                        //TPM0->CONTROLS[0].CnV =
                                        //TPM0->CONTROLS[2].CnV =
                                        TPM0->CONTROLS[0].CnV = PWstr
                                        TPM0->CONTROLS[2].CnV = PWstr


                                        FPTB->PCOR   =
                                }
                                else if(valIFB_A > currentFB ||
                                        diffFB_A = (valIFB_A -
                                        diffFB_B = (valIFB_B -
                                        FPTB->PCOR   =
                                        //TPM0->CONTROLS[0].CnV =
                                        //TPM0->CONTROLS[2].CnV =
                                        TPM0->CONTROLS[0].CnV = PWstr
                                        TPM0->CONTROLS[2].CnV = PWstr
                                        while(hillUPCounter < 30){
                                                hillUPCounter++;



                                        hillUPCounter = 0;


                                        //while(hillDOWNCounter <
                                        //hillDOWNCounter = 0;
                                }

                                PW3 = 4500;
                                //TPM0->CONTROLS[0].CnV = PWstr +
                                //TPM0->CONTROLS[2].CnV = PWstr +
                                TPM1->CONTROLS[0].CnV = PW3;
                        }
                        */
                        /*if(valIFB_A < currentFB || valIFB_B <

                                valIFB_A = currentFB;
                                valIFB_B = currentFB;
                        }
                        else if(valIFB_A > currentFB || valIFB_B >

                                valIFB_A = currentFB;
                                valIFB_B = currentFB;
                        }*/

                        //FPTB->PCOR   = led_mask[LED_GREEN];
                        if(valIFB_A > currentFB || valIFB_B > currentFB){
                                        diffFB_A = (valIFB_A -
                                        diffFB_B = (valIFB_B -
                                        //TPM0->CONTROLS[0].CnV =
                                        //TPM0->CONTROLS[2].CnV =
                                        TPM0->CONTROLS[0].CnV = PWstr
```

```
- 50;
                                                                                TPM0->CONTROLS[2].CnV = PWstr
- 50;

                                                                                PW3 = 4500;

                                                                                TPM1->CONTROLS[0].CnV = PW3;
                                                                                }
                                                                                                        // LED
Change Color Based on ST Center of Track
                                                                        else{
                                                                                PW3 = 4500;
                                                                                TPM0->CONTROLS[0].CnV = PWstr - 10;
                                                                                TPM0->CONTROLS[2].CnV = PWstr - 10;
                                                                                TPM1->CONTROLS[0].CnV = PW3;
                                                                        }
                                                                        //}
                                                                        /*if(TPM0->CONTROLS[0].CnV >= maxSpeed ||
TPM0->CONTROLS[2].CnV >= maxSpeed){
                                                                                TPM0->CONTROLS[0].CnV = maxSpeed;
                                                                                TPM0->CONTROLS[2].CnV = maxSpeed;
                                                                        }*/
                                                                }
                                                        //Print Values onto terminal
                                                        printHexAscii(centerPingIndexVT);
                                                        uart0_putchar('/');
                                                        printHexAscii(centerPingIndexVT2);
                                                        uart0_putchar(' ');
                                                        Done_Ping = 0;
                                                }
                                        }
                                }
                        }
                }
```