A.R.T.S.S.

Automatic **R**ear-wheel **T**raction **S**poiler **S**ystem

Final Report

Santiago Ruiz

12/03/01

TABLE OF CONTENTS

ABSTRACT

ARTSS (Automatic Rear-wheel Traction Spoiler System) is meant to act as a co-pilot to the driver. He assists in increasing traction to the rear wheels by applying a constant level of down-force to the rear of the car. This will increase the level of frictional force that is opposing the tendency of the car to slide during high-speed turning situations and braking. The down-force is achieved through an adjustable rear spoiler system that senses the amount of force, through 2 FSR's (Force Sensitive Resistors), being applied by the airflow across the surface of the wing. ARTSS can also monitor speed, through a pulse emitting IR beam across the flywheel, so that the wing adjustments are made at speeds where there would actually be enough airflow to affect the down-force. ARTSS can increase the safety as well as the performance of rear-wheel-drive cars.

EXECUTIVE SUMMARY

Using an R/C car and a balsa wood spoiler as the test bed, ARTSS manipulates the pitch of the rear spoiler (with a servo) according to the speed of the car and down-force on the wing. The purpose is to apply more down-force to the rear of the car and increase the level of frictional force that is opposing the tendency of the car to slide during high-speed turning and braking situations. ARTSS is an autonomous system that uses two FSR's (Force Sensing Resistors) and an IR pulsing speedometer as inputs and controls the pitch of the spoiler servo as its only output.

At low speeds, the FSR's and servo are turned off. This is because at low speeds there is not enough airflow across the spoiler to have a significant effect on the down-force to increase traction. If the spoiler were engaged at these speeds it would simply act as a drag surface, keeping it off allows the car to get to an appropriate speed quicker. After a preprogrammed speed is reached the spoiler adjusting system is turned on. If the speed decreases to a point below the threshold, the servo and FSR's are once again turned off.

The FSR's vary in resistance according to how much force is being applied to them. These are connected to the HC11 microcontroller's A/D system. The voltage supplied by the pulled-up FSR's is between 3.5 and 4.5 volts. The HC11 translates these values to the range of the servo's motion. At maximum voltage the spoiler is at its smallest pitch angle from horizontal. In other words, the greater the down-force read, the less the pitch of the spoiler.

For testing I set up a 20 foot diameter circle. With the spoiler system engaged, lap times dropped by an average of about 10%, since the car was able to run at higher speeds with the extra down-force on the rear wheels keeping the car from sliding. Obviously, this result was predicted, but the purpose of the system is to also reduce the drag that is created by the spoiler when extra down-force is not necessary, as in straight acceleration situations at low speeds. On a track with 2 straightaways, the lap times decreased by an average of about 12%. This is attributed to the ability of the adjustable spoiler to turn off at speeds that are too low to have a significant effect on the traction of the vehicle. Basically, at speeds where the car would not normally slip, the wings drag is nullified.

Overall, the project was a success. I achieved my goals of integrating two inputs, an output, and an intelligence routine into an HC11 to provide more traction as was proved by the decreased lap times. A drawback to the system is, of course, the added weight, which altered the center of gravity of the car. However, with slight hardware changes, more sophisticated software, and additional sensors, like accelerometers, the system could definitely be improved.

INTRODUCTION

Spoilers have been prevalent in motorsports and commercial applications for over 40 years. The airflow across the flat surface create a low and high pressure region which force the car to “squat” and increase the force pushing down on the tires and therefore increase the friction between the tires and the ground. The trade off between increasing down-force is that you are also increasing drag. This drag reduces the efficiency of the car resulting in slower acceleration and decreased fuel economy. What if you could have all the positive effects of a spoiler and also, make the spoiler disappear when its negative effects become prevalent?

Porsche has a system where the spoiler lies flat against the car until it reaches 40 mph. At that point the spoiler lifts up providing some extra down-force, as well as better cooling for the engine compartment, which is located in the rear. This is a good system and a simple solution to the problem. But, what about straight line acceleration over 40 mph, the spoiler is a hindrance in most of these cases, as extra traction is not necessary until speeds are in excess of about 100 mph. Dodge has also recognized this issue and has added a driver adjustable rear spoiler to the latest release of the Viper. Again a simple solution to the problem as the pitch needs to be adjusted from the cabin by the driver, and once the pitch is set that is where it remains until it is changed again.

ARTSS is the start to a complete solution for the problem. An autonomous system which negates the drawbacks to fixed spoilers completely and can actually provide several more benefits. For example, during braking situations, the spoiler could go to full pitch to act as an “air brake” to help slow the car down. A system that is invisible when not useful and effective when necessary.

INTEGRATED SYSTEM LAYOUT

The system integration consists of simply 2 inputs, the IR speed sensor and the FSR's, to the micro controller and one output, the spoiler servo (*Figure 1a*).

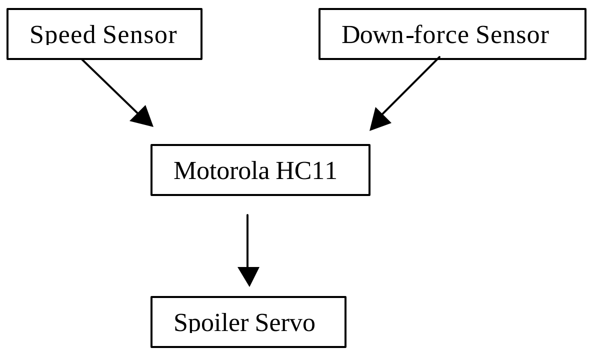


Figure 1a. I/O integrated system layout

The HC11 microcontroller is mounted on a Mekatronix MRC11 board with 64 Kbytes of expanded RAM. This will be on the rear of the vehicle, under the spoiler. The HC11 outputs varying voltage levels to a servo with a control arm attached to the spoiler. The output is dependant on the voltage levels attained through the A/D (Analog to Digital) port on the controller. The two FSR's are pulled up so that the more force that is sensed, the greater the voltage level applied to the controller's A/D port.

The speedometer is connected to one of the HC11's Input Capture (IC) ports. The IR sensor triggers an interrupt when the beam is broken (by one of the spokes on the car's flywheel). The time in between interrupts is used to calculate a numerical speed value for the car. The Output Compare (OC) feature of the HC11 is used to send a pulse width modulated (PWM) signal to the servo to control its position. The layout of the system is shown in the HC11 block diagram below for simplicity (*figure 1b*).

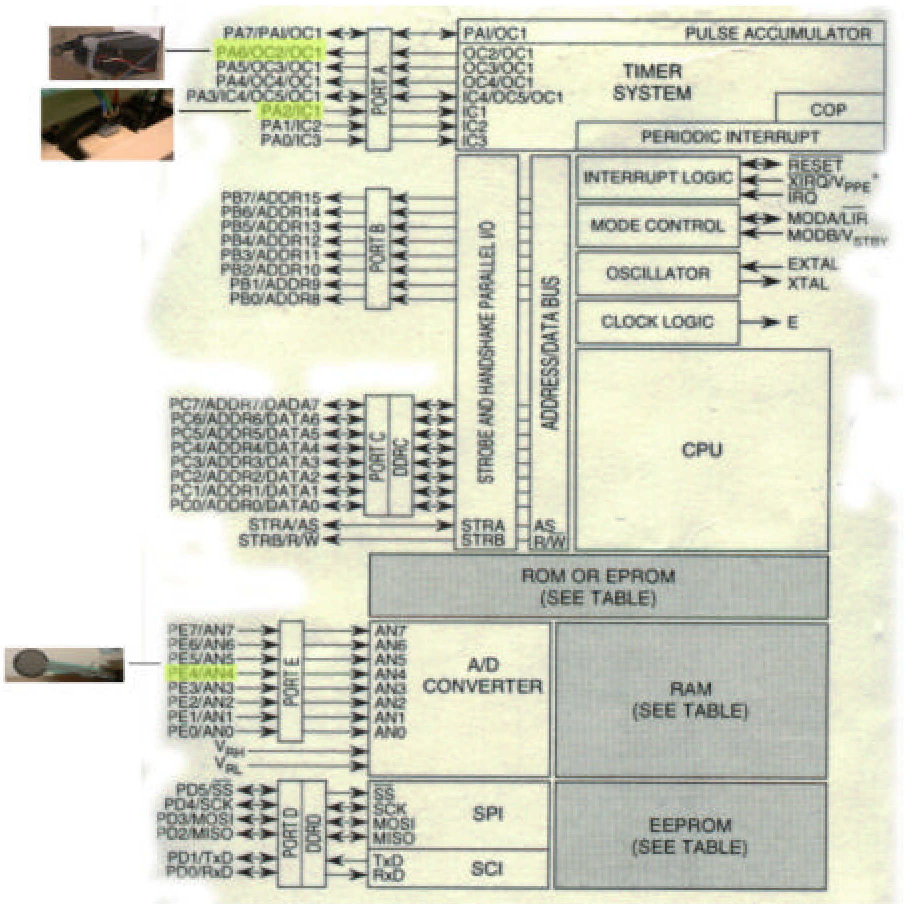


figure 1b. Microcontroller system integration block diagram

As far as software is concerned, there was not too much coding involved. The system recognizes that at a speed below *onSpeed* the servo and FSR's are not being used. The FSR voltage levels are checked in an infinite loop where they are constantly adjusting the servo position to maintain a preset load on the rear of the car. If the speed of the car falls below the *onSpeed* value, the spoiler adjustment is turned back off. The system keeps polling the speed of the car until it once again exceeds the threshold value. The flow chart in *figure2a* shows an illustration of the system's logic process.

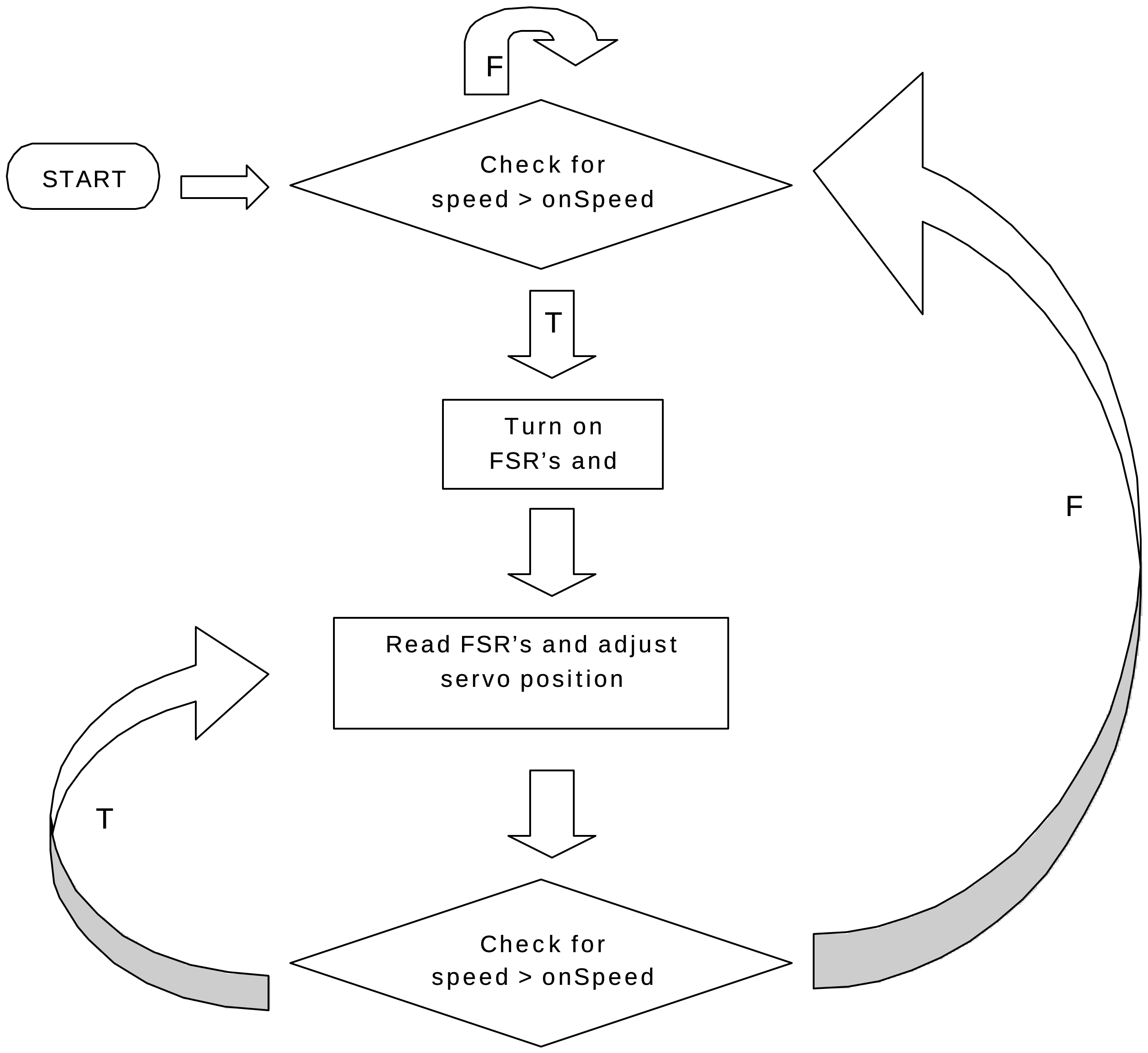
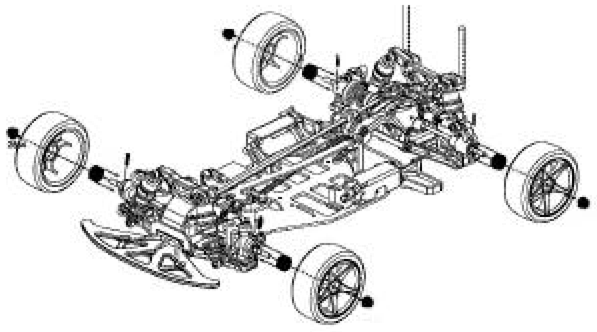
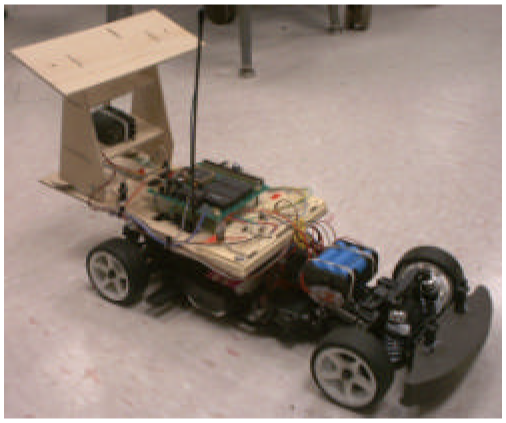

START
Check for speed > onSpeed
F
T
Turn on FSR's and
Read FSR's and adjust servo position
Check for speed > onSpeed
T
F

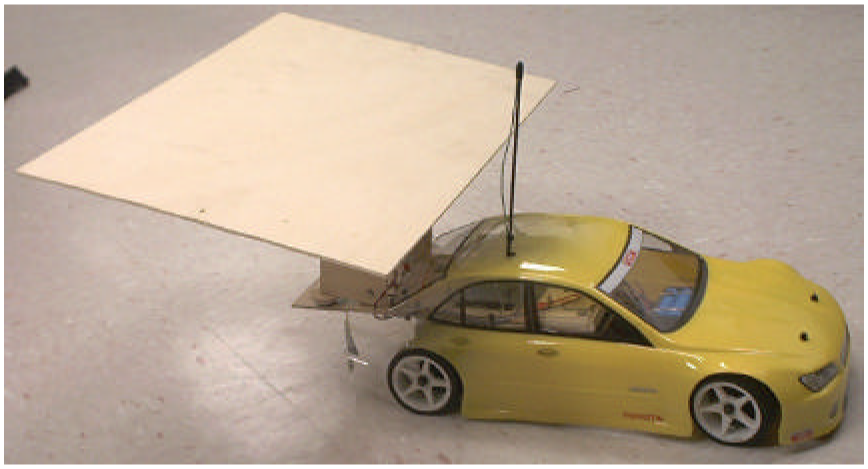
Figure 2a. system logic flow chart

MOBILE TEST PLATFORM

Using a 1/10 scale R/C car I can simulate the effects of wind deflection on a vehicle's traction due to the rear spoiler. To do this I need to use a very large spoiler surface area. This is due to the fact that the model does not exceed 30 mph, which is much less than a full-size passenger vehicle or race car. With so little airflow at this speed I need to either increase speed, which is not feasible, or simulate the down-force at higher speeds by increasing the amount of air deflection caused by the spoiler. The model has an electrically powered rear-wheel drivetrain, similar to many current sports and race cars. I chose rear-wheel drive over front-wheel or all-wheel, because rear tire traction has the greatest effect in controlling, or losing the control of, these cars in high-speed situations.







SPOILER ACTUATION

Spoiler pitch is controlled by a single high-torque servo. Mounted at the rear of the car on the base of the spoiler, it is the only output produced by the system. Using the HC11 OC feature I send out a PWM signal with a high pulse width between 1500 cycles/period (full pitch) and 3000 cycles/period (no pitch, flat). The input from the FSR's is multiplied by a ratio in that subroutine to attain the proper pulse width update for the servo. An adjustable rod connects the servo push arm to the spoiler base (*figure3a*). This allows for easy tweaking to configure the system to the proper base settings after assembly.

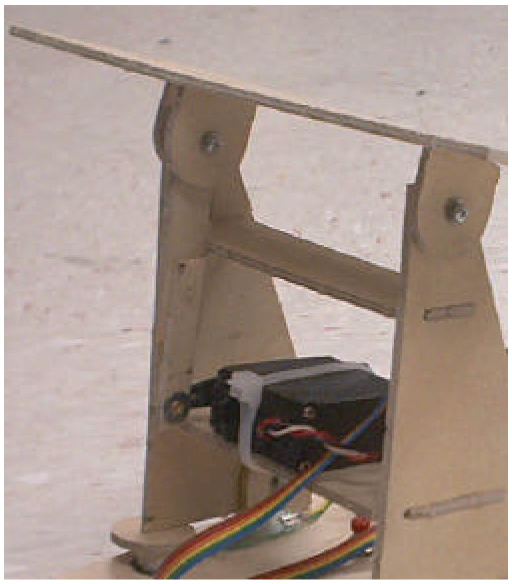


figure3a. Spoiler Actuation System

SENSORS

ARTSS has two sensor inputs, the FSR's and the IR pulse speedometer. The down-force sensing is handled by a pair of FSR's located at the base of the spoiler (*figure 4a*). They are pulled high and inputted into the A/D system of the HC11. They are wired in parallel as shown in *figure4b*. The output from the FSR circuit is in the range of 3.5V - 4.5V. This is correlated to a range of about 40 integer counts by the A/D system. This value is then used to calculate the new servo pulse width using the following equation:

$$\text{New Servo Setting} = 3000 - ((\text{ADvalue})/40) \times 1500$$

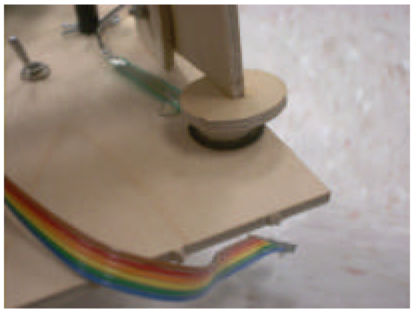


figure4a. FSR under spoiler base leg

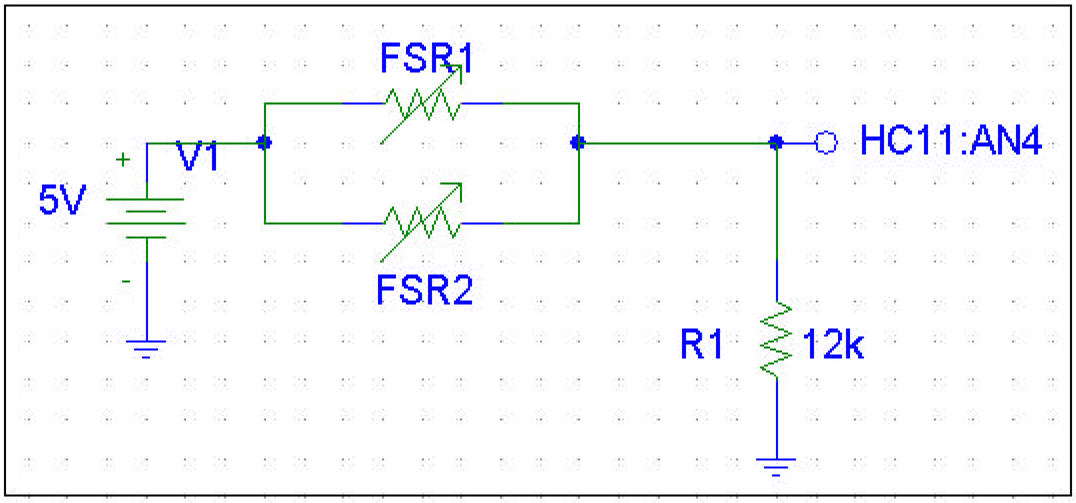


Figure4b. Force Sensing circuit diagram

The other sensor employed by ARTSS is a special sensor consisting of an IR emitter and a photosensitive transistor. The IR beam is shot across a flywheel on the electric motor of the car. When the IR beam is blocked by one of the spokes on the flywheel, the output through the transistor is 0V, and when it is unblocked, the output is 5V (*figure5a* and *figure5b*). The IC system of the HC11 is used to detect the high to low pulse and record the time. This time is then subtracted from the time of the next pulse to determine the speed of the wheel, as a numerical quantity.

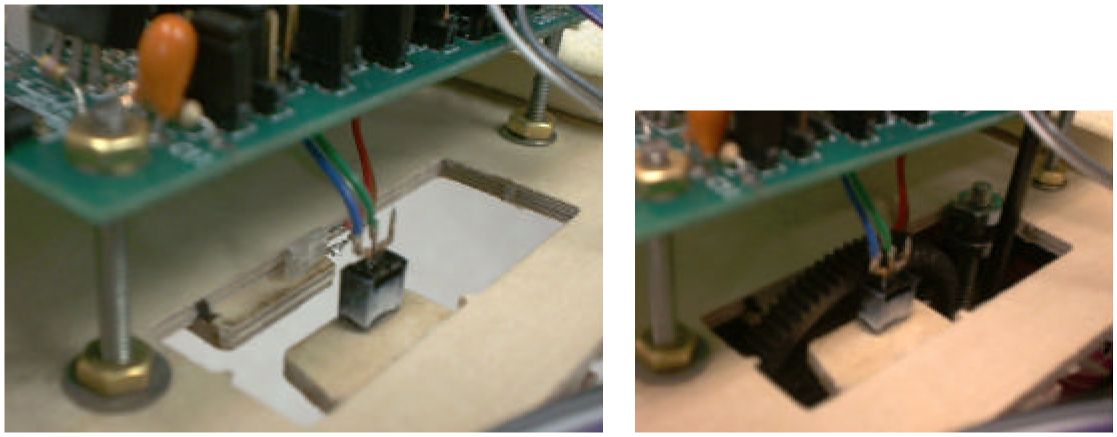


figure5b. Speedometer sensor off the car (left and on the car (right)

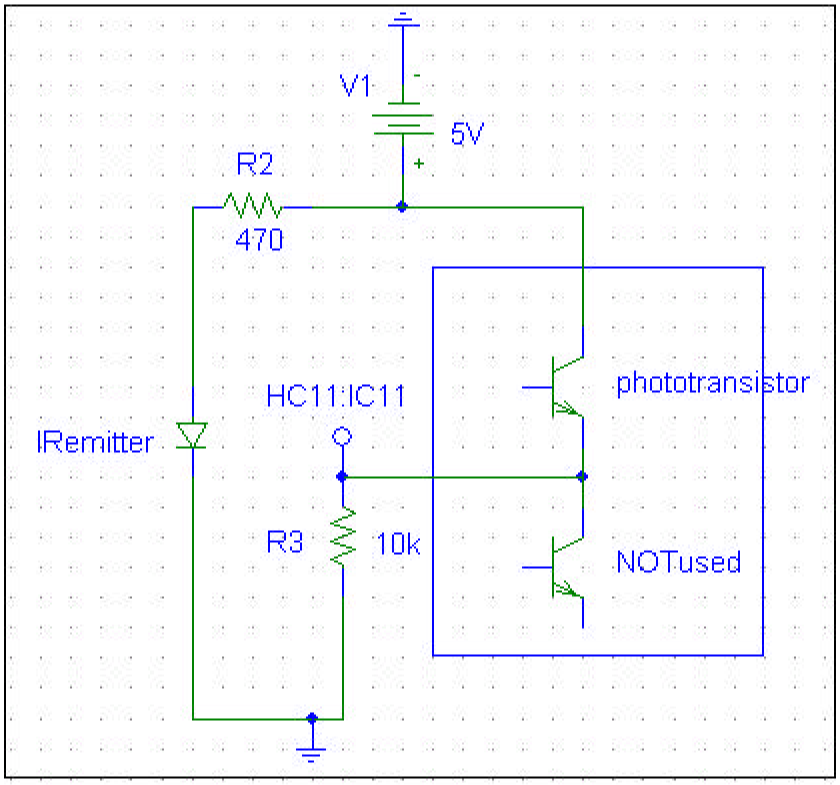


figure5b. Speedometer sensor circuit

EXPERIMENTAL LAYOUT AND RESULTS

To test ARTSS I set up a pair of tracks, one was just a circle with a 20ft diameter, the other was an oval with two straightaways and two 10ft radius curves. Clearly performance was going to improve, after all, spoiler systems are indeed proven devices for increasing lap times. The true test was to see if these times could be reduced even further by turning off the spoiler when not necessary. The results showed that this is possible:

CIRCLE TRACK:

RUN #	**TIME w/ WSPOILER (s)**	**TIME w/o SPOILER (s)**
1	10.3	11.0
2	9.8	10.7
3	9.9	10.6
4	9.3	10.8
5	9.1	10.5
AVERAGE	**9.68**	**10.72**

OVAL TRACK:

RUN #	**TIME w/ WSPOILER (s)**	**TIME w/ ARTSS (s)**	**TIME w/o SPOILER (s)**
1	19.2	18.9	21.3
2	19.5	19.1	20.9
3	18.9	18.9	21.0
4	19.2	18.5	20.8
5	18.8	18.3	20.7
AVERAGE	**19.12**	**18.74**	**20.94**

Through this limited pair of tests one can see that the spoiler does provide an improvement when engaged. Since the car ran at a consistent speed the entire time through the circle track test (timer started after car got up to speed), the ARTSS system was not effective. With the spoiler engaged to full, there is an improvement of 9.7% faster lap times. In the second test where the ARTSS system can be compared to no spoiler and full spoiler pitch, we can see a further decrease of about 2%. These results show that, as expected, the system does provide an advantage by increasing traction at the appropriate speeds.

CONCLUSION

It is no surprise that spoilers can improve performance and safety of rear-wheel-drive vehicles. This fact has been known for several years. This was not the goal of this project. ARTSS can further improve on that performance by being "invisible" to the drag of airflow, when the spoiler is not needed. There are some issues with the system. Too much weight on the rear of the car is one, but this can be fixed or improved by moving the system forward on the chassis. Another problem is vibration. The FSR's are too sensitive and respond to the most minimal bump in the road, making testing a nightmare. Ideally, I would want to switch to an airflow sensor to replace the FSR's, or integrate with the FSR's.

Clearly there are ways to improve the system, but these simple tests showed that I have accomplished my goal. A better performance, by 2% over a conventional fixed spoiler, through an autonomously adjusting spoiler system.

REFERENCES

1. Fred Martin, *The 6.270 Robot Builder’s Guide*, MIT Media Lab, Cambridge, MA, 1992

2. Motorola HC11 pink books and “bible”

APPENDIX (source code)

```
#include <hc11.h>
#include <mil.h>
#include <stdio.h>
#include <math.h>
#include <float.h>

/*Interrupt Procedures/Variables*/

#define DUMMY_ENTRY (void (*)(void)) 0xFFFF

#pragma interrupt_handler isrTOC2
#pragma interrupt_handler isrTIC1


  int menuDisplay = 1;

  int period = 40000;

  int pulseWidth_TOC2 = 3000;

  int pulseOn_TOC2 = 0;

  int totalPulses_TOC2 = 0;

  int speed = 0;

  int currentTime = 0;


  int prevTime = 0;

  int pulseTime = 0;
```

```
/************** Prototypes */
  extern void _start(void);

  void processSelection(int);
  void displayMenu(void);
  int parseInputToInt(void);
  void initializeSystem(void);
  void initializeServos(void);
  void init_ic1(void);
  int readAD(void);
  void FSRloop(void);
  void isrTOC2(void);
  void isrTIC1(void);

/**************  MAIN*/

  void main(void)
  {
    char menuSelection = '0';

    initializeSystem();

    /*while (menuSelection != 9)
    {
      displayMenu();
      menuSelection = ( (int)getchar() ) - (0x30);
      processSelection(menuSelection);
    }*/
        menuSelection = 6;
        processSelection(menuSelection);

    printf("System. off.\n");
  }

/*********** set baud and init */

  void initializeSystem(void)
  {
    printf("Initializing System.\n");
    setbaud(BAUD9600);
    initializeServos();
    SET_BIT(OPTION, 0x80);//turn on A/D

    totalPulses_TOC2 = 0;

    init_serial();
          init_ic1();
    printf("Done Initializing.\n");

  }

/************ Menu display */

  void displayMenu(void)
  {
    if (menuDisplay)
    {
      printf("\n\n");
      printf("     MENU\n\n");
      printf("1.)  Adjust Motor\n");
      printf("2.)  Set Speed\n");
      printf("3.)  Turn On Interrupts.\n");
      printf("4.)  Turn Off Interrupts.\n");
      printf("5.)  Read A/D 4.\n");
      printf("6.)  FSR control loop.\n");
```

```
        printf("7.)  IC1 pulse test.\n");
        printf("9.)  Exit\n");
      }
    }

/************ Process Input*/

    void processSelection(int menuSelection)
    {
    int returnValue;

      printf("Processing Selection:  %d\n", menuSelection);

      switch (menuSelection)
      {
        case 1:
          returnValue = parseInputToInt();
          pulseWidth_TOC2 = returnValue;
          printf("Servo updated: pulseWidth = %d\n", pulseWidth_TOC2);
          break;
        case 2:
          printf("Enter servo speed:\n");
          returnValue = parseInputToInt();
          pulseWidth_TOC2 = returnValue;
          printf("Servo updated: pulseWidth = %d\n", pulseWidth_TOC2);
        case 3:
          asm("cli");
          break;
        case 4:
          asm("sei");
          break;
        case 5:
          {
```

```
          returnValue = readAD();
                                    if (returnValue)
                                    printf("A/D read successful\n");
         }
         break;

       case 6:
         FSRloop();
         break;

       case 7:
         while(1){};
         break;

       default:
         printf("Program terminated\n");
         break;
     }
  }

//********** read A/D
int readAD(void)
  {
       printf("Reading A/D:\n");

     ADCTL = 0x14;
     while ( (ADCTL & 0x80) == 0 );
     printf("A/D 4: %d \n", ADR1);
     return 1;
  }

//*********** FSR controlled loop
void FSRloop(void)
```

```
  {
  double servoSetNew = 0;
  double servoSetOld = 0;
  while(speed > 2050)
  {

    int ADvalue = 0;
    //printf("FSRloop:\n");

    ADCTL = 0x14;
    while ( (ADCTL & 0x80) == 0 );
    //printf("A/D 4: %d \n", ADR1);

    ADvalue = ADR1;
    servoSetNew = ((ADvalue*300)/4);
    servoSetNew = ((servoSetOld * 14) + servoSetNew)/15;
    servoSetOld = servoSetNew;

    //printf("servoSet: %d\n", servoSetNew);
    pulseWidth_TOC2 = (int)servoSetNew;
  }
  }


//*********** Read keyboard input

int parseInputToInt(void)
  {
    int input = 0;
    int n = 0;

    printf("Getting Int Input....\n");
    input = ((int)getchar() - 0x30)*10000;
    printf("%d ",input);
    input = input + ((int)getchar() - 0x30)*1000;
    printf("%d ",input);
    input = input + ((int)getchar() - 0x30)*100;
    printf("%d ",input);
    input = input + ((int)getchar() - 0x30)*10;
    printf("%d ",input);
    input = input + ((int)getchar() - 0x30);
    printf("%d \n",input);

    return input;
  }

/************** Interrupt Routines */

  void isrTOC2()
  {

    if (count>90)
    {
    int temp = period - pulseWidth_TOC2;





    if (totalPulses_TOC2 < 10)
    {
      //printf("In ISR_TOC2\n");
      totalPulses_TOC2 += 1;
    }
```

```
      if (pulseOn_TOC2)
      {
        TOC2 = TOC2 + temp;
        SET_BIT(TCTL1, 0xC0);
        pulseOn_TOC2 = 0;
      }
      else
      {
        TOC2 = TOC2 + pulseWidth_TOC2;
        SET_BIT(TCTL1, 0x80);
        CLEAR_BIT(TCTL1, 0x40);
        pulseOn_TOC2 = 1;
      }


      CLEAR_FLAG(TFLG1,0x40);
    }

//*******IC1 service routine
void isrTIC1(void)
{

CLEAR_BIT(TMSK1, 0x4);



//if ((TCNT - TIC1)>0)
//printf(">0 read\n");

currentTime = TCNT;



if (TFLG2 >=  128)
{
        CLEAR_FLAG(TFLG2, 0x80); //clear TCNT O.F. flag

}
//printf("currentTime= %d\n", currentTime);

if (currentTime < 0)
currentTime = (currentTime ^ 0xFFFF) + 1;

//printf("currentTime= %d\n", currentTime);
//printf("prevTime= %d\n", prevTime);

if (TCNT >=  128)
        CLEAR_FLAG(TFLG2, 0x80); //clear TCNT O.F. flag

if ((currentTime - prevTime)<0)
{
        pulseTime = ((currentTime - prevTime)*32/0xFFFF +32);

}
else
        pulseTime = ((currentTime - prevTime)/0xFFFF)*32;



speed = 3600 - pulsetime;

//printf("pulseTime= %d\n", pulseTime);

//printf("Speed= %d\n", speed);

prevTime = currentTime;



//printf("pulse!!\n");
```

```
//printf("%d\n", TFLG1);

CLEAR_FLAG(TFLG1, 0x4);  //clear interrupt flag

//printf("%d\n", TFLG1);

}//End isrTIC1

/************ servo init */

  void initializeServos()
  {
    printf("Initializing Servos:  Begin.\n");
    *((void (**)())0xffe6)  = isrTOC2;          /*  Set Interrupt Vector:  TOC2 */

    CLEAR_BIT(OC1M, 0xff);        /*  Disable OC1 Access */
    SET_BIT(TMSK1, 0x40);                   /*  Enable TOC2 Interrupt */
    CLEAR_FLAG(TFLG1, 0x40);  /*  Clear Possible Flag:  TOC2 */

    printf("Initializing Servos:  Complete\n");
  }

//******IC1 init
void init_ic1(void)
{
        printf("Initializing IC1.\n");
        INTR_OFF();

   *((void (**)())0xffee)  = isrTIC1;   /*  Set Interrupt Vector:  TIC1 */

        SET_BIT(TCTL2, 0x30);

        SET_BIT(TMSK1, 0x4);// enable IC1 interrupt
        CLEAR_BIT(TCTL1, 0x20); // clear IC1 flag

        INTR_ON();
        printf("Finished initializing IC1.\n");

}//End init_ic1
```