OSU Mini Baja
Electronic Traction Control System
(ETC System)

Senior Design Project
Electrical and Computer Engineering

Group 6 & 14
04/29/2007
ECE 44x

Table of Contents

Introduction

The Mini Baja Car competition is an intercollegiate contest to design, build, and race a Baja vehicle against teams from other colleges across the nation. The object of the competition is to provide a real-world project for engineering students to design and manufacture a new product. Students work together in all phases of the project, not only to design and build the system, but also to test and race the end product.

Mini Baja at OSU

OSU enters a new Baja car into the competition every year. In 2006 the OSU team won first place overall at the SAE Mini Baja West competition in Portland, beating 84 other teams from 22 countries, including teams from top U.S. engineering schools. The OSU Baja car then took the top honor again at the SAE Mini Baja Midwest competition held in Elkhorn, Wisconsin, where 141 teams competed, some from as far away as Brazil and South Africa.

Electronic Traction Control System

In 2001 the OSU Mini Baja Car team decided to implement an Electronic Traction Control System (ETC) on the vehicle to improve the performance in race competitions. However, due to poor design and hardware failure the past ETC systems on the car had seen little success. In 2005, the Mini Baja Car entered the race with the ETC turned off as the driver did not have any confidence in the system. This year we successfully built a functional prototype several months before the competition.

Our Mission

Our primary goal for this year was to create a functional ETC system well ahead of competition so we have time to test and fine tune the system as well as gaining driver confidence on the system. We also wanted to improve the response time from previous designs since past designs all had slow response times of half a second. We also wanted to implement and redesign the power system to be less complicated to ease implementation.

System Overview

Our ETC consists of 4 systems:

- Power:
  - High Voltage and Low Voltage
- Computation
  - Atmel Atmega 128 Micro-controller
- Sensor
  - Hall-Effect, Pressure, Steering Wheel, Current, Temperature
- Actuation
  - 12V DC Motor and sector gear

Hardware Design Introduction

The hardware has been redesigned from previous years to improve and ensure performance of the final product. The system consists of the following:

- 12V battery
- 5V battery
- Sensors (Pressure, Temperature, Hall-Effect, Current, Steering Wheel)
- Atmel Atmega 128 Micro-Controller
- Ultra Capacitor (2 Farads)
- 12V DC Motor

The 12V battery feeds all of the electrical components used on the ETC System and the 5V battery is for to power the micro-controller. Sensors are used to monitor various components of the car and to signal the micro-controller. Once the micro-controller is done with the computation, the 12V DC motor allows us for a quick and powerful actuation.

Hardware Overview

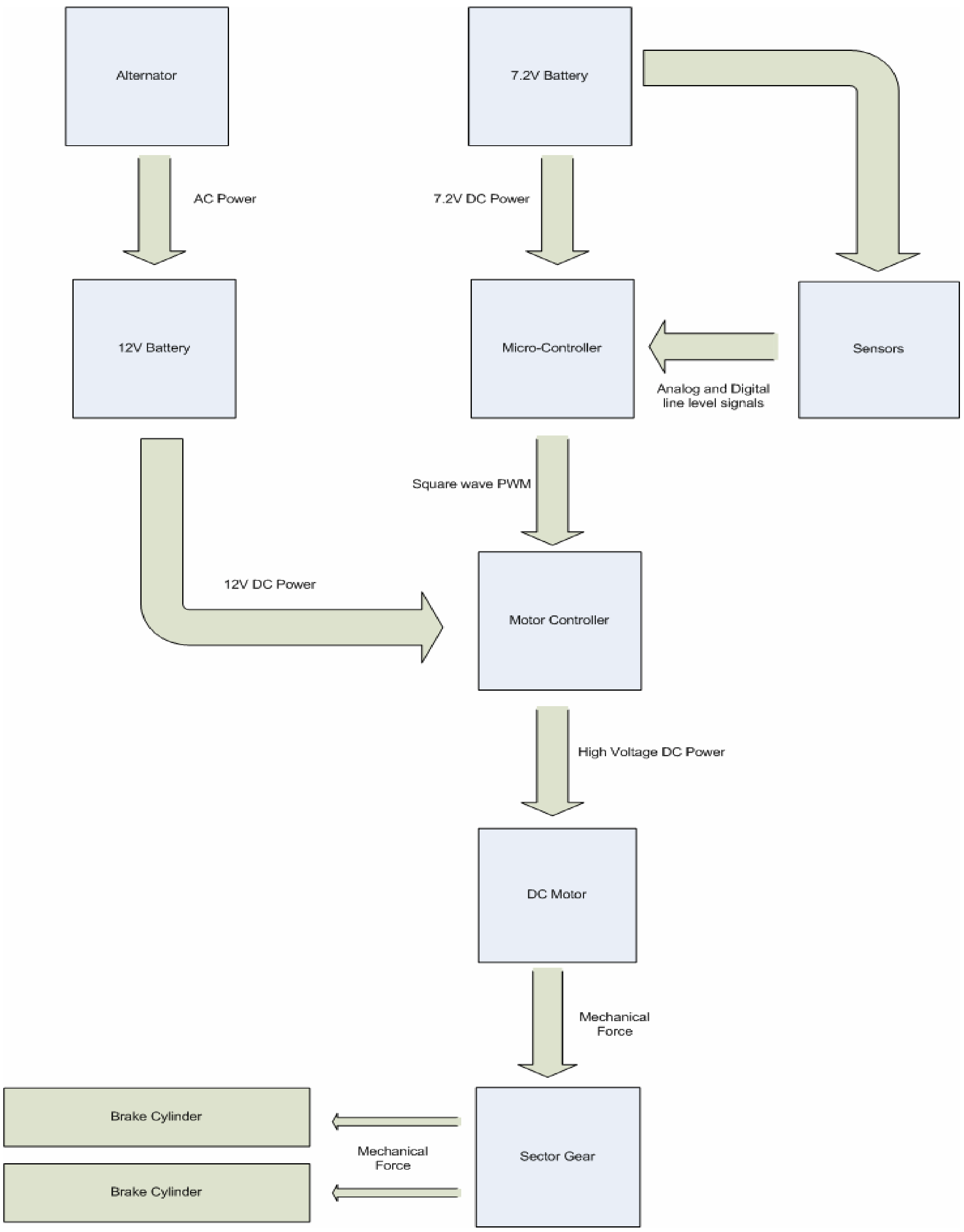

Figure 1

<u>Hardware Documentation for Sensors</u>

Hall-Effect Sensor

Functionality:
The Hall-Effect sensor is a digital sensor that monitors the speed of each wheel. This sensor is connected to the external interrupt pin in the microcontroller. It triggers the interrupt each time the sensor rolls over a hole on the break rotor to calculate the differential wheel speed of the 2 holes.

Output: 0V when wheels are stationary and 5V when wheels are spinning

Temperature Sensor

Functionality:
The temperature sensor monitors the temperature of the actuation box.

Output: The output ranges from 0-5V depending on the temperature.

Steering wheel sensor

Functionality:
The steering wheel sensor on the Baja car is used for monitoring the angle of the steering wheel. A potentiometer is used for this purpose in our design. Utilizing the output from the sensor to determine what the current wheel speed ratio of the car should be.

Output: The output ranges from 0-5V. The sensor is set up to be outputting 2.5V when the car is centered, 0V to one side, and 5V to the other.

Current Sensor

Functionality:
The current sensor is an analog sensor that monitors current coming out from the alternator and the current going into the motor. It ensures that the current does not over shoot and damage the system.

Output: The output ranges from 0-5V depending on current.

Pressure Sensor

Functionality:
The pressure sensor monitors the pressure outputted by the actuator. The actuator is able to output pressures from 250-1000psi, and the sensor is capable of monitoring up to 1000psi.

Output: The output ranges from 0.5-4.5V depending on outputted pressure.

Hardware Documentation for Circuit Design

Reducing noise for DC power line

The noise on the DC power line is reduced by putting two capacitors in parallel, one electrolytic and one ceramic. Putting 2 different types of capacitors together will generate a better filter. Electrolytic capacitors will allow us to have a bigger capacitance to filter out the low frequency noise and ceramic capacitors will filter out the high frequency noise.

Ground

There are two different grounds for the car, chassis ground and micro-controller ground. These two grounds are connected with a 100 Ohm resistor shown in figure (3). By connecting these two grounds together, it allows the microcontroller to have the same reference voltage as the chassis.

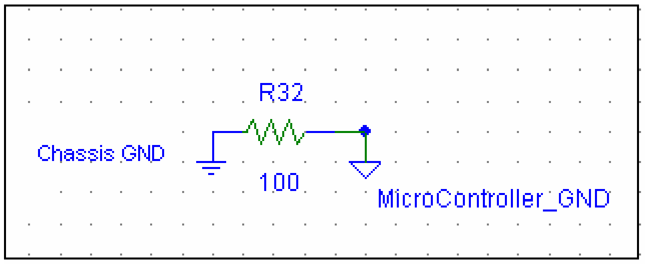

Figure 2

Hall Effect Sensors

JP 4 is directly connected to the Hall Effect Sensor. Pin 3 and 4 are the outputs from the Hall Effect Sensor and it is connected to the external interrupt pins (PORT E pin 4 and 5) of the microcontroller. Inserting a pull up resistor R1 and R2 on the signal line is required to activate the Hall Effect Sensor.
In order to reduce the noise, we put a low pass filter on the signal lines. The low pass filter is a simple RC circuit consisting of R3, R4, C2 and C3, which gives the cutoff frequency at about 20 kHz. The following equation is used to calculate the cutoff frequency.

$$f_c = \frac{1}{2\pi RC} = \frac{1}{2\pi \times 82 \times 0.1 \times 10^{-6}} \approx 20khz$$

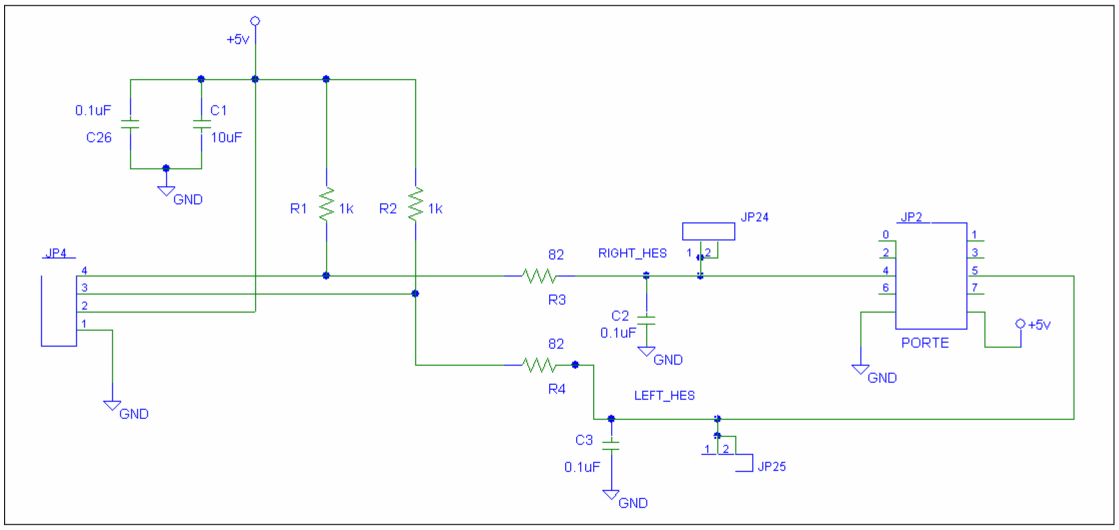

Figure 3

To filter out the noise on the DC power line, C26 and C1 are applied parallel to the line.

Steering Wheel Sensor

The Steering wheel sensor is constructed with a potentiometer which is connected to JP5. A capacitor C4 is applied across pin 1 and 2 to stabilize the DC power line. The signal line is connected to PORT F pin 0 of the microcontroller. A low pass filter is also applied to the signal line consisting R5 and C5, which gives the cutoff frequency at about 10Hz. The following equation is used to calculate the cutoff frequency.

$$f_c = \frac{1}{2\pi RC} = \frac{1}{2\pi \times 160000 \times 0.1 \times 10^{-6}} \approx 10hz$$

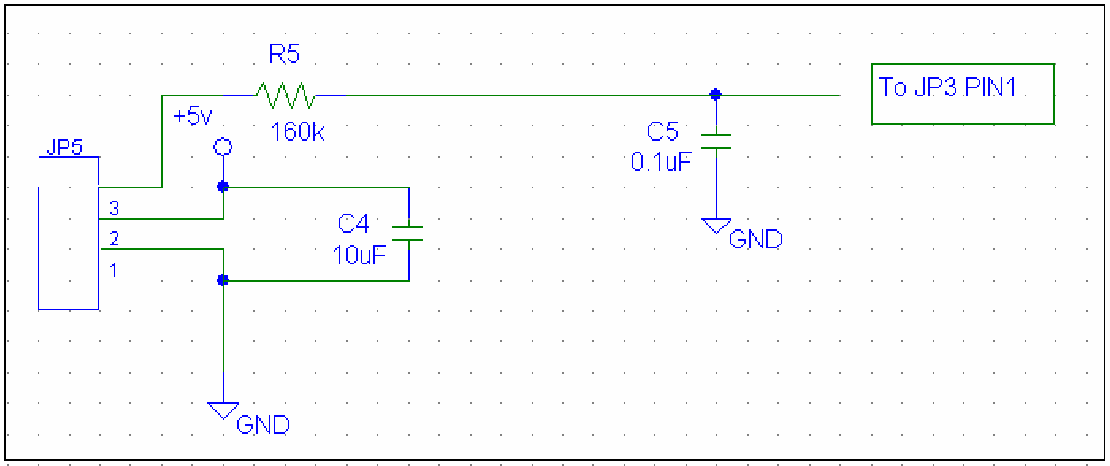

Figure 4

Current Sensor

There are 3 current sensors which monitor the currents coming out from the regulator, 12 Volt battery, and the total current of the speed controller. The circuitries for all of the current sensors are identical and it is shown in figure (6). The outputs of the current sensor are connected to PORT E pin 2, 3 and 5, which are also the ADC port of the microcontroller. A low pass filter consisting of R8 and C9 is applied to the signal line of the current censors. The cutoff frequency can be calculated from the following equation.

$$f_c = \frac{1}{2\pi RC} = \frac{1}{2\pi \times 150 \times 0.1 \times 10^{-6}} \approx 10khz$$

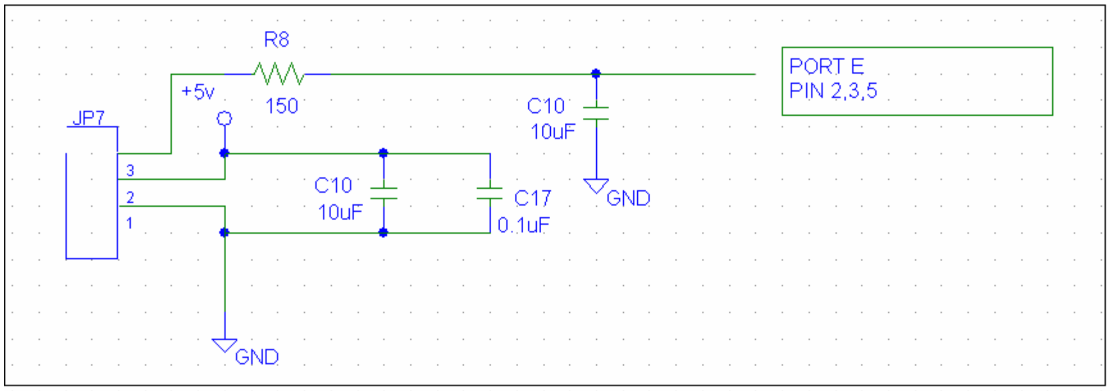

Figure 5

Pressure Sensor

There are two pressure sensors which are monitors the pressure of the fluids for the braking system. The circuitries for both the pressure sensors are identical and it is shown in figure (7). The outputs of the pressure sensor are connected to PORT E pin 6 and 7, which are also the ADC port of the microcontroller. Low pass filters consisting of R6 C8 R7 and C7 are also applied to the signal lines of the pressure sensor. The cutoff frequency can be calculated from using the following equation.

$$f_c = \frac{1}{2\pi RC} = \frac{1}{2\pi \times 1500 \times 0.1 \times 10^{-6}} \approx 1khz$$

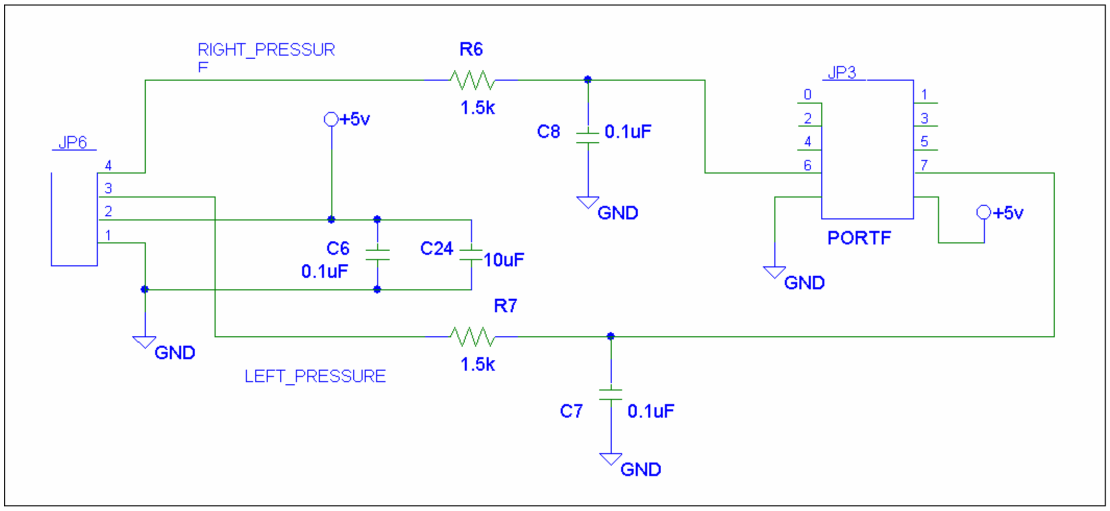

Figure 6

Voltage Sensor

The voltage sensor is made by a simple voltage divider consisting of R22, R23 and R24. The input will then step down to 1/3 of the original voltage. The input is connected to PORT E pin 5. A low pass filter is also applied to the signal line.

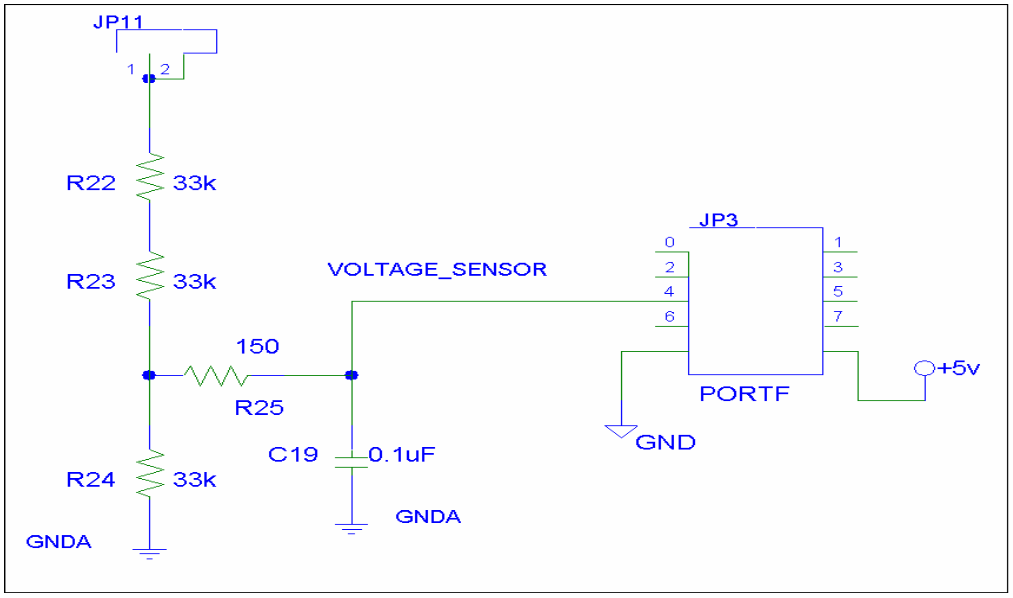

Figure 7

Temperature Sensor

Temperature Sensors are directly connected to JP8. In order to have the sensor working properly, a pull up resistor R10 is required. The signal line from the sensor is feeding to PORT F pin1 of the micro-controller. A low pass filter consisting of R11 and C12 is also applied on the signal line to reduce the noise. The following equation is used to calculate the cutoff frequency.

$$f_c = \frac{1}{2\pi RC} = \frac{1}{2\pi \times 100000 \times 0.1 \times 10^{-6}} \approx 16hz$$

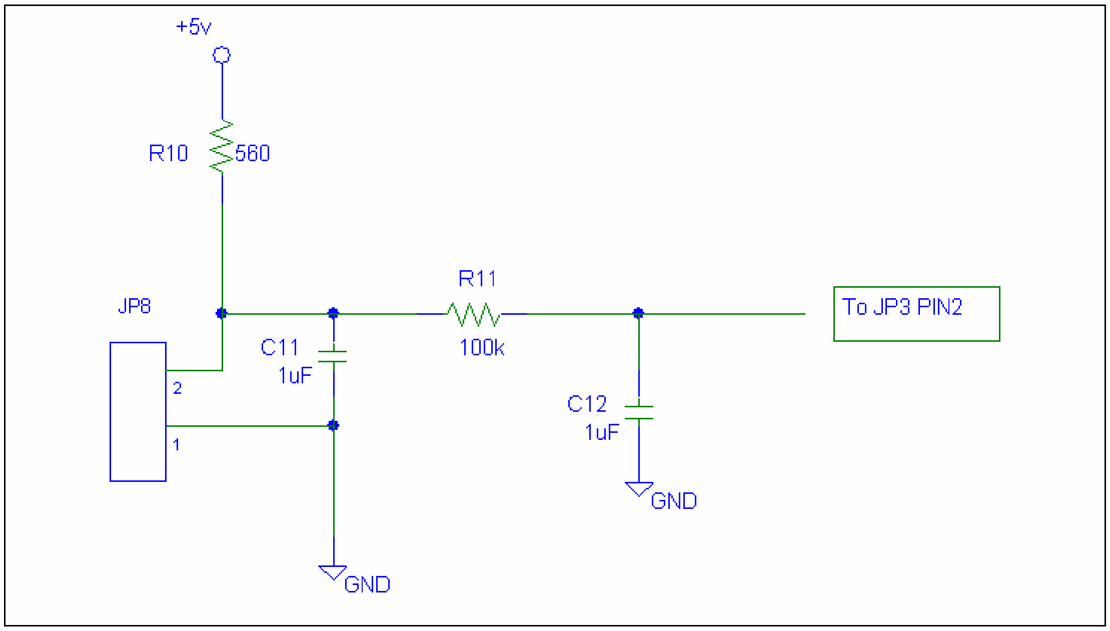

Figure 8

LED Indicator

An LED indicator is also implemented in the ETC system. This is used to notify the driver when the actuation system kicks in and applies braking to the wheel. To achieve this purpose, the software system sets a flag when a loss of traction is detected. The microcontroller then sets PIN0 and PIN1 on PORTB to ON or OFF according to the flag. The 5V DC voltage is then stepped down using a voltage divider to turn on the NPN transistor. The LED's are connected between 5V and the pins on JP13.

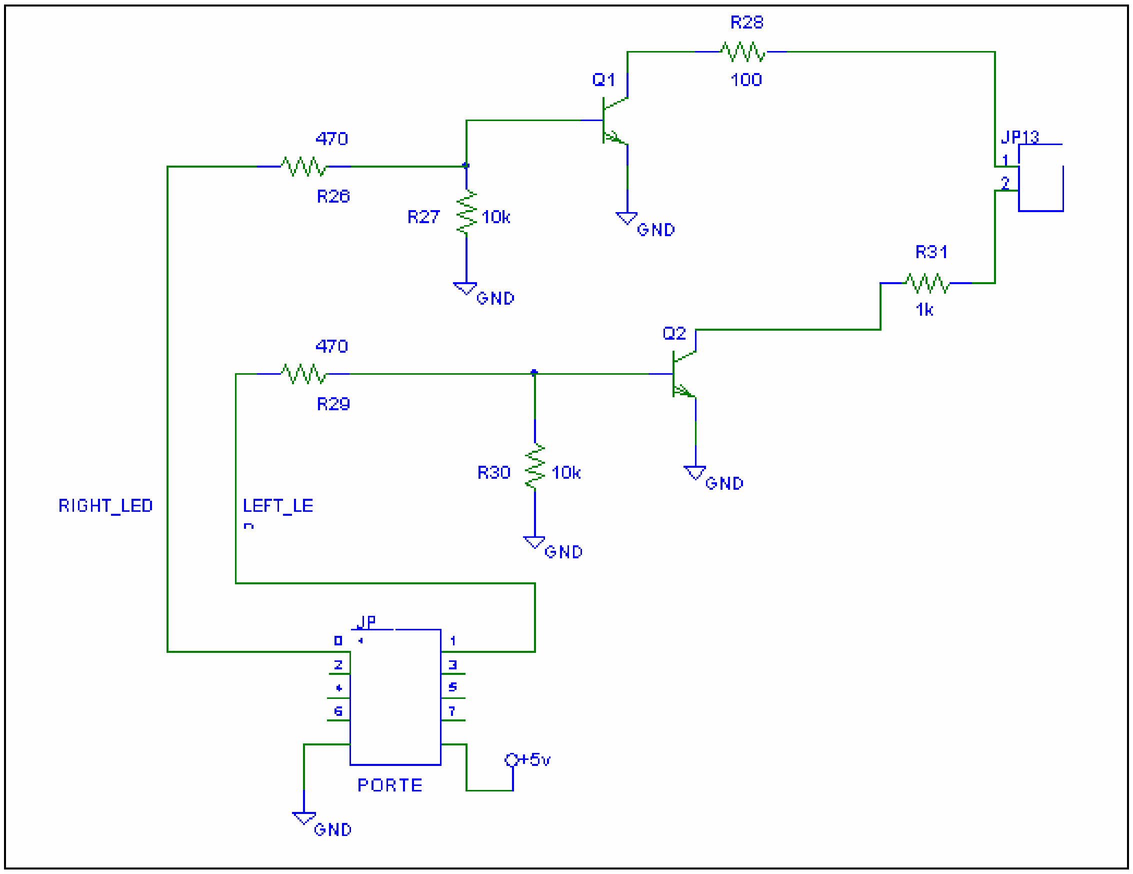

Figure 9

Software Documentation for Initialization

Introduction

The global variables header file contains all of the global structures and variables used by the ETC system. It also has a couple of flags, the first of which signals when new data is available, and the second one is used for debugging purposes.

Structures

The structures available include analog sensor, digital sensor, car, 40-bit counter, and 40-bit timestamp.
An analog sensor structure contains data fields that hold the channel the sensor is on, raw data read from the sensor, a meaningful value that data is converted into, as well as a multiplier which is used to avoid floating point calculations within the sensor drivers. Finally, there is a flag which has various uses throughout the sensor drivers.

The digital sensor structure is set up for the Hall Effect sensors used on the car, it contains data fields for the two different clocks that the sensor generates in order to determine the wheel speed, another data field. The multiplier and flag fields are used in the same manner as in the analog sensor structure.

The car structure holds fields for the cars speed, the actual wheel speed ratio, the desired wheel speed ratio, and variables for the left and right wheels to indicate the direction of their motion.

The 40-bit counter structure holds two fields, a high and low value. Whenever the low value in the counter overflows, the high value is incremented so the actual value in the counter is not lost, which extends the time before the whole counter overflows drastically.

The 40-bit counter timestamp structure functions in the same way the 40-bit counter, however it allows for even more precision with the inclusion of a middle value.

Variables

The global variables portion of the file consists of a list of the structures that need to be initialized. It also includes a variable which holds the 40-bit counter value, this is necessary so that an ISR can read its value.

Software Documentation for Main

Introduction

Main implements the execution routine for the computation system by calling other sub functions. It includes *main*, *UpdateSensors* and *CalculateWSR* methods.

Structure

This is the main method of the ETC computation system. There are no input parameters. It initializes its local variables and utilizes other sub functions to create the execution routine for the ETC system.

Execution

Once the main function has started executing, it begins by calling the init functions to initialize the different components of the system. Components such as the Watchdog timer, SPI ports, LCD display, the serial port interface, the 40-bit timer counter, pulse-width-modulation, the ADC MUX and the Hall-effect sensors have to be initialized to set the system to a known state.

After all the components have been initialized, the main loop of the function will start executing. The main loop slits into two parts depending if the serial interface is in use or not, because inputting commands using the serial interface will freeze data processing. Therefore separating the loops is important. The main process of the ETC is then begun by calling the *UpdateSensors* method to process the information from the sensors into more meaningful data, and put them into the related data structures. After that the program will capture the current timestamp, and by comparing it to the previous timestamp, it will calculate the delta value for the SISO PD algorithm. From the SISO PD algorithm, the process will be able to calculate the appropriate pressure that's needed to for the brakeline to correct the traction problem of the car.

As mentioned above, the main loop operates differently when the serial interface is active. The reason is because the serial interface involves the USART applications. The loop for the serial interface starts by setting up the user's manual. Short descriptions are included to explain particular commands to the user. Through the serial interface, user can test the actuation system by changing the pwm output to the system. Also, while doing bench testing, the user will be able to monitor the values of all the sensors on the screen.

void UpdateSensors(void);

There are no inputs for UpdateSensors function. It simply calls the different sensor driver functions to interpret and update the data from the sensors. The method to calculate the desire and actual wheel speed ratio are also being called in this method to update their values.

void CalculateWSR(void);
There are no inputs for the CalculateWSR function. It uses to the data from the Hall-effect sensors to calculate the current Wheel-Speed-Ratio of the car. It compares the speed of the two wheels, and then it will apply the appropriate formula to calculate the current wheel-speed-ratio depends on which wheel is faster. The formula for calculating the speed ratio is just using the faster wheel speed divide by the slower wheel speed. It then multiplies by 100 to create more resolution without dealing with the floating point numbers.

applications.c
Applications.c implements two essential functions for the serial interface, the SendVars and SendBinaryData. These functions simply utilize the USART functions to transmit the current variables from the Microcontroller memory to the serial port and display them on the monitor.

void SendVars(void);
There are no input parameters to this function. It uses the function USART0_TransmitString to send a display of all the current variables in the Microcontroller memory to the serial port in ASCII encoded format for display.

void SendBinaryData(uint16_t delta_ms, int siso_pd_return);
uint16_t delta_ms: Delta value that's calculated from using the timestamps, it's one of the input to the SISO PD algorithm.

int siso_pd_return:
The integer value returned from the SISO PD algorithm. This function utilizes the USART0_Transmit function to transmit a snapshot of the local variables in a binary format to the serial port for display on the monitor.

Software Documentation for Drivers

Hall Effect Sensor

Structure

int8_t debounce(int pin);
int pin: uC's external interrupt channels (PORT E PIN 4 & 5) to debounce.
Return value: if it returns zero (pin value), then this interrupt was triggered by noise.

void hallEffectSensor_init(void);
Initialization, each external interrupt channel configured for rising edge detection.

void HES_driver(async_sensor *this_sensor);
async_sensor *this_sensor: async_sensor structure pointer

Interrupt Service Routines
ISR(SIG_INTERRUPT4);
ISR(SIG_INTERRUPT5);

Execution

The hallEffectSensor_init function is called once during startup. The right HES sensor is attached to Port E, pin 4 (PE4) which is also External Interrupt channel 4 (INT4). The left HES sensor is attached to Port E, pin 5 (PE5) which is also External Interrupt channel 5 (INT5). Both channels 4 and 5 are triggered by rising edge detection. Then the HES_driver detects if new data is available. If new data is not available it will exit the driver, otherwise it will clear the flag bit. Updating the HES is dependent of wheel speed, in this case the uC may read in new ADC data before the HES data is updated, the uC will use the old HES data and move on to the control section.
For hes_driver_temp variable we use for the brake rotor holes=36, r=12, miles per hole=0.0000330563, and a clock speed=16Mhz. The multiplier for the output variable is fixed at 2.

The Interrupt Service Routines are configured so that the processor can handle any interrupt requests without interrupting any other time critical code. The ISR top half debounces the sensors by calling the debounce function to eliminate spurious readings from noise. After that it records time data, calculating the change in time between readings.

The ISR bottom half -- the non time-critical portion -- computes the actual wheel speed and wheel speed ratio (WSR).

Pressure Sensor

Structure

void left_pressure_driver(analog_sensor *this_sensor);
void right_pressure_driver(analog_sensor *this_sensor);
analog_sensor *this_sensor: analog_sensor structure pointer

Execution

The right pressure sensor is attached to Port F, pin 6 (PF6), which is also ADC channel 6 (ADC6). The left pressure sensor is attached to Port F, pin 7 (PF7), which is also ADC channel 7 (ADC7).

The left_pressure_driver and right_pressure_driver detects if new data is available, if new data is not available it will exit the driver, otherwise it will clear the flag bit.

The voltage range of the ETC pressure sensors is between .5V and 4.5V and representing 0 to 600 psi linear relationship between voltage and pressure, equation used to convert raw ADC value to meaningful value: y=mx+b , where m is the slope of the line m = (600-0)/(4.5V-.5V). Using 10-bit number represent 4.5volt = 922 0.5volt =102, m = (600-0)/(922-102) = .732 and the offset b = -75, we set actual_value. The multiplier variable is then set to -1 here.

Current Sensor

Structure

void current_driver(analog_sensor *this_sensor);
void voltage_driver(analog_sensor *this_sensor);
void bat_reg_driver(analog_sensor *this_sensor);
analog_sensor *this_sensor: analog_sensor structure pointer

Execution

The motor current sensor is attached to Port F, pin 2 (PF2), which is also ADC channel 2 (ADC2). Each of the drivers detect if new data is available, if new data is not available it will exit the driver, otherwise it will clear the flag bit.

The Voltage range will be between 0V and 2.5V and representing 0 to 90 Amps. There is a linear relationship between voltage and current, equation used to convert the
raw ADC value to meaningful value y=mx+b , where m is the slope of the line m = (90-0)/(2.5V-0V). Using 10-bit number represent 2.5volt=512 0volt=0, m = (90-0)/(512-0) =

.17578 and the offset b = 0 since we don’t want to use floating numbers, we multiply .17578 by 10^3 to arrive at the actual_value. The multiplier variable is then set to -3.

Temperature Sensor

Structure

void temp_driver(analog_sensor *this_sensor);
analog_sensor *this_sensor: analog_sensor structure pointer

Execution

The temperature sensor is attached to Port F, pin 1 (PF1), which is also ADC channel 1 (ADC1).

The temp_driver detects if new data is available, if new data is not available it will exit the driver, otherwise it will clear the flag bit.

The voltage range of the thermal sensors is from 0V to 2.5V. m = (125)/(819-563) = .49 and the offset b = -m*125 is used to calculate actual_value.

No multiplier in use here since the max value is well beyond the scope of an 8 bit datatype.

Steering Sensor

Structure

void steering_driver(analog_sensor *this_sensor);
analog_sensor *this_sensor: analog_sensor structure pointer

Execution

The steering sensor is attached to Port F, pin 0 (PF0), which is also ADC channel 0 (ADC0).

The steering_driver detects if new data is available, if new data is not available it will exit the driver, otherwise it will clear the flag bit.
The voltage range of the steering sensor is 0 to 5 volts, m = (125)/(819-563) = .49 and the offset b = -m*125

The driver function shifts the 10-bit ADC value into an 8-bit local variable, and computes DWSR based upon the given equation.
Every sensor state would redundantly put a value of -2 into the multiplier variable, so it just saves code to do it here.

The if-then statements serve as a lookup table to calculate the DWSR based upon the ADC table due to the fact that the original function calculates based upon a sine function.

Motor Speed driver

<u>Structure</u>

void pwm_init(void);
void pwm_setperiod(uint16_t period)
void pwm_set(uint8_t mode);
mode: 1 => forward; 2 => stop; 3 =>reverse

<u>Execution</u>

The pwm_init function is called once. It set up counter 3 (16 bit) to act as a dual channel PWM generator.
Since we want OC3A to be set on reaching TOP, we set on reaching compare match, use ICR3 as TOP and have pre-scale of 8. So we set OC3A at BOTTOM (inverting mode), mode 14 (fast PWM, clear TCNT3 on match ICR3) and then we generate the desire period.

The pwd_set receives the mode that is an unsigned 8 bits integer, being 1 for forward, 2 for stop and 3 for reverse to get the appropriate OCR3A.
We also set the period using the function pwm_setperiod. The purpose of pwm_setperiod is to set the duty cycle of timer-counter1 by setting the appropriate registers on the controller.

Software Documentation for ADC MUX

Introduction

Mux.c is a simple code to activate the built in ADC in AtMega 128 micro-controller board. It scans the analog sensors one at a time, then it converts the analog data into a 10 bit digital values.

ADC Connection

The ADC is located at port F, and a list of the sensor connections are shown below.
Pin 0 – Steering wheel sensor
Pin 1 – Temperature sensor
Pin 2 – Current sensor for DC motor
Pin 4 – Voltage sensor
Pin 5 – Current sensor for alternator
Pin 6 – Right pressure
Pin 7 – Left pressure
The multiplexer reference voltage is AVCC with external capacitor at AREF pin

ADC Multiplexer Selection Register – ADMUX

-Set zero to REFS1 and REFS0 to turn off the internal voltage reference and allows the reference voltage to be AREF pin
-Set one to ADLAR to allow us to have right adjusted results.

ADC Control and Status Register A – ADCSRA

-Set one to bit 7 to enable ADC
-Set one to bit 6 to start a new conversion
-Set one to bit 3 to enable ADC enable
-Set pre-scalar to 128.

ADC Execution/ISR

The compiler will jump to the interrupt after the conversion is done. So we have to start the first conversion and execute the ADC interrupt, and it's done in the function mux_init. Notice that the program will start another conversion at the end of the interrupt, so the subroutine will loop forever and automatically scan all channels and convert analog inputs to digital values, and the results are right adjusted.
Jumping between channels is simply done by a switch and case statement.

ADC Timing

Since the compiler will only go into the interrupt when the conversion is done, it will never change the channel when the ADC is converting, so the multiplexer will never fail. To get the best resolution for a 10 bit output, the ADC clock frequency should be less then 200 kHz. We used the pre-scalar of 128, which gives a clock frequency of 125 kHz which will give us a very fine resolution.

<u>**Software Documentation for SISO PD algorithm**</u>

<u>Introduction</u>

The SISO PD algorithm is a relatively simple algorithm which compares the current wheel speed ratio to the desired wheel speed ratio to the actual individual wheel speeds, and then generates a command through a function comprised of proportional and linear components.

<u>Structure</u>

int siso_pd(uint16_t WSR_des, uint16_t time);

uint16_t WSR_des: The desired wheel speed ratio expressed as a fraction of right wheel speed over left, divided by 100.
uint16_t time: The number of milliseconds that have passed since last execution. Pass zero upon the first run to avoid complications.

Return value: This function returns -1 to indicate an error has occurred during execution. If no problems have happened, a value of 0 will be returned.

<u>Execution</u>

Once the function has been invoked, it begins by checking for safety of the system. If the temperature sensor indicates that the motor is overheating, or if the current sensor indicates that the motor is drawing too much power, the motor is set to turn off and the function returns with an error value.

Next, the function will check the border condition of whether the wheels are moving fast enough beyond a set threshold (currently 2.5 mph). If the motors are below this speed, the function will return with no error. After this step, the desired wheel speed ratio (DWSR) and actual wheel speed ratios (WSR) are compared. If they are less than 10% apart, then the algorithm returns with no error.

After that, the delta WSR is calculated by taking the difference of the actual WSR and the DWSR. This error is capped at 125% to prevent strange errors from occurring. This error is then multiplied by two different values to form the proportional and differential components of the algorithm.

These resultant components are added together, then capped at 300. Finally, the value is added or subtracted based upon its sign to the designated edge of the motor controller's linear region of the pwm signal. This pwm signal is applied to the motor controller. The routine then returns without error.

Laptop Interface

The laptop can interact in two different ways with the ETC microcontroller:

- Manual Control – This is where you can manually control the actuator and is mostly used for debugging the system while in the field.
- Data Logging – This is uploading sensor data to Matlab while driving.

The different modes are selected by changing pre-processor vaiable DATA_LOGGING in the file glob_var.h. Follow these instructions to reprogram the microcontroller with the desired value:

1. Open the file glob_var.h with a text editor such as notepad. For Manual Control, set DATA_LOGGING to 0, for Data Logging, set it to 1.
2. Click Start->Run type in 'command' without the quotes. Click OK.
3. In the command window, navigate to the directory where glob_var.h is stored.
4. Type 'make clean', the file main.hex should be deleted.
5. Type 'make all', the file main.hex should be recreated
6. Open Pony-Prog by clicking Start->Programs->PonyProg->PonyProg2000.
7. Once PonyProg loads, click OK on the welcome screen, then open the file main.hex.
8. Connect the programming dongle between the ETC AVR board and laptop.
9. Turn on the ETC AVR board.
10. Click Command->Program and wait for the window that says 'Program Successful'

<u>Manual Control</u>
To acess the manual control, open the hyperterminal program avr.ht by clicking Start->Programs->Accessories->Communications->Hyperterminal->avr.ht. In case this file disspears, the communication settings are as follows:

Baud = 115200
Start = 1
Stop = 1
Parity = Odd
Flow Control = None

There are several available commands. To see a list of command, type 'help' at the command prompt. For details on each command, type 'help command' where 'command' is the command you desire more information on.

Data Logging

In order to properly perform data logging, do the following:

1. Make sure the AVR is turned off
2. Make sure the laptop is in the padded box on the car with the serial cable connected between the laptop and the AVR box.
3. Open Matlab on the Laptop
4. In Matlab, open the file data_logging.m and run it (Press F5).
5. Close the laptop and latch the box closed.
6. Start the car and then turn on the AVR box.
7. Drive around
8. When done, turn off the AVR and the car.
9. Open the laptop and move your finger on the mouse pad to make the screen turn back on.
10. In the Matlab command window, type: 'fclose(s); delete(s)'
11. In the Matlab variable window, right click the mouse and select 'Refresh'.
12. Right click on the variable 'data' and save it as a *.m file somewhere on the hard drive.
13. To look at the data, open the file 'convdata.m'. At the top, change the *.m file to be opened to the file you just saved.
14. Run convdata.m. It takes a few minutes to process.

Cost Analysis

Item	Part Name	Purchased	Fabricated	Vendor	Qty	Material Cost	Labor Cost	Extended Material Cost	Labor Cost	ExtendedTotal
1	Master Cylinder, ETC 0.5"	X		Polaris Ind.	2	$110.00	$0.00	$220.00	$0.00	$220.00
2	Brake Line	X		Summit Racing	2	$19.88	$0.00	$39.76	$0.00	$39.76
3	MicroController	X		OSU Marketplace	1	$109.95		$109.95	$0.00	$109.95
4	Motor Driver	X		Vantec	1	$159.95		$159.95	$0.00	$159.95
5	Ultra Capacitor	X		Massive Audio Inc	1	$53.95		$53.95	$0.00	$53.95
6	Battery 12V AGM	X		McMastercar	1	$38.15		$38.15	$0.00	$38.15
7	Pressure Sensors	X		Measurement Specialist	2	$75.60		$151.20	$0.00	$151.20
8	Actuator Motor	X		Ryobi	1	$15.00		$15.00	$0.00	$15.00
9	Sector Gear ETC		X	Linn Gear	1	$3.88	$210.00	$3.88	$210.00	$213.88
10	Pinion Gear ETC		X	Linn Gear	1	$0.27	$70.00	$0.27	$70.00	$70.27
11	Rear Motor Mount		X		1	$0.61	$31.50	$0.61	$31.50	$32.11
12	Front Motor Mount		X		1	$31.50	$32.11	$31.50	$32.11	$63.61
13	Clutch Lock		X		1	$0.30	$35.00	$0.30	$35.00	$35.30
14	Bearing Housing		X		1	$0.43	$35.00	$0.43	$35.00	$35.43
15	Base ETC		X		1	$1.26	$17.50	$1.26	$17.50	$18.76
16	Master Cyliner mounts		X		1	$0.14	$35.00	$0.14	$35.00	$35.14
17	Actuator Housing		X		1	$1.20	$35.00	$1.20	$35.00	$36.20
18	Capacitor ETC	X		Massive Audio Inc.	1	$53.95		$53.95	$0.00	$53.95
19	Battery 12V ETC	X		McMasterCar	1	$38.15		$38.15	$0.00	$38.15
20	Pressure Sensors ETC	X		Measurement Specialist	2	$75.60		$151.20	$0.00	$151.20
21	Actuator Motor ETC	X		Ryobi	1	$15.00		$15.00	$0.00	$15.00
22	Temp Sensor ETC	X		DigiKey	1	$0.90		$0.90	$0.00	$0.90
23	Hall Effect Sensor ETC	X		DigiKey	2	$2.49		$4.98	$0.00	$4.98
24	Wire ETC	X		DigiKey	100	$0.21		$21.00	$0.00	$21.00
25	Kill Switch ETC	X		Radio Shack	1	$5.95		$5.95	$0.00	$5.95
26	Stearing Position Sensor ETC	X		Radio Shack	1	$1.30		$1.30	$0.00	$1.30
	Assembly Time (min)		30	Subsystem Assy Cost					$17.50	$17.50
	Totals							$1,119.98	$518.61	$1,621.09
								Total:		**$1,638.59**