



8-2000

The design and implementation of a dual hybrid electric vehicle control system

Matthew D. Smith

Follow this and additional works at: https://trace.tennessee.edu/utk_gradthes

Recommended Citation

Smith, Matthew D., "The design and implementation of a dual hybrid electric vehicle control system. " Master's Thesis, University of Tennessee, 2000.
https://trace.tennessee.edu/utk_gradthes/9501

This Thesis is brought to you for free and open access by the Graduate School at TRACE: Tennessee Research and Creative Exchange. It has been accepted for inclusion in Masters Theses by an authorized administrator of TRACE: Tennessee Research and Creative Exchange. For more information, please contact trace@utk.edu.

To the Graduate Council:

I am submitting herewith a thesis written by Matthew D. Smith entitled "The design and implementation of a dual hybrid electric vehicle control system." I have examined the final electronic copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Mechanical Engineering.

William R. Hamel, Major Professor

We have read this thesis and recommend its acceptance:

Jeffrey W. Hodgson, J. A. M. Boulet

Accepted for the Council:

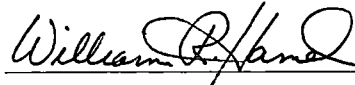
Carolyn R. Hodges

Vice Provost and Dean of the Graduate School

(Original signatures are on file with official student records.)

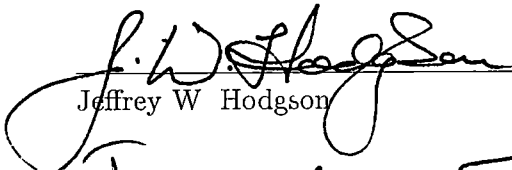
To the Graduate Council

I am submitting herewith a thesis written by Matthew D Smith entitled "The Design and Implementation of a Dual Hybrid Electric Vehicle Control System" I have examined the final copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Mechanical Engineering

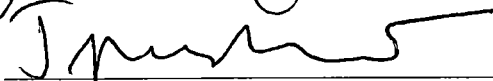


William R. Hamel, Major Professor

We have read this thesis
and recommend its acceptance



Jeffrey W. Hodgson



J. A. M. Boulet

Accepted for the Council



Associate Vice Chancellor and
Dean of The Graduate School

THE DESIGN AND IMPLEMENTATION OF A DUAL HYBRID
ELECTRIC VEHICLE CONTROL SYSTEM

A Thesis
Presented for the
Master of Science
Degree
The University of Tennessee, Knoxville

Matthew D Smith
August 2000

Copyright ©, 2000 by Matthew D. Smith
All rights reserved

*This thesis is dedicated to my parents
Eugene and Marianne Smith
Without their love and encouragement this would not be possible*

Acknowledgements

I would like to thank Dr William R. Hamel and Dr Jeffrey W Hodgson for giving me the chance to do what I enjoy the most. The advice and guidance they have given me over the last few years has been immeasurable. I also owe Dr. J A M Boulet a great deal of gratitude for the help he has provided in editing this thesis. Special thanks goes to Stephen Jesse and Craig Rutherford for the control algorithm development and the transmission construction. Without these two, this would be nothing but theoretical conjecture. I would also like to thank the FutureCar Team members, specifically Claudell Hatmaker, Veronika Gospodareva, Fred Mottley, Paul McCown, and Doug Ferguson. Without whom, the FutureCar project could not have happened. Also while at the University of Tennessee, I had the pleasure of working with those who call the Robotics and Electromechanical Laboratory home. Help from Dr Steven E Everett, Surya Singh, Mohammad Khalid, Sewoong Kim, Ge Zhang, Sam Richardson, and Yasunobu Isoda on general software issues and L^AT_EX typesetting is greatly appreciated.

--

Abstract

This research describes the development of a control system for a hybrid electric vehicle that uses the relatively novel configuration called the dual hybrid. The system is implemented in the UTK 1999 FutureCar Challenge entry, a Dodge Intrepid converted to dual hybrid operation with a student designed and constructed planetary based transmission. The control system includes models for the custom epicyclic transmission and battery pack state-of-charge. Control system implementation is done with the QNX real time operating system on a PC based microcomputer. Extensive discussion of the details of the software development is done with an emphasis on the reusability of the code. The control software includes modes for electric-only, hybrid electric, park, neutral, and reverse operation. While extensive testing has yet to be done, preliminary tests indicate that the control system provides a working code base that can be easily updated, modified, and reused.

Contents

1	Introduction	1
1.1	HEV Types	3
1.1.1	Series HEV	3
1.1.2	Parallel HEV	3
1.1.3	Dual Hybrid	5
1.2	FutureCar Competition	7
2	Design Phase	8
2.1	Hardware Interface	8
2.1.1	Drive Train Controllers	9
2.1.2	Battery Pack	10
2.1.3	Driver User Interface	11
2.2	System Controller	11
2.2.1	I/O requirements	11
2.2.2	Computational requirements	15
2.2.3	System Controller Hardware	15
2.3	Transmission Model	22
2.3.1	Drive Train Components	22
2.3.2	Torque and Speed Relations	24
2.4	Battery State of Charge Model	27
2.4.1	Initial Approach	27
2.4.2	Common Sense Approach	29
2.5	Software Concepts	39
2.5.1	POSIX and QNX	39
2.5.2	Multiple Tasks	42
2.5.3	Interprocess Communication	45
2.5.4	Real Time Features	47
3	Control Algorithms	49
3.1	ZEV Mode	49
3.2	Hybrid Mode Fundamentals	51
3.2.1	Throttle Position	51

3.2.2	Throttle-Up Procedure	52
3.2.3	The Generator	55
3.2.4	Electric Take-off	56
3.3	Hybrid Mode	56
3.3.1	Hybrid Regime Transition	60
3.3.2	Transition from ZEV to HEV Mode	61
4	Software Details	64
4.1	Overview	64
4.2	IPC Library libfclient	64
4.2.1	Shared Memory contents	65
4.2.2	libfclient functions	66
4.3	Task Primitives Library libftask	68
4.3.1	Time functions	69
4.3.2	Task management functions	70
4.4	Hardware Interface fcard	75
4.4.1	Features	76
4.4.2	Operation	77
4.5	Control Program fcar	81
4.5.1	Overview	81
4.5.2	Task details	82
4.6	Monitoring and Diagnostics	92
4.6.1	Shared Memory Monitor. mon	92
4.6.2	Data Trends Viewer trends	93
4.6.3	Data Logging flogger	93
5	Conclusions	96
5.1	Improvements	96
5.2	Future Work	99
	Bibliography	100
	Appendices	103
A	Basic Control Code Modification	104
B	Real time Trend Graph Viewing	108
C	Retrieving Logged Data	110
D	Control Code Listing	112
D.1	fcard Hardware Control Dæmon	112
D.1.1	Makefile	112
D.1.2	fcard.h	113

D 1 3	octagon_io h	114
D 1 4	octagon_io_map h	116
D 1 5	fcard c	118
D 1 6	parse_cmdline.c	121
D 1 7	daemon c	122
D 1 8	initshm c	124
D 1 9	fcard conf	125
D 1 10	read_conf.c	127
D 1 11	kwh_meter c	129
D 1 12	hardware_io c	131
D 1 13	octagon_init c	132
D 1 14	octagon_io c	133
D 2	fcar Main Control Program	140
D 2 1	Makefile	140
D 2 2	fcar h	140
D 2 3	fcar c	142
D 2 4	mode_select c	144
D 2 5	modes c	146
D 2 6	motor_eqn c	151
D 2 7	misc c	157
D 2 8	ice_ctrl c	159
D 2 9	soc c	161
D 2 10	throttle_ctrl c	164
E Support Libraries Code Listing		166
E 1	libfclient IPC Library	166
E 1 1	Makefile	166
E 1 2	fclient h	166
E 1 3	fcar_common h	167
E 1 4	fclient_create_shm c	170
E 1 5	fclient_delete_shm.c	171
E 1 6	fclient_open_shm c	172
E 1 7	s_die c	173
E 2	libftask Task Primitives Library	174
E 2 1	Makefile	174
E 2 2	ftask h	175
E 2 3	ftask_private h	176
E 2 4	timer_qnx_private h	176
E 2 5	trigger_private h	177
E 2 6	ftask c	177
E 2 7	ftask_time c	181
E 2 8	timer_qnx c	182
E 2 9	trigger_pipe c	183

E 2.10	trigger_qnx.c	..	185
E 2.11	trigger_sig.c	.	186
F	User Interface Code Listing		188
F 1	vfd Vacuum Fluorescent Display program	.	188
F 1.1	Makefile	.	188
F 1.2	vfd.h	.	189
F.1.3	vfd.c	..	191
G	Diagnostic Program Code Listings		195
G.1	mon Shared Memory Display Utility	.	195
G 1.1	Makefile	.	195
G 1.2	mon.h	..	195
G 1.3	cal.c	.	196
G 1.4	inputs.c	..	197
G 1.5	mode.c	.	199
G 1.6	mon.c	.	200
G 1.7	outputs.c	.	205
G 2	flogger Data Logging Utility	.	206
G 2.1	Makefile	..	206
G 2.2	flogger.c	.	206
G.3	trends.cgi CGI Interface to Shared Memory History	...	211
G 3.1	Makefile	.	211
G 3.2	trends.h	..	211
G 3.3	trends.c	..	212
G 3.4	trend_helper.c	.	214
G 4	libcgi C Language CGI Library	.	217
G 4.1	Makefile	..	217
G 4.2	cgi.h	.	217
G 4.3	get.cgi.c	.	217
Vita			221

List of Tables

2 1	Basic Input/Output requirements	. . .	13
2 2	System Controller hardware	. . .	16
2 3	Unique Mobility SR180 Motor Specifications	. . .	25
2 4	Unique Mobility SR218 Motor Specifications	. . .	25
2 5	Effect of Window size on <i>err</i>	. . .	38
4 1	Partial listing of fclient shared memory contents	. . .	67
4 2	die(), warn(), and notice() output format	. . .	69
B.1	TCP/IP configuration parameters	. . .	109

List of Figures

1 1	Series HEV configuration	4
1 2	Parallel HEV configuration	5
1 3	Split Dual HEV configuration	6
1 4	UTK 1999 FutureCar team and entry vehicle	7
2 1	Matrix Orbital Vacuum Fluorescent Display Module	12
2 2	System Controller Input/Output interface diagram	13
2 3	System Controller Input/Output list	14
2 4	Vehicle Control System Wiring Diagram, Top	20
2 5	Vehicle Control System Wiring Diagram, Bottom	21
2 6	UTK FutureCar Drive Train Schematic	23
2 7	Transmission Planetary Gear Connections	23
2 8	Unique Mobility SR180 Efficiency Map	25
2 9	Unique Mobility SR218 Efficiency Map	26
2 10	Open Circuit Voltage vs SOC for a single 12V lead acid battery	28
2 11	Discharge Schedule for Hawker 13Ah Genesis battery	29
2 12	EPA Standard Driving Cycles	32
2 13	A UTK FutureCar Driving Cycle	32
2 14	Fitting Experimental Data to Polynomials	33
2 15	A State of Charge Prediction Model	34
2 16	Measured and Predicted SOC, window size 1	35
2 17	Measured and Predicted SOC, window size 5	35
2 18	Measured and Predicted SOC, window size 10	37
2 19	Measured and Predicted SOC, window size 30	37
2 20	Measured and Predicted SOC, window size 14	38
2 21	Partial Test Case of Measured and Predicted SOC, window size 14	39
2 22	QNX microkernel architecture	42
2 23	UNIX Process Creation	44
2 24	UNIX Process Tree Relationship	45
2 25	QNX Send-Receive-Reply IPC Mechanism	46
3 1	ZEV generator algorithm	51
3.2	Throttle Position Torque Correction Factor	53

3 3	Estimated Engine Map	53
3.4	Throttle Up Procedure	55
3 5	Generator Speed Request and Torque Relation	57
3 6	Generator speed as a function of vehicle speed and engine speed	57
3 7	Maximum allowable throttle for given vehicle speed	58
3.8	Hybrid mode drive train component speeds	58
4 1	Shared memory connection	76
4 2	Fcard Process Tree	78
4 3	Fcard Octagon Hardware I/O Task Flowchart	78
4 4	Fcard kWh Meter Reader Task Flowchart	79
4 5	Fcard Config File Reader Task Flowchart	79
4.6	Fcard Task Manager Flowchart	80
4 7	Fcar Process Tree	82
4 8	Fcar SOC Calculator Task Flowchart	83
4.9	Fcar Throttle Controller Task Flowchart	83
4.10	Fcar ICE Controller Task Flowchart	85
4.11	Fcar Mode Selector Task Entry Point Flowchart	86
4 12	Fcar Mode Selector Task, ZEV Function Flowchart	87
4 13	Fcar Mode Selector Task, Reverse Function Flowchart	89
4.14	Fcar Mode Selector Task, Neutral Function Flowchart	90
4.15	Fcar Mode Selector Task, DriveEcon Function Flowchart	91
4 16	Mon shared memory viewer tool	93
4 17	Trends shared memory history viewer tool	94

List of Symbols and Abbreviations

<i>Ah</i>	Ampere hour capacity
<i>APU</i>	Auxiliary Power Unit
<i>CNG</i>	Compressed Natural Gas
<i>DOHC</i>	Dual Overhead Camshaft
<i>EPA</i>	United States Environmental Protection Agency
<i>ESS</i>	Energy Storage System
<i>HEV</i>	Hybrid Electric Vehicle
<i>ICE</i>	Internal Combustion Engine
<i>kWh</i>	kilowatt hour capacity
<i>POSIX</i>	Portable Operating System Interface
<i>SOC</i>	State of Charge
<i>T</i>	Torque (N m)
<i>ZEV</i>	Zero Emissions Vehicle
<i>N</i>	number of gear teeth
<i>N_{gen}</i>	generator gear teeth
<i>N_{ring}</i>	ring gear teeth
ω	angular speed
ω_{eng}	engine speed
ω_{gen}	generator speed
ω_{ring}	ring gear speed
<i>r</i>	gear ratio

Chapter 1

Introduction

Freedom has always been the appeal of the automobile. From the very beginning of the development of the “horseless carriage” when automobiles quickly evolved from noisy and harsh curiosities to a form of personal transportation that far exceeded the capabilities of the horse and buggy, the automobile has symbolized personal freedom. The freedom to go anywhere, anytime has made the automobile one of the most popular forms of transportation.

To accommodate the popularity of the automobile, vast infrastructures have been put in place. The United States Federal Interstate Highway system allows a person to drive from coast to coast without ever leaving pavement. All along this coast to coast route are fueling stations to feed the energy requirements of automobiles. Something strange happened along the road to personal freedom: the loss of freedom.

In the U.S., the explosive growth of the automobile required oil to be imported. Dramatic events in the 1970s, recorded in history as the “Energy Crisis,” demonstrated the reliance on imported oil when foreign oil supplies were artificially restricted. Since the automobile is the single largest consumer of petroleum products, the U.S. federal government made mandates to improve the efficiency, hence reducing fuel consumption and the reliance on foreign oil, of automobiles used on public roads. Research into automobile fuel efficiency

revealed that tail pipe emissions could also be lessened with a more fuel efficient automobile. In the time since the 1970s, great strides have been made to reduce fuel consumption and emissions, but for some states, like California, this has not been enough

The ultimate solution, claimed by some, is the electric vehicle, or EV. An EV burns no fuel and has no exhaust emission. Energy is taken from the electric "grid" infrastructure where power can be collected in a manner more efficient than the automobile's internal combustion engine. The main drawback to a purely electric vehicle is range. Current energy storage systems allow an EV to drive only about 100 km before needing to be recharged. Also the performance of most EVs leaves a lot to be desired by the general buying public. These disadvantages, limited driving range and poor performance, appear as a loss of freedom to the car buying public. While energy storage systems improve, there is a sort of "stepping stone" to the EV, the hybrid electric vehicle or HEV.

An HEV is a vehicle that has both an electric motor and a fuel burning engine of some kind. The HEV design is a combination of a conventional vehicle and an all electric vehicle. The combination of the two allows for a greater driving range than an EV. While an EV has to plug into the grid to recharge, an HEV can carry its own power plant. When electric storage is low, the onboard power plant can recharge the electrical system. Performance can be as good or better than a conventional vehicle when the electric motor and engine work together. The engine in an HEV can be smaller and more fuel efficient than a conventional vehicle because it has an electric power source to assist in driving demands. Since the HEV addresses the weaknesses of both the conventional petroleum burning vehicle and the purely electric vehicle, it actually represents an increase in personal freedom. In addition to the freedom to drive anywhere at any time, HEVs can give cleaner air and less reliance on fossil fuels.

1.1 HEV Types

An EV's drive train, almost by definition, is powered by an electric motor. Since a hybrid electric vehicle requires at least one electric motor and some other auxiliary power unit, APU, there is great flexibility in what an HEV actually is. The APU can take many forms. For the sake of simplicity, the following discussion will assume that the APU is a conventional internal combustion engine, or ICE. While the goal of all hybrid vehicles is to improve the overall vehicle efficiency, there are some advantages and disadvantages specific to each configuration.

1.1.1 Series HEV

The series HEV configuration, shown in Figure 1.1, is the simplest hybrid configuration. It is basically an EV with an onboard recharging engine/generator combination. Primary tractive power is provided by an electric motor. The drive train transmission can be simpler than one based on a conventional engine. The flat torque curve of an electric motor can simplify gearing because multiple gears are not needed to compensate for the narrow torque band of an ICE. Advantages to the series configuration are that because the engine/generator combination operates independently from the vehicle speed, the small engine can be run in its most efficient operating range and only when the electric storage needs recharging. A disadvantage to the design is the relatively large traction motor. This large motor, because it has to provide all motive force, needs to be over-specified because it has to power the vehicle not only during level stop-and-go traffic but also during a long, steep grade, highway drive.

1.1.2 Parallel HEV

The parallel HEV configuration is another basic HEV type. While the series configuration is basically an EV with an onboard power plant to extend driving range, the parallel hybrid is a

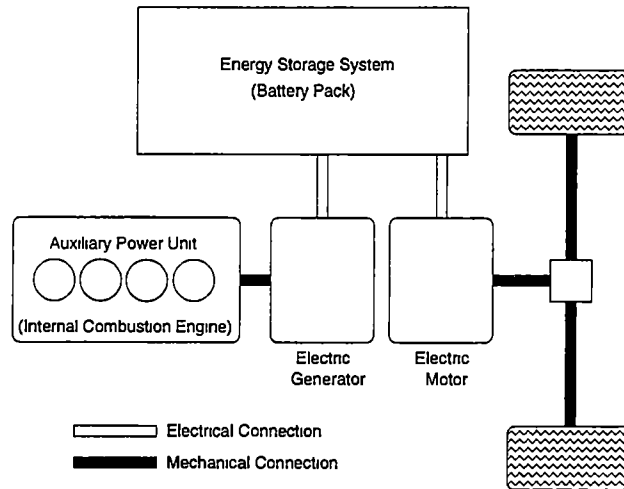


Figure 1.1 Series HEV configuration

conventional vehicle with a smaller engine and an electric motor to assist in providing torque. For this reason, the series and parallel types are sometimes called ‘range extending’ and ‘power assisting’ configurations respectively. While the series design requires two electric machines, a tractive motor and a generator, the parallel design, shown in Figure 1.2, requires only a single electric motor. The parallel configuration is so named because a small engine and small electric motor work in parallel to drive the wheels. The advantage to a parallel hybrid relative to a conventional vehicle is that a smaller, more efficient engine can be used. The smaller engine can fulfill most driver requests, and the electric motor can be used to provide additional power when needed. A disadvantage comes from the fact that both the engine and the electric motor have to operate at speeds dependent on vehicle speed. Also, a parallel hybrid requires a conventional multi-speed transmission because the engine is the prime mover. This means that the engine has to operate outside of its most efficient range. Additionally, because the electric motor is coupled to the engine, it must transfer power through the transmission which results in power loss. Recharging the battery pack in a parallel hybrid is limited because recharging can only be done while the vehicle is in motion. The already under-powered engine is burdened even more with the

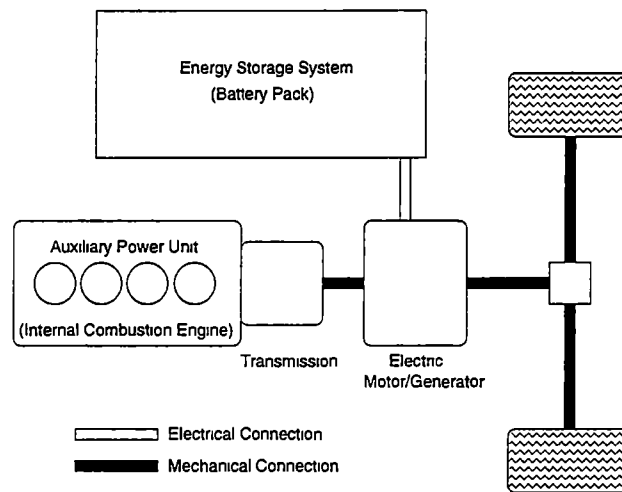


Figure 1.2 Parallel HEV configuration

task of recharging. Because of this, some parallel hybrids are “charge depleting” designs. That is, electric power is only used when the vehicle’s torque request is greater than can be provided by the engine and no attempt is made to recharge except during regenerative braking. These designs require the HEV to be recharged from the grid, but allow a greater driving range than an EV.

1.1.3 Dual Hybrid

While the series and parallel configurations have been long established, a relatively new design has been introduced. As described by Yamaguchi [20] the new configuration, the dual hybrid, is a combination of both series and parallel systems. A dual hybrid, like a series hybrid, requires two electrical motors and an engine. The dual hybrid description can be applied to two different configurations, switching and split. The split configuration is illustrated in Figure 1.3.

The switching configuration is very similar to a series hybrid except that a clutch can optionally allow the engine/generator combination to drive the wheels. A switching system can allow the vehicle to operate as a strict series when stop-and-go traffic makes a series

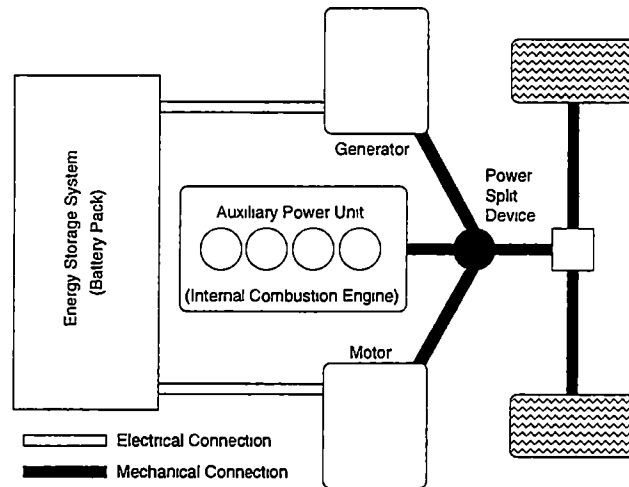


Figure 1 3 Split Dual HEV configuration

connection the most efficient choice. During conditions where a parallel connection is most efficient, high speed cruising for example, the switching system's clutch can be engaged to allow the engine to provide additional drive wheel torque.

A split dual hybrid operates as both a series and parallel at all times. Instead of an on/off device like a clutch, a power splitting device like a planetary gear set is used to connect the engine/generator and traction motor. A planetary gear connection allows engine power to be split along the parallel path, engine to driving wheels, and the series path, engine to generator. The use of two electric motors allows the split dual hybrid to operate in two modes, split positive and split negative. During split positive operation the generator motor acts as a generator and the traction motor acts as a motor. While in split negative operation, usually during times when battery state-of-charge (SOC) is high, the generator acts as a motor along with the main traction motor allowing increased torque availability to the drive wheels. Clearly, the multiple operating modes of the split dual hybrid allow for great flexibility in control schemes employed.

1.2 FutureCar Competition

The FutureCar competition is an event organized by the U.S. Department of Energy's Argonne National Laboratory in which vehicles converted to hybrid electric operation by university students compete in a series of events. As the three major U.S. automobile manufacturers, Ford Motor Company, General Motors, and Daimler Chrysler, are major sponsors of the event, schools were given a choice of stock vehicles to modify. A 1998 model year Dodge Intrepid, shown in Figure 1.4, was chosen as the base vehicle by the University of Tennessee, Knoxville.

The 1999 UTK FutureCar team included both undergraduate and graduate students. The major responsibility of the graduate section of the team was to provide a control system for the team's dual hybrid electric conversion vehicle. What follows is a description of the control system developed by the graduate team.

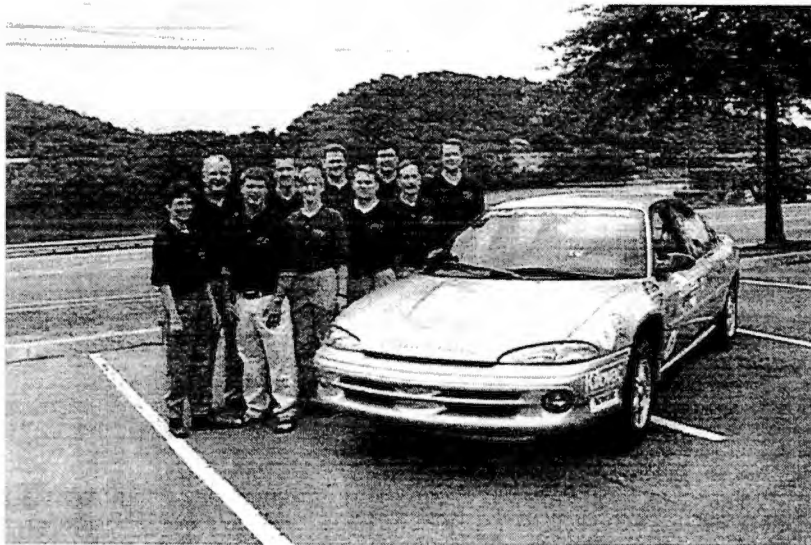


Figure 1.4: UTK 1999 FutureCar team and entry vehicle

Chapter 2

Design Phase

In the design phase many aspects of the vehicle as a system need to be considered. Specifically, the available hardware interface to the various components needs to be specified. With this information, a decision on hardware for the system controller can be made. Knowing how everything is wired together only shows part of the system. The interactions of the drive train components through the transmission need to be investigated in addition to how state-of-charge is determined for the battery pack. With the known requirements of the system, the software environment can be specified.

2.1 Hardware Interface

The UTK FutureCar has many subsystems that must be monitored or controlled by the system controller. The major drive train subsystems include the ICE, traction motor, generator, battery pack, and driver interface. Each of these has separate and unique interfaces

2.1.1 Drive Train Controllers

Engine Control

The Engine Control Unit, ECU, chosen by the FutureCar team was a TEC-2 made by Electromotive Inc [3] The TEC-2 determines fuel injection pulse widths and spark ignition timing, and it controls the on board engine emission control equipment. Sensors are standard GM parts. The TEC-2 controller was chosen because of past UTK experience[16] and the ability to specify various operating parameters. The user interface is provided by a DOS program that communicates over a standard RS-232 serial port. Since the TEC-2 was designed to control a wide variety of engines, the control interfaces are simple. Engine speed feedback is provided by a tachometer signal, a 0-12V square wave. Starting the engine is performed by powering the TEC-2 controller and then shorting the starter control wires. Stopping is just as simple by just removing power from the TEC-2 which then halts ignition and fuel delivery.

While most conventional vehicles control engine speed through a linkage connecting the accelerator pedal to the engine throttle body, the UTK FutureCar uses a remote powered throttle body manufactured by Mikuni Inc. The remote throttle body encompasses a servo motor whose control signal is 0-5VDC and provides throttle position feedback with a signal in the same range as the control. This feature is a requirement to the vehicle design since, unlike a conventional vehicle, the engine must be controlled independently from the driver request through the accelerator pedal.

Traction Motor Controller

The traction motor controller interface is an EVPH332 Digital Controller that is part of the Unique Mobility[19] "Caliber EV 53" motor package. The controller is a sophisticated piece of hardware that provides a multitude of performance and status indicators. The controller inputs for brake and accelerator request are two 0-5VDC pins on the controller housing.

Digital lines are also provided for direction and system enabling. A less complex hybrid design could use the EVPH332 directly without a system controller.

Generator Controller

The generator control interface is the CR20-300A inverter/controller that comes packaged with a Unique Mobility, UQM, SR180 motor. The UQM SR180 package is an older style compared to the SR218. While the SR218 interface is a small microprocessor, the SR180 requires that control connections are made directly on the same hardware that acts as a high voltage inverter. Many digital signals exist for gathering status information, and the main control input is a $\pm 10\text{VDC}$ signal that represents both requested speed and direction. A 0 to -10VDC signal is used to indicate the requested regeneration level. A $\pm 10\text{VDC}$ signal is provided for motor speed feedback.

2.1.2 Battery Pack

The UTK FutureCar uses Lead Acid battery technology for its Energy Storage System, ESS. Lead Acid was chosen because of its low cost, relative robustness, and availability. Specifically, 27 Hawker Energy 13Ah Genesis model batteries compose the battery pack. Using 27 batteries in series allows for a nominal pack voltage of about 345V. While most previous UTK HEV projects used a battery pack with a much lower nominal voltage, using a higher pack voltage was specified in an attempt to lower I^2R losses and, hence, operate more efficiently. Originally, 28 batteries were specified, but this introduced problems during aggressive charging with the generator. Occasionally the pack voltage would exceed 400V, beyond the capabilities of the generator system.

To monitor the battery pack, a Cruising Equipment Kilowatt-Hour+2 meter was installed. The meter provides an RS-232 serial interface that represents electrical energy consumed, pack current, and voltage all as ASCII text data. The traction motor microcontroller, the EVPH332 unit, also provides pack voltage as an analog 0-10VDC output.

signal Both motor controllers also provide feedback on motor current consumption.

2.1.3 Driver User Interface

The stock vehicle driver interface included analog gauges for vehicle speed, engine speed, engine coolant temperature, and fuel level and a few indicator lights Ideally, a hybrid version of a Dodge Intrepid would leave as many of the original driver controls as possible. The initial idea was to leave the original speedometer gauge and add a Matrix Orbital vacuum fluorescent display, VFD, Figure 2 1, for hybrid specific driver display The VFD displays characters written to its RS-232 serial port and has a few special features such as the ability to display bar graphs and large characters Eventually the stock speedometer gauge was removed because it proved to be too difficult to control without manufacturer documentation with the I/O hardware available The stock vehicle used a center console mounted shift lever for controlling the automatic transmission This was replaced with an eight position switch mounted on the dashboard in front of a blocked-off heating vent The original shifter position had a mounting plate installed for a small laptop computer that might provide additional feedback for the driver

2.2 System Controller

The previously mentioned hardware is self-contained, that is, each component can work independently of the others In a simpler vehicle, perhaps only one of the subsystems might be used This dual hybrid vehicle has to have a central authority that monitors and controls each individual subsystem The system controller must perform this task

2.2.1 I/O requirements

One of the main responsibilities of the system controller is to manage all of the input/output (I/O) lines for each piece of hardware Some systems, like the motor controllers, interface

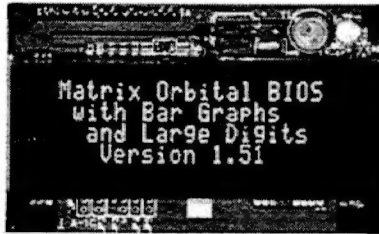


Figure 2.1: Matrix Orbital Vacuum Fluorescent Display Module

with multiple analog DC voltages and digital TTL level voltages. Others, like the TEC-2 engine controller, convey information with time dependent waveforms. Still others interface with an RS-232 serial link. Figure 2.2 shows how the system controller must manage multiple subsystems with differing I/O interfaces. In addition to the main drive train subsystems, other smaller signals have to be monitored. These include brake line pressure, fuel tank pressure, accelerator pedal position, and more. Figure 2.3 shows the working list of I/O parameters that need to be managed by the system controller hardware. Table 2.1 summarizes the basic I/O requirements.

Isolation

Another requirement of the system controller is that it must be electrically isolated from potentially dangerous systems. Electrical isolation is usually achieved with an optical transmitter and receiver pair. This allows the input and output to not share any common electrical signals. Because the conversion from electrical to optical and back to electrical is a highly non-linear process, analog optoisolation modules are more complex and expensive than digital modules where the non-linearity has no effect. Dangerous systems include the high voltage battery system and the internal combustion engine. These two systems have operating voltages that are potentially harmful to computer equipment, especially if an unintentional ground loop introduces currents that could be dangerous to both computers and people. Ideally, all I/O would be electrically isolated, but a few critical systems demand it.

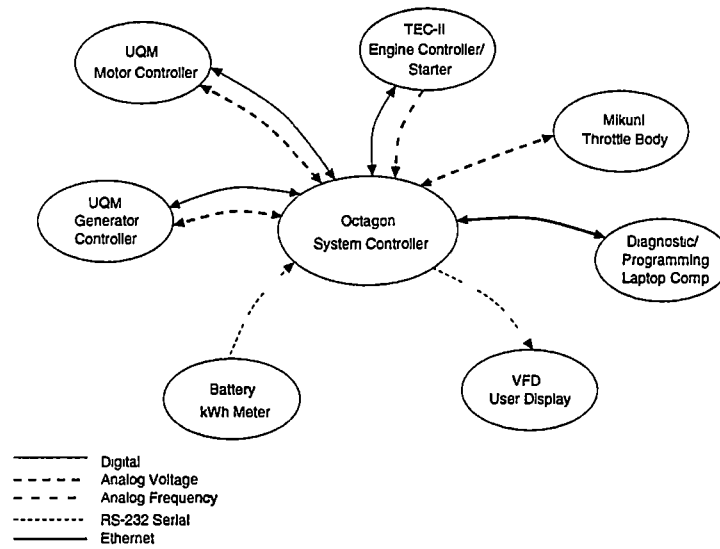


Figure 2.2 System Controller Input/Output interface diagram

Table 2 1 Basic Input/Output requirements

I/O type	subtype	Number
Digital	Input	16
	Output	8
Analog	Input	13
	Output	9
Frequency	Input	1
Serial	RS-232	2

<i>Digital Input - all from one card, one 1x16 G4 rack</i>								
#	card	port bit	connector	Description	Range	Sampling Freq	Sensor/Actuator	notes
1	5710-0	PC 0	J1-0 (DIG0)	shifter button	0 to 12vdc	medium	blender digital switch	
2	5710-0	PC-1		shifter button	0 to 12vdc	medium	blender digital switch	
3	5710-0	PC 2		shifter button	0 to 12vdc	medium	blender digital switch	
4	5710-0	PC 3		shifter button	0 to 12vdc	medium	blender digital switch	
5	5710-0	PC-4		shifter button	0 to 12vdc	medium	blender digital switch	
6	5710-0	PC 5		shifter button	0 to 12vdc	medium	blender digital switch	
7	5710-0	PC 6		HVAC enable/disable	0 to 12vdc	low	existing switch	
8	5710-0	PC 7		EM Temperature warning	0 to 12vdc	low	EM microprocessor	
9	5710-0	PA-0		EM Controller Ready	0 to 12vdc	low	EM microprocessor	
10	5710-0	PA-1		EM Fault Indicator	0 to 12vdc	low	EM microprocessor	
11	5710-0	PA 2		EM OverTemp Indicator	0 to 12vdc	low	EM microprocessor	
12	5710-0	PA-3		GEN Temperature warning	?	low	GEN controller	
13	5710-0	PA-4		GEN Controller Ready	?	low	GEN controller	
14	5710-0	PA 5		GEN Fault Indicator	?	low	GEN controller	
15	5710-0	PA-6		GEN Direction Indicator	?	low	GEN controller	
16	5710-0	PA 7		ICE Fault Indicator	0 to 12vdc	low	TEC-II	

<i>Digital Output - all from one card, one 1x16 G4 rack</i>								
#	card	port bit	connector	Description	Range	Sampling Freq	Sensor/Actuator	notes
1	5710-1	PC 0	J1-1 (DIG1)	ICE Starter Solenoid	0 to 12vdc	medium	short starter wires	
2	5710-1	PC 1		ICE TEC II enable	0 to 12vdc	low	TEC-II	
3	5710-1	PC 2		EM enable	0 to 5vdc	low	EM microprocessor	open=on, gnd=off
4	5710-1	PC 3		EM direction	0 to 5vdc	low	EM microprocessor	open=fwd, gnd=rev
5	5710-1	PC-4		GEN enable	0 to 12vdc	low	GEN controller	
6	5710-1	PC 5		Power Steering enable	0 to 12vdc	low	ps motor pump rly	
7	5710-1	PC 6		Mikuni Throttle Pwr Cycle	0 to 12vdc	low	relay	

<i>† Analog Input - most straight thru except 3 5B modules on a 1x6 5B rack (differential mode)</i>									
#	card	channel	card	pin(+,-)	Description	Range	Sampling Freq	Sensor/Actuator	notes
1	5710-0	1	J2-0	1 3	EM Motor Speed	0 to 10vdc	high	EM microprocessor > vdiv	1
2	5710-1	1	J2-1	1 3	GEN Motor Speed	-10vdc to +10vdc	high	GEN controller > vdiv	1
3	5710 1	3	J2-1	5 7	ICE Engine Speed	0 to 5vdc	high	TEC-II > 5B	2
4	5710-1	5	J2	19 11	Accelerator pedal position	0 to 5vdc	medium	Mikuni elec. throttle	
5	5710 1	7	J2-1	13 15	Brake pedal level	0 to 5vdc	medium	?pressure gauge?	
6	5710-1	9	J3	1 1 3	Actual EM Torque	0 to 10vdc	medium	EM microprocessor > vdiv	1
7	5710-1	11	J3	1 5 7	Fuel Pressure	0 to 6vdc	low	press transducer	
8	5710 1	13	J3	19 11	EM Current	0 to 10vdc	low	EM microprocessor > vdiv	1
9	5710-1	15	J3	1 13 15	GEN Current	0 to 10vdc	low	GEN controller > vdiv	1
10	5710-0	3	J2-0	5 7	Battery Pack Temperature	0 to 5vdc	low	?thermistor?	
11	5710-0	5	J2-0	9 11	Battery Terminal Voltage	0 to 10vdc	low	EM microprocessor -> vdiv	1
12	5710-0	7	J2-0	13 15	Battery Current	0 to 5vdc	low	shunt -> amp > 5B	
13	5710-0	9	J3-0	1 3	EM Rotor Temperature	0 to 10vdc	low	EM microprocessor > vdiv	1
14	5710-0	10	J3-0	5 7	EM Inverter Temperature	0 to 10vdc	low	EM microprocessor -> vdiv	1

<i>Analog Output - all straight thru, no isolation, never reference chassis ground (differential mode)</i>									
#	card	channel	card	pin(+,-)	Description	Range	Sampling Freq	Sensor/Actuator	notes
1	5710-1	0	J2	1 17 16	GEN Speed Request	-10vdc to +10vdc	high	GEN controller	
2	5710 1	1	J2	1 19 16	GEN Regen Limit	0 to -10vdc	high	GEN controller	3
3	5750-0	0	J1-2	1 3	EM Accel Req	0 5vdc to 4 5vdc	high	EM microprocessor	
4	5750-0	1	J1	2 4 6	EM Brake Req	0 5vdc to 4 5vdc	high	EM microprocessor	
5	5750-0	2	J1-2	7 9	ICE Throttle Position	0 to 5vdc???	high	Mikuni elec throttle	
6	5750-0	4	J1	2 13 15	Display Speedometer 0	-5vdc to +5vdc	low	amp -> existing gauge	
7	5750-0	5	J1	2 16 18	Display Speedometer 1	-5vdc to +5vdc	low	amp -> existing gauge	
8	5750-0	6	J1	2 19 21	Display Fuel/Energy Level	-5vdc to +5vdc	low	amp -> existing gauge	
9	5750-0	7	J1	2 22 24	Power on Digital racks	5vdc to +5vdc	low	relay	4

<i>What's Left</i>			
	5710-0	5710-1	5750-0
Digital	0	9 (1 out 8 I/O)	-
Analog Input	2 differential	0 differential	-
Analog Output	2	0	1 (ch 3 0-5vdc)

Notes

- 1 5710 analog input only capable of +/- 5vdc (voltage divider needed)
- 2 Tach signal needs to go through a 5B frequency to dc voltage conditioner
- 3 5750 not able to output +/- 10vdc have to use a 5710
- 4 relay needed to suppress startup jitters of the digital output lines
- † Analog Inputs all set to operate in differential mode (only all diff or all single possible)

Figure 2 3 System Controller Input/Output list

2.2.2 Computational requirements

Since the system controller must be more than a data logger, some computational ability is needed. The primary task of the system controller is to make decisions based on input data and relay these choices with output signals. According to Laplante [10], a system that is over loaded or very highly loaded, 98%, is undesirable because of the lost flexibility. A system without enough computational power restricts changes that could be made to the control system code, while a system that is continuously underloaded, < 10% or so, is also undesirable for a production system. Because underloading represents wasted resources, a production system's CPU hardware could be lessened along with a decrease in cost. Since the UTK FutureCar is not a production system but a research platform, too much CPU power is not possible. By using a very overpowered CPU, restrictions on later unforeseen control code changes can be lessened. While more CPU power than initially needed is desired, there are potential drawbacks to using a high powered CPU, namely high electrical power consumption and/or high operating temperature, neither of which are desirable.

2.2.3 System Controller Hardware

Much of the system controller hardware was inherited from the UTK HEV NEON [18] project, including components manufactured by Octagon Systems Corporation. Octagon manufactures ruggedized IBM PC compatible computer equipment suitable for embedded operation. A PC compatible system was chosen because of the wide variety of software and hardware available for the platform. The Octagon cards are connected with an 8-bit wide PC ISA bus in a passive backplane. Table 2.2 summarizes the chosen hardware. All of the controller hardware operates at 5VDC. Any higher voltage levels are produced on-board.

CPU

An Octagon 5066 "Micro PC" form factor CPU card houses the main CPU and other components found in a regular PC with the exception of a video card. The 5066 is an

Table 2 2 System Controller hardware

CPU	1 Octagon 5066 card 133 MHz AMD 80486 CPU w/ integrated FPU 33 MB RAM (1MB soldered, 32MB socketed) 2 RS-232 capable serial ports programmable watchdog timer persistent real time clock (with optional battery) flexible solid state storage options PS2 keyboard interface
Persistent Storage	1 M-Systems DiskOnChip 2000 72 MB storage capacity able to emulate standard IDE disk
Communication	1 Octagon 5500 Ethernet Card IEEE 802.3 ethernet capability (10 Mbit/sec) 10-base-T, 10-base-2, 10-base-5 interfaces based on Western Digital 8003 ethernet controller
Multifunction I/O	2 Octagon 5710 cards 16 single-ended or 8 differential analog inputs, each 2 analog outputs, each 16 digital I/O lines, each
Analog Output	1 Octagon 5750 card 8 analog outputs
Digital Isolation	Opto-22 G4 modules 1500 VAC isolation mounted in two Octagon MPB-16 racks
Analog Isolation	2 Dataforth 5B analog signal conditioners 1500 VAC isolation mounted in Computer Boards Inc ISO-DA08 rack
Frequency-Voltage Converter	1 Dataforth 5B module 1500 VAC isolation mounted with other 5B modules 0-500 Hz to 0-5 VDC conversion
RS-232 Serial Isolation	Computer Boards Model 268 1500 VAC Isolation up to 19.2 Kbps operation passive, operates on serial line power

updated version of the 5025A card used in the UTK HEV NEON. The card is fitted with an Advanced Micro Devices Inc (AMD) 133 MHz 80486 processor with an integrated 80487 floating point coprocessor and 1 MB of memory. An onboard SO-DIMM (Small Outline, Dual Inline Memory Module) socket allows system memory to be increased to the maximum of 33 MB RAM. Some of the features of the 5066 that are unlike a regular desktop PC include a programmable watchdog timer, extended temperature operation, and SSD (Solid State Disk) support. It also features a BIOS (Basic Input Output Services) with settings stored in non-volatile EEPROM (Electrically Erasable Programmable Read Only Memory) that allows for battery-less operation. If an external 4.5VDC battery is used, the onboard real time clock can retain date and time information when powered off. Two serial ports are also available.

Persistent Storage

The Octagon 5066 card has a socket in which an M-Systems DiskOnChip (DOC) 2000 is mounted. The DOC is a solid state flash memory device that emulates a standard PC IDE disk in a compact package. The relatively large 72 MB capacity was chosen to allow for onboard data logging. While the solid state device might be somewhat slower than an IDE disk, the benefit of compact size and no moving parts influenced its choice.

Communication

An Octagon 5500 ethernet card was chosen for high speed communication. Since the card is based on a fairly common ethernet controller, software drivers for a multitude of operating systems are available. Additionally, having many standardized physical interfaces, 10-base-T (twisted pair), 10-base-2 (thinnet coax), and 10-base-5 (thicknet coax) allows the card to easily integrate into almost any ethernet topology.

Multifunction I/O

Two Octagon 5710 cards provide the majority of the system controller's I/O capabilities. In total, this amounts to 16 differential analog inputs, 4 analog outputs, and 32 digital I/O lines that can be addressed in groups of eight. All 5710 analog signals are 12 bit, that is, the precision available is 1 count in 4096 (2^{12}). The analog input ranges are fixed at $\pm 5\text{VDC}$ while the analog output ranges can individually be set to 0-10VDC, $\pm 10\text{VDC}$, or $\pm 5\text{VDC}$. The digital I/O lines are designed to interface to Opto-22 G4 style optoisolator modules.

Analog Output

A single Octagon 5750 card provides 8 12-bit analog output channels. Since the 5710 cards only provide a total of four analog output signals, a 5750 card fulfills the remaining required capability. The output ranges for each channel can be set independently to $\pm 5\text{VDC}$, 0-5VDC, or 0-10VDC.

Digital Isolation

To protect the system controller, all of the digital I/O lines on the 5710 cards are connected to optical isolation modules. Opto-22 G4IDC5D modules protect the input lines and G4ODC5 modules allow the output lines to switch loads up to 3A. The digital isolation modules are mounted in two Octagon MPB-16 racks with input lines on one rack and output on the other. The MPB-16 rack allows a direct connection to a 5710 card with a 26-pin ribbon cable.

Analog Isolation

While ideally all analog signals would be electrically isolated from other systems, only two input channels can be, cost being the limiting factor. Mounted in a Computer Boards Inc. ISO-DA08 rack are two Dataforth SCM5B41 modules that isolate a $\pm 10\text{VDC}$ input signal and convert it to a 0-5VDC output signal.

Frequency to Voltage Conversion

Also mounted in the ISO-DA08 rack is a single Dataforth SCM5B45 module that isolates a 0-500HZ input signal and converts to a 0-5VDC analog signal. The main purpose of this module is to interface with the TEC-2 tachometer signal

RS-232 Serial Isolation

One of the serial connections on the 5066 CPU card connects directly to the VFD display module. The other must interface with the kWh meter. Since the serial line from the meter references ground from the main battery pack, a Computer Boards Model 268 RS-232 isolation module is used to protect the system controller.

The system controller is housed in an aluminum box constructed by team members. This box is mounted in the trunk in the spare tire well. All signals that are not otherwise protected with isolation equipment are protected with $\frac{1}{16}$ A fuses and connect to the I/O cards with terminal blocks. Switching loads greater than the 3A capacity of the digital optoisolators is accomplished with 30A automotive lighting relays. The complexity of the customized vehicle wiring to the system controller is illustrated with the top of the diagram in Figure 2.4 and the bottom in Figure 2.5. This diagram shows how each component is wired to the system controller.

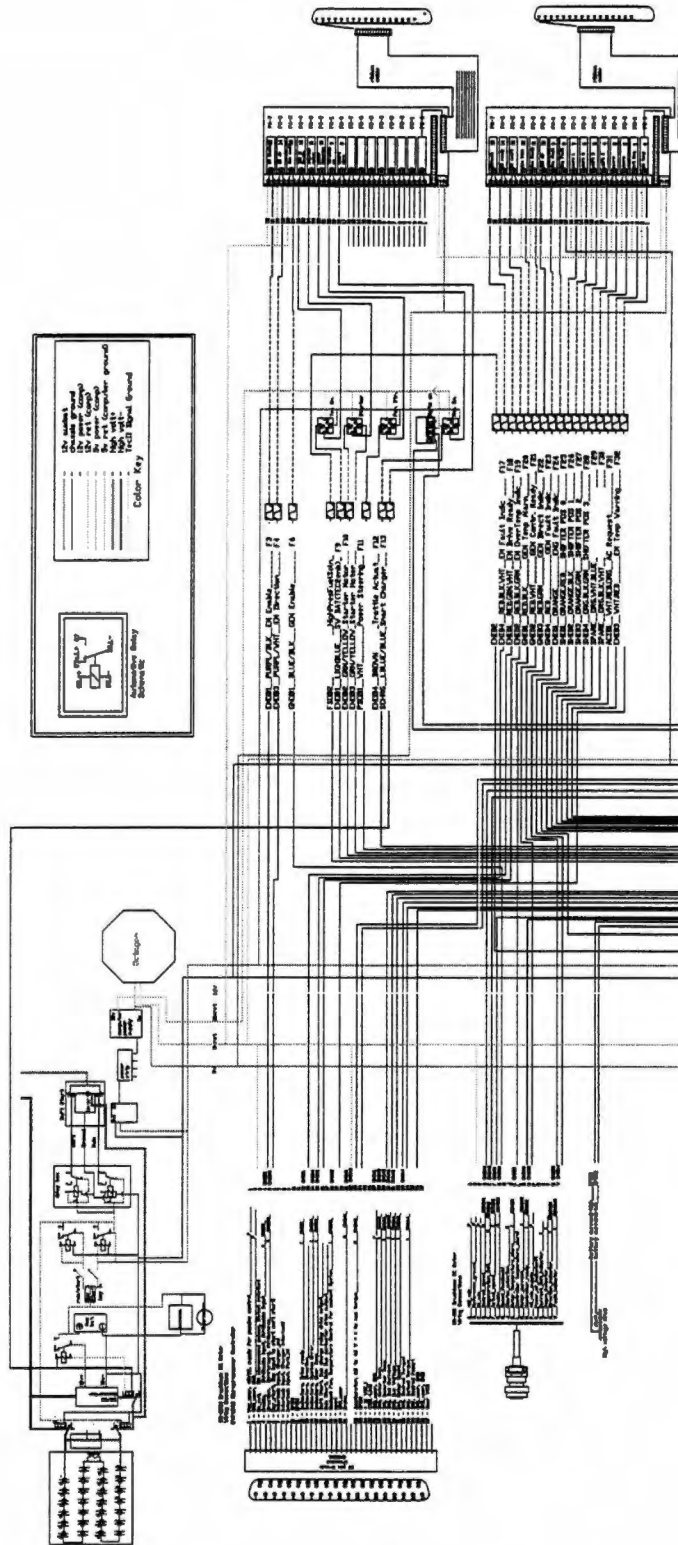


Figure 2.4: Vehicle Control System Wiring Diagram, Top

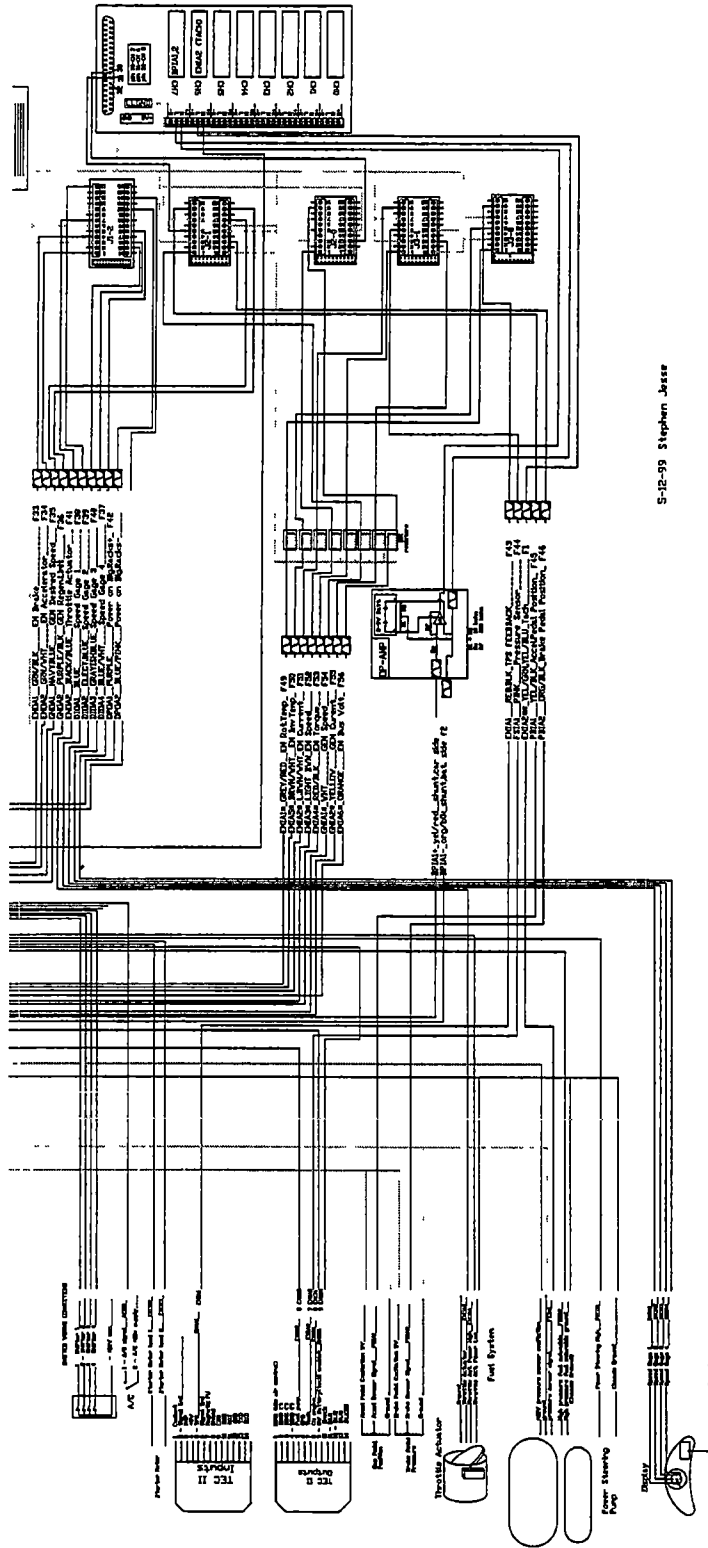


Figure 2.5. Vehicle Control System Wiring Diagram, Bottom

2.3 Transmission Model

The planetary, or epicyclic, gear train design of the UTK FutureCar allows for enormous flexibility in control strategy. The control scheme employed will follow the description of the positive split mode as described by Yamaguchi[20]. A schematic of the drive train with all of the relevant gear ratios is shown in Figure 2.6.

2.3.1 Drive Train Components

The drive train includes an engine, a generator motor, a traction motor, a planetary gear set, an over running clutch on the engine, and reduction and differential gears. As shown in Figure 2.7, the engine is connected to the planet carrier, the generator is connected to the central sun gear, and the main traction motor is connected to the output ring gear. This configuration acts as a sort of Continuously Variable Transmission, CVT, because there are no gears to shift. There is only a single forward gear. There is no gearing for reverse. Reverse 'gear' is accomplished by just reversing the direction of the traction motor.

Engine

Since the hybrid vehicle design allows for a smaller-sized engine, the stock 1998 Dodge 3.6 liter 6 cylinder gasoline engine was replaced with a 1998 model year Saturn 1.9 liter DOHC engine converted to run on compressed natural gas. While a smaller displacement engine would be adequate, previous UTK experience with the Saturn 4 cylinder engine design dictated its selection. The engine is coupled through an over running clutch to the planetary carrier to prevent the generator from inadvertently spinning the engine backwards.

Transmission Housing

The transmission housing comes from a 1982 Audi Quattro. It was chosen because its longitudinal four-wheel-drive design features drive outputs like a front wheel drive transmission.

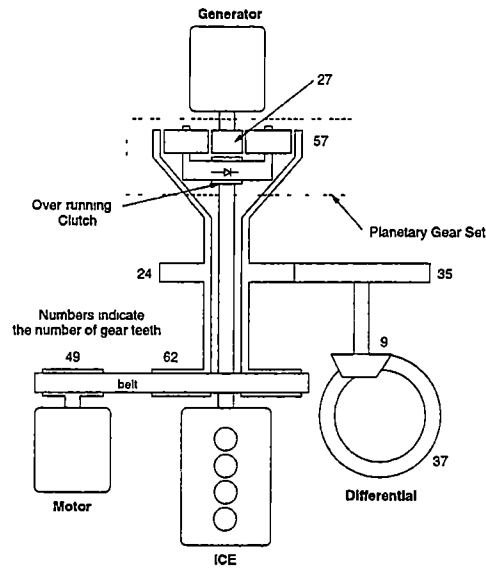


Figure 2 6· UTK FutureCar Drive Train Schematic

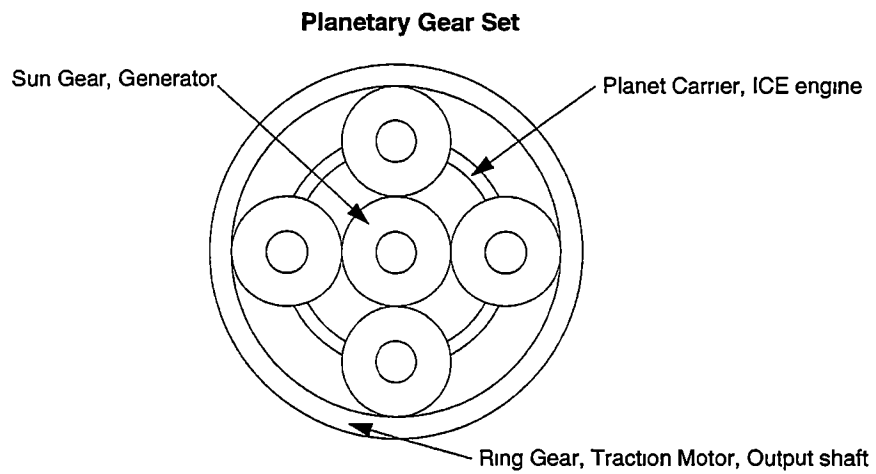


Figure 2 7 Transmission Planetary Gear Connections

and access from the rear of the housing for mounting the generator. While custom designing and manufacturing the housing in-house was considered, the availability of a commercial cast aluminum housing that could be modified for hybrid operation made the Audi housing a better choice.

Generator Motor

A custom Unique Mobility SR180 motor is mounted at the rear of the transmission housing to act as a generator. This model was chosen because it is physically small, lightweight, and speed controlled. Previous UTK experience with the SR180 came from the NEON HEV project where the nominal high voltage system bus voltage was 180V. A custom motor had to be ordered to work with the UTK FutureCar's 345V nominal high voltage bus. Figure 2.8 shows the manufacturer's efficiency map and Table 2.3 shows some of the motor's specifications.

Traction Motor

A Unique Mobility SR218 motor serves as the primary traction motor. This motor was chosen because of its physically small size, lightweight design, and torque based control. Because this motor acts as the prime mover in the drive train, torque control makes it more closely mimic the torque based feedback of a conventional vehicle's accelerator pedal coupled to an engine throttle. Figure 2.9 shows the manufacturer's efficiency map and Table 2.4 shows some of the motor's specifications.

2.3.2 Torque and Speed Relations

From Muller[12], the governing equation for speed of the planetary gear for this specific transmission is

$$\omega_{ring} = \frac{(1+r)\omega_{eng} - \omega_{gen}}{r}, \quad (2.1)$$

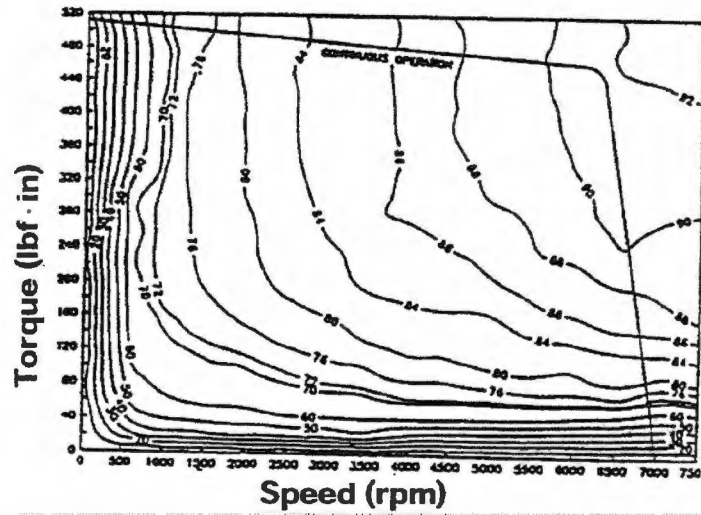


Figure 2.8: Unique Mobility SR180 Efficiency Map

Table 2.3: Unique Mobility SR180 Motor Specifications

Peak Power Rating	32 kW (42.9 hp)
Continuous Torque @ 6600 rpm	46.3 N·m (410 lbf·in)
Peak Torque in continuous stall	57.6 N·m (510 lbf·in)
Peak Torque in intermittent stall	90.4 N·m (800 lbf·in)
Maximum no-load speed @ 195V	7000 rpm
Weight	23.6 kg (52.0 lb)
Control Method	Speed Based

Table 2.4: Unique Mobility SR218 Motor Specifications

Peak Power Rating	53 kW (71 hp)
Continuous Power Rating	32 kW (43 hp)
Maximum Speed	8000 rpm
Weight	40 kg (89 lb)
Control Method	Torque Based

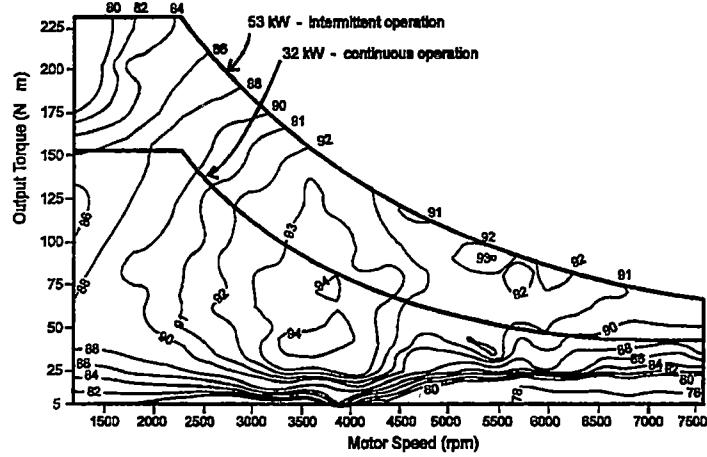


Figure 2.9 Unique Mobility SR218 Efficiency Map

where

$$r = \frac{N_{ring}}{N_{gen}} = \frac{57}{27} \approx 2.11, \tag{2.2}$$

thus,

$$\omega_{gen} \approx (3.11)\omega_{eng} - (2.11)\omega_{ring} \tag{2.3}$$

As is evident from these equations, setting one component speed to a specific value does not determine the speeds of the other components. This allows the speed of the engine and generator to vary over a range independent of wheel speed.

The governing steady state torque balance equations for this specific transmission are

$$T_{ring} = T_{eng} \quad r_1 = T_{gen} \quad r_2, \tag{2.4}$$

where

$$r_1 = \frac{N_{ring}}{N_{gen} + N_{ring}} = \frac{57}{84} \approx 0.68 \tag{2.5}$$

and

$$r_2 = \frac{N_{ring}}{N_{gen}} = \frac{57}{27} \approx 2.11 \tag{2.6}$$

As can be seen from these equations, component torques do not share the same degrees of freedom that speeds do. Under steady state conditions, generator torque and engine torque always have the same ratio regardless of the magnitudes of the speeds and torques of all components.

The transmission will not always be operating in steady state or even quasi-steady state conditions. That is, at some times, the rate of change of the component speeds is on the same order of magnitude as the speeds of the components. The assumption is made that, if the transmission is not in steady or quasi-steady state conditions, then it will be accelerating to such conditions. The transmission will not diverge from steady state conditions by virtue of the fact that the generator is speed controlled, and its torque is significant enough to force the engine to any operating speed regardless of engine power output. If this assumption is incorrect and the generator is unable to maintain a constant speed while under load, safety precautions are taken to trim the engine throttle before the generator exceeds safe operation speeds.

2.4 Battery State of Charge Model

One critical run-time parameter of any HEV is battery pack State of Charge, or SOC. A charge sustaining system must monitor the SOC and take appropriate action when the SOC is low. Additionally it must prevent battery over charging when the SOC is high.

2.4.1 Initial Approach

The initial approach to estimating SOC utilized a rather traditional method. When current draw on the high voltage system is low, a simple linear model can be used[6]. A graph of open circuit voltage for a single cell, shown in Figure 2.10, shows that SOC is linearly proportional to battery terminal voltage when current draw is negligible. While this might be all the information needed for a low power application, an HEV operates much of the

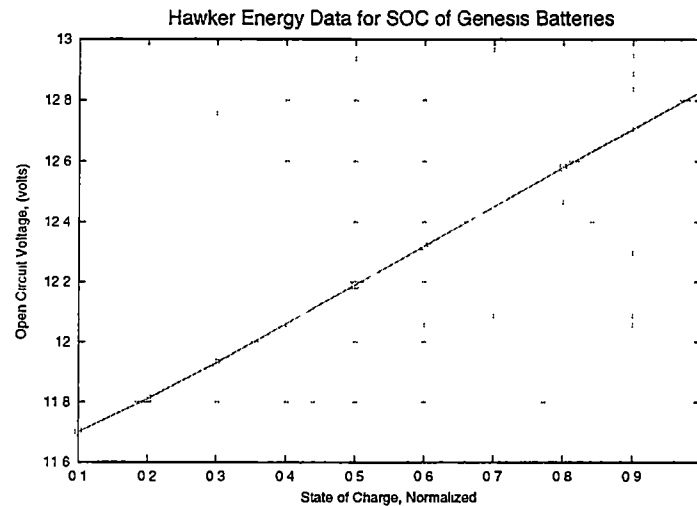


Figure 2.10 Open Circuit Voltage vs SOC for a single 12V lead acid battery

time when the current draw can be hundreds of amperes. For load conditions the battery manufacturer [5] provides data for capacity over varying current loads, shown in Figure 2.11. This figure shows capacity in ampere-hours (Ah) over constant discharge rates to 1.67 Volts per Cell, VPC. The 12V batteries used have six cells each, so total discharge is taken to be about 10V across the terminals under load. The initial algorithm for determining SOC operated in two modes based on the current draw. If the current draw was approximately zero, the data from Figure 2.10 would be used. Otherwise, the number of Ah consumed would be calculated by discretely integrating over time which then would be subtracted from the total capacity calculated from the data in Figure 2.11. This method seems reasonable, but there are several disadvantages.

- The manufacturer's data is given for a single 12V battery, while the UTK FutureCar's pack consists of 27 batteries in series. While each battery is manufactured to a certain specification, there are differences from battery to battery. This model does not take battery-to-battery interactions into account.
- While there are 'rules of thumb', or heuristics, for temperature changes, e.g. "battery

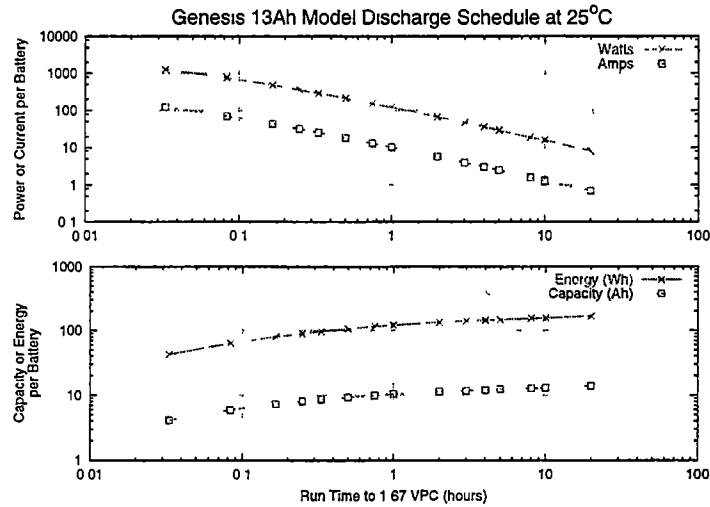


Figure 2.11 Discharge Schedule for Hawker 13Ah Genesis battery

life is reduced by 50% for every 30 °C drop with a 30 °C nominal temperature” and “battery life is increased by 50% for every 30 °C increase but capacity is decreased,” battery performance characteristics due to temperature changes are not provided by the manufacturer

- The manufacturer provides no information on how a battery will react to charging. Considering that a charge-sustaining hybrid spends a great deal of time recharging the batteries, lack of this data is a severe problem.

2.4.2 Common Sense Approach

While the initial approach proposed above seems sound, it was never implemented because of concerns about its inability to track SOC during charging. Also, hardware that could accurately measure battery pack current was not functional at the time of the competition.

After functional battery pack monitoring equipment was installed, the whole concept of State of Charge was reexamined. All previous methods required summing current readings

over time and dividing that by a hypothetical capacity. While measuring a pack current value could be accurate enough, all models for estimating total pack capacity seemed lacking. Conventional wisdom says

A lead-acid battery will react by way of its terminal voltage in response to a given current demand uniquely based on its state of charge

Since the battery current measuring device was not functional during most of the vehicle testing but pack voltage measurements were readily available, a periodically updated voltage value was displayed for the driver to use. During the first drive after hybrid mode was functional, the pack failed after about 15 km, probably due to over-charging. This required disassembling the pack, locating damaged batteries, and replacing them with known good batteries. After this initial failure, the driver learned to closely monitor the battery pack voltage to assess the condition of the pack. With only the knowledge of how the pack voltage reacts to an estimated load (initial startup, acceleration, grade, etc.), the human driver could make an educated guess to the pack SOC after some experience behind the wheel.

The goal for this approach is to encompass the human knowledge and express it as an algorithm so that a computer could make the same decisions as an expert test driver. To quote from Tsoukalas[17]

Artificial intelligence is a branch of computer science that attempts to emulate certain mental processes of humans by using computer models. In expert systems, perhaps the first field of artificial intelligence to be commercially recognized in its own right, one of the primary objectives is to mimic human expertise and judgment using a computer program by applying knowledge of specific areas of expertise to solve finite, well-defined problems.

One of the primary characteristics[7] of an expert system is the reliance on human knowledge instead of formal reasoning methods. This comes from the fact that, for most problems where expert systems techniques are applied, there exist no definitive algorithmic solutions. This is so because these problems involve complex social or physical situations which resist precise description and rigorous analysis. A battery SOC calculation is one of

these problems where there are many complex physical, chemical, and electrical reactions within a single battery, and even more in a pack of multiple individual batteries.

Derivation

To collect data for this experiment, the UTK FutureCar was driven around the University of Tennessee, Knoxville, campus in early August of 1999 when the temperature was approximately 35 °C and the relative humidity was about 85%. While most vehicle experiments are performed on a chassis dynamometer following a standard velocity profile, such as the Federal Urban Driving Schedule (FUDDS) or the Federal Highway Driving Schedule (FHDS), shown in Figure 2 12, this test was performed on the road. During the test many vehicle subsystem parameters were recorded, but only battery pack voltage and current and a few notes of the driving conditions were used. This driving schedule is shown in Figure 2 13. Battery pack current and voltage and total Ampere-hours consumed were sampled at a rate of 1 Hz from a Cruising Equipment Kilowatt-Hour+2 meter[2]. Other parameters such as vehicle speed were recorded at a rate of 1 Hz using the control system's *flogger* (Section 4 6 3) data logger. Because the two systems recorded data to separate files, a duplicated signal, battery pack voltage, was used to synchronize the two data sets.

From the total Ahs consumed an estimate for SOC was made by making the guess that the average total capacity was 2 0kWh. This number was chosen because it placed the SOC at approximately 25% at a point, about 2000 seconds into the test, where the vehicle was noted as seeming to be at a low SOC. With an SOC estimate to correspond to a pack current and voltage reading, each data pair, current and voltage, was assigned an SOC attribute. From this data, polynomial curves were fitted to the "high" and "medium" SOC data sets, see Figure 2 14. Cubic equations seemed adequate to generalize these data sets. The "low" SOC data set resisted a third order polynomial curve that looked like it fit the data. This can be attributed to the relatively few points available for this set. To compensate the "medium" curve was modified to fit the "low" data set. Because of the lack

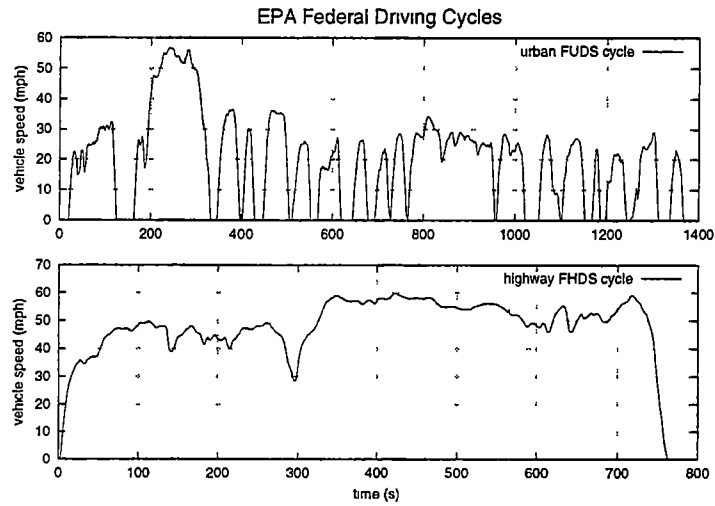


Figure 2 12 EPA Standard Driving Cycles

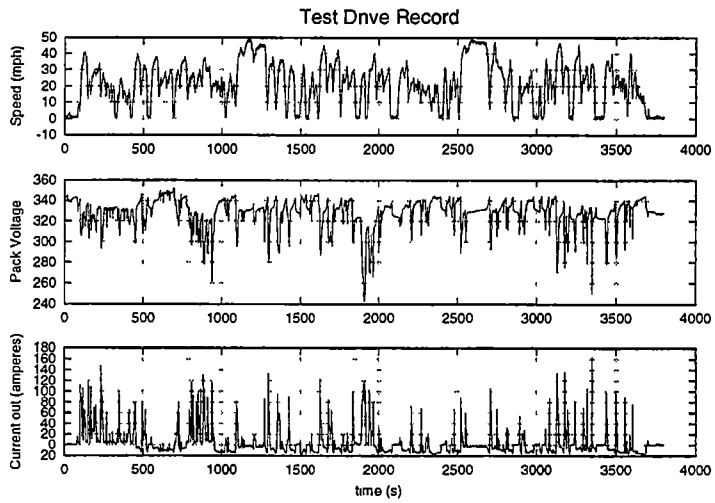


Figure 2 13 A UTK FutureCar Driving Cycle

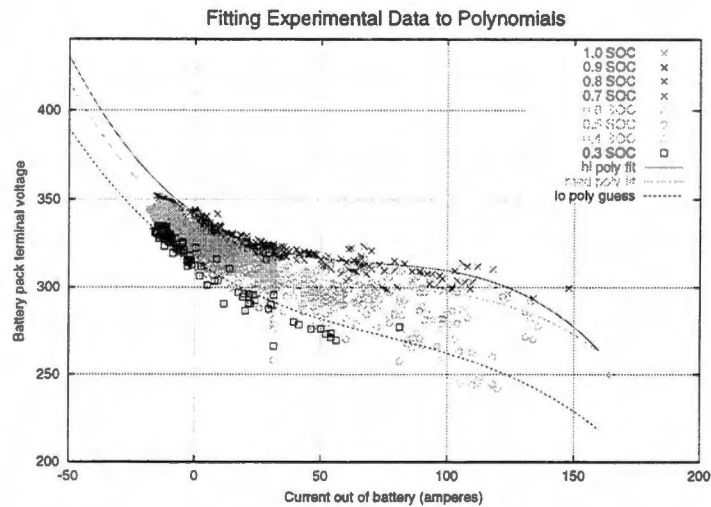


Figure 2.14: Fitting Experimental Data to Polynomials

of data, this curve could be considered a guess at how the pack terminal voltage reacts to current demand while in a low SOC. The values for high, medium, and low are 0.90, 0.50, and 0.25 respectively where a value of 1.0 is 100% SOC and 0.0 is 0% SOC. These three curves then uniquely describe SOC for given current and voltage values if the input values are linearly interpolated between the curve boundaries, see Figure 2.15. It should be noted that the converse to the previous statement is not true; an SOC value does not imply a unique current-voltage pair.

This method of reducing SOC estimation to curves of voltage reaction to current draw is not original. Previous UTK hybrid vehicle research[8], namely Xiaoling He's thesis on hybrid vehicle simulations, used this method to reduce SOC to first order polynomials for a less dynamic battery pack. He's purpose was different, though. The previous research involved simulating hybrid vehicle dynamics, while this study focuses on a hybrid control system. This SOC estimation method extends on the previous model because the battery being modeled undergoes greater discharge currents and at least considers pack charging. These extensions justify the change from a first order, linear model to a third order, cubic model.

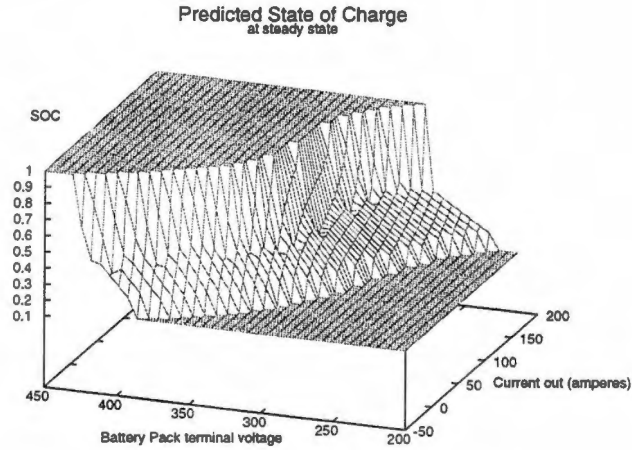


Figure 2.15: A State of Charge Prediction Model

Simulation Results

To test the SOC predictor the original data used to create it was used. While this is not the ideal testing method, lack of a second test set required it. For testing, a single value, err , was calculated to determine the correctness of the model. The values of err are calculated in a least squares sense according to

$$err = \sum (SOC_{measured} - SOC_{predicted})^2, \quad (2.7)$$

where the units of err are unimportant and the values of err are only meaningful when comparing data sets of the same number of samples.

Using the equation graphed in Figure 2.15 to predict SOC and plotting against the measured SOC results in Figure 2.16. This figure shows that there is a significant amount of error in the prediction. The prediction seems to follow the trend of the measured value, but there is an oscillatory element that suggests that the model is under damped. On average it seems to overshoot the desired value as much as it undershoots. Figure 2.17 shows that if the prediction model uses a moving average with a window size of 5 data

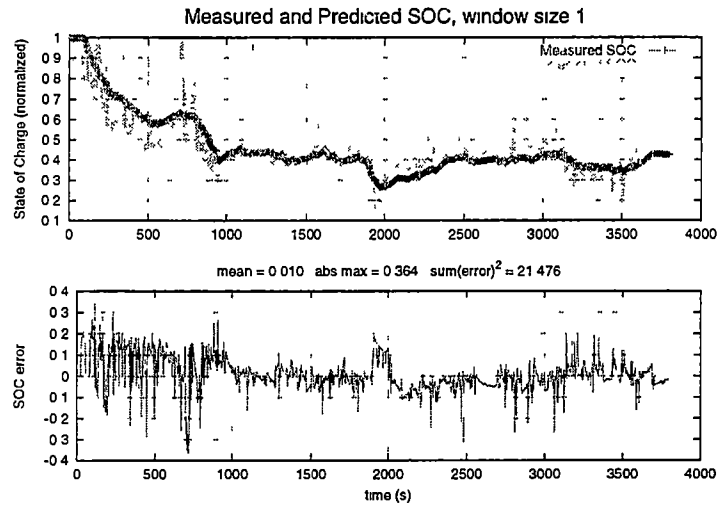


Figure 2.16 Measured and Predicted SOC, window size 1

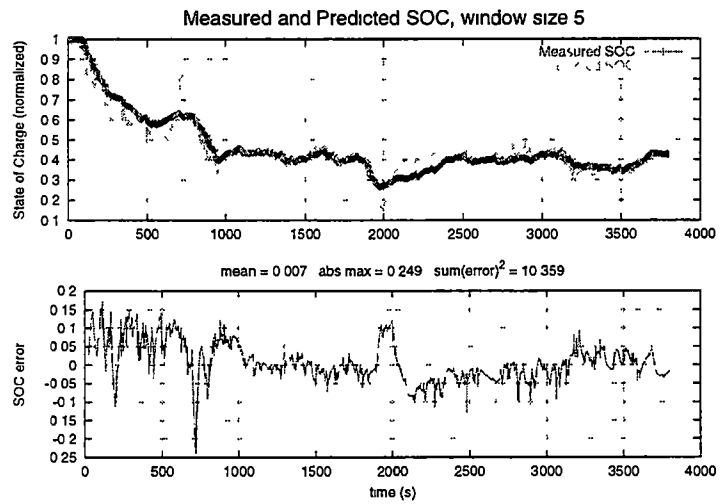


Figure 2.17 Measured and Predicted SOC, window size 5

points, the value of *err* can be halved. Figure 2.18 shows that *err* can be reduced even more with a window size of 10, but Figure 2.19 indicates that a window size of 30 makes the model slow to respond to changes. The optimal window size, the one that minimizes *err* for this data set is 14, shown in Figure 2.20. A summary of the effects of different sized windows is shown in Table 2.5.

Because training and testing on the same data set might lead a generalized model to reproducing the non-general variance in a certain data set, a portion of the training set was used to test the predictor. Figure 2.21 shows how the model predicts only the second half of the training set. It seems to be over damped a bit, taking 200 seconds to catch up to actual value, but otherwise acceptable.

Conclusions

This model of battery SOC works fairly well in simulation. Its strengths include.

- It is conceptually simple and based on actual experimental data and human experience.
- Only two variables are required, pack current and voltage, to estimate SOC. Persistent storage of any data is not required, but averaging over a few past data points is helpful.
- It is simple to modify or adapt to a new environment. For example, when adapting to a different pack, a good first step would be to move the y-intercept of the three equations to correspond to the different nominal pack voltage.

Even with these strengths there are a few weaknesses that follow the weaknesses of expert systems in general[17]. The model does not handle the dynamics of SOC calculation well, the three equations, shown graphically in Figure 2.15, work best under steady state conditions, but can be adapted to a dynamic situation with a moving average scheme. The results are very dependent on the adequacy of the knowledge incorporated into the system.

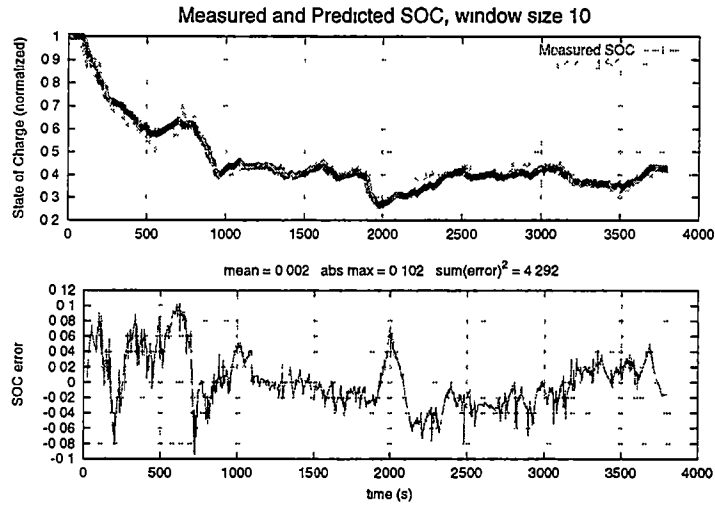


Figure 2 18 Measured and Predicted SOC, window size 10

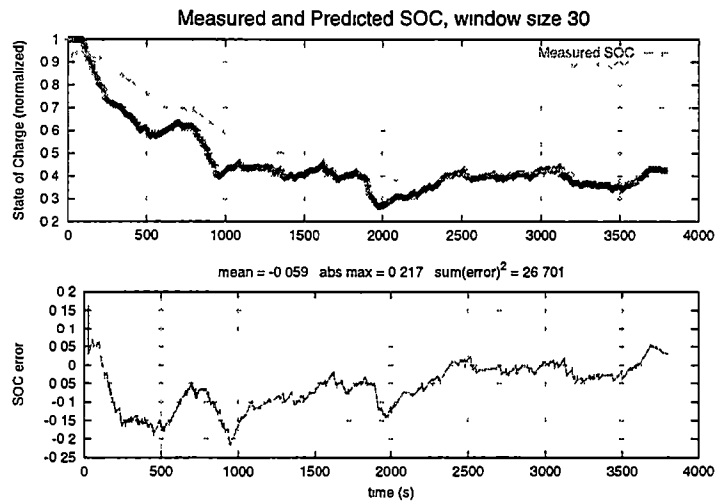


Figure 2 19 Measured and Predicted SOC, window size 30

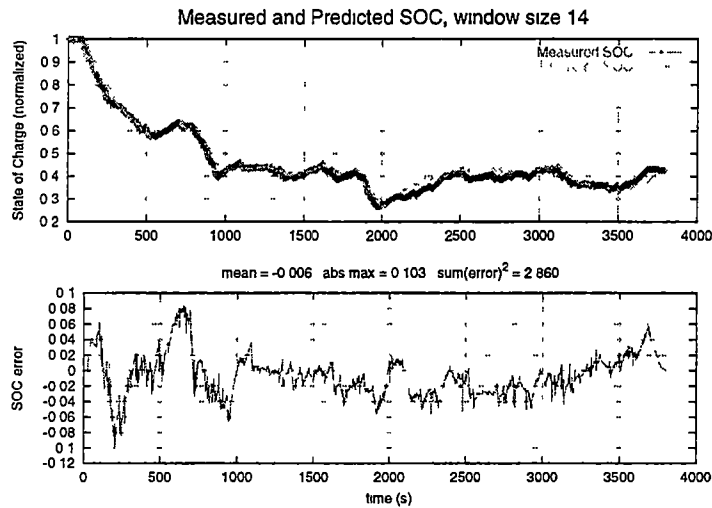


Figure 2.20. Measured and Predicted SOC, window size 14

Table 2.5. Effect of Window size on *err*

Window Size	<i>err</i>
1	21 476
5	10 359
10	4 292
14	2 860
30	26 701

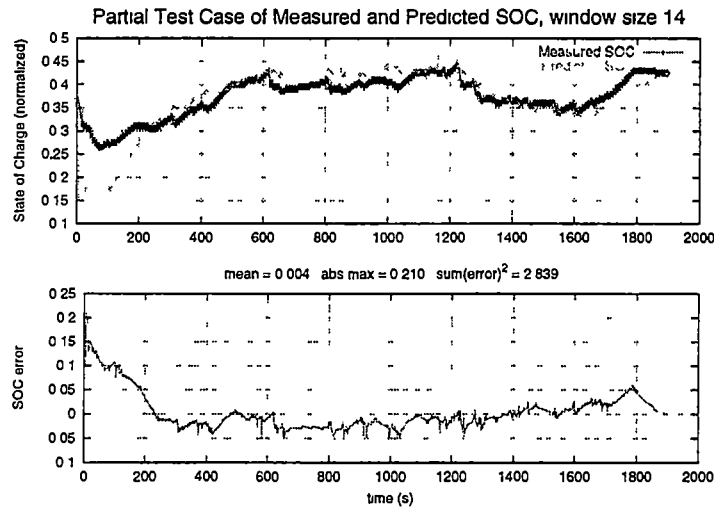


Figure 2.21 Partial Test Case of Measured and Predicted SOC, window size 14

Overall, the method appears to be sound. The implementation of the algorithm is discussed in Section 4.5, and code is in Section D.2.9.

2.5 Software Concepts

To control all of the system controller hardware, software is used. Previous UTK hybrid vehicle projects have used bare Motorola assembly [1] language in the “Ground Up” project and a C language system for DOS [18] in the HEV NEON project. As the UTK projects have grown more complex, more software support is needed to make the most of the computing hardware available. While the attempt was made to use a combination of the QNX operating system and C programming language [9] in the later phases of the NEON project, the FutureCar project is the first to have a system based on it.

2.5.1 POSIX and QNX

QNX is a computer operating system written and distributed by QNX Software Systems Ltd., (QSSL), in Ontario, Canada. It is designed as a real time embedded OS. Real time

features allow for time critical operations to happen deterministically, and embedded features allow the system to be used in a resource constrained environment where it is not readily apparent that a computer is even in use during normal operation. QNX, or more specifically, QNX4, runs on Intel and compatible processors in an environment similar to a regular desktop PC. While newer OSs from QSSL run on a wider variety of hardware platforms, the UTK FutureCar control system is implemented on QNX4 with PC style hardware.

Some of QNX features include multitasking, memory protection, priority driven preemptive scheduling, and fast context switching [15]. Most of these features come from QNX's microkernel architecture. Multitasking means that it supports the execution of multiple processes or tasks in what appears as simultaneous action. While only one process might actually be using the processor at a given instant, the speed of modern computing hardware makes it appear to a human observer that more than one process is running. The term "program" refers to an executable file or image on disk or some other storage. The term "process" or "task" usually refers to the in-memory and runnable image of a program. Memory protection, a feature usually implemented in hardware, is the ability to prevent an unauthorized process from reading or writing memory that has not been allocated for it. Memory protection greatly increases the stability of a system because a single misbehaving user process cannot cause a system crash. Prioritized preemptive scheduling allows processes to be given a priority level. In this scheme a higher priority task may preempt, or temporarily prevent from running, a lower priority task. Context switching refers to the process of the kernel suspending one running task, saving its state, loading another previously suspended task's state, and running this other task.

The microkernel concept is a direct reaction to the traditional UNIX monolithic kernel. In a monolithic system, there are two types of processes, the single kernel process and multiple "user" processes. The monolithic kernel is the operating system core. It manages all the hardware, all device drivers are included in its core. If a user process wants to do

I/O it has to ask the monolithic kernel to do so on its behalf. Also monolithic kernels are usually not preemptible, that is, no task can run at a higher priority than the kernel. Most of these characteristics of a monolithic kernel are a detriment to real time determinism.

The QNX microkernel approach, illustrated in Figure 2.22, moves every service out of the kernel that can be done from "user" space. What remains in the kernel are only the lowest level services. An advantage to the microkernel design is that procedures that in a monolithic kernel would have no memory protection are now in separate memory spaces. This reduces the possibility of a crash due to a single faulty service. The disadvantage is the same as the advantage, memory protection. In the monolithic design, kernel internal communication can be as simple as passing variables. When services are moved to separate processes in separate memory spaces, special constructs must be made so that the independent processes may communicate. QNX uses message passing as its native form of interprocess communication, IPC. One of the primary responsibilities of the microkernel is to handle the routing of all messages.

QNX is a POSIX conforming operating system. POSIX, the *IEEE Portable Operating System Interface Computing Environments* [11], is both an American, *ANSI*, and International, *ISO*, Standard. POSIX defines a standard way for computer applications to obtain information and services from the underlying operating system. An application that follows these standardized interfaces is considered portable, that is, it can easily be reconfigured or ported to run on a different operating system easily. The basis of POSIX is a combination of the behavior and interfaces of AT&T UNIX System V and Berkeley Standard Distribution UNIX. POSIX is not an operating system itself, but it does define a set of standards. Because the standards only define the external characteristics, a system like QNX which is very dissimilar to traditional UNIX internally, may still follow these guidelines. Some of the POSIX standards that QNX conforms to are.

P1003.1 defines the interface between portable application programs and the operating system, based on historical UNIX system models. This consists of a library of functions

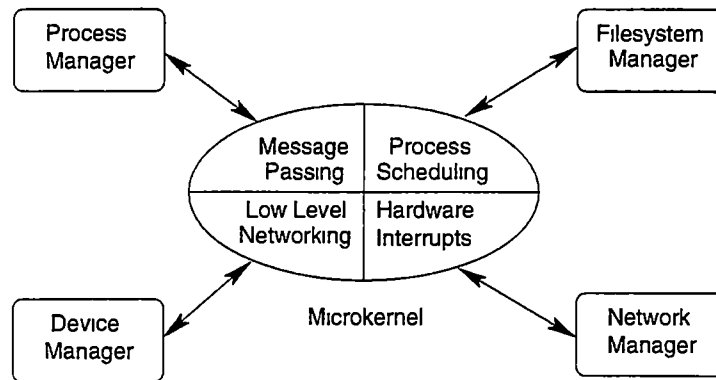


Figure 2 22 QNX microkernel architecture

that are frequently implemented as system calls Approved as IEEE Std 1003.1-1990.

P1003.1b real time extensions including binary semaphores, process memory locking, memory mapped files and shared memory, and priority scheduling. Approved as IEEE Std 1003 1b-1993

P1003.2 specifies a shell command language based on the traditional UNIX Bourne shell with features from the C and Korn shells Additionally specifies over 70 utilities that can be used in shell scripts or interactively Approved as ISO/IEC 9945-2:1993 and IEEE Std 1003 2-1992.

2.5.2 Multiple Tasks

Most real time applications are defined by the requirement to handle multiple jobs at once in a timely manner The ability of an operating system to handle multiple tasks, each running at a different loop frequency, concurrently is a definite advantage These advantages[4] include

Scalability Multiple tasks can run on multiple processors If a control application's requirements are best served with more than one CPU, multiple tasks take advantage of the computing hardware

Modularity Processes can easily be added and removed to adjust to new requirements.

Protection In QNX, processes are in separate memory spaces. This means that a single errant task cannot corrupt the memory of another. A single bad instruction has a much smaller chance of taking the whole system down with memory protection.

QNX follows the traditional UNIX semantics for task creation.

Fork and Exec

Task creation in QNX is performed with the C language function `fork()` as specified by POSIX. The historical UNIX method of starting a program is done in two steps. For example, as shown in Figure 2.23, if one process, A, wants to start another program, B, whose executable image is stored on disk, it first calls the `fork()` function which makes a copy of process A. In the aptly named `fork()` function, a single process enters and two return. The return value uniquely identifies which process is which. The two copies of process A exist in separate memory spaces. The newly created copy of process A then calls another function, `exec()`, to load the executable image of B. The sole purpose of the `exec()` function is to replace the calling process with a process that results from executing the B program. After the `exec()` function call, what was two copies of process A becomes a single A process and a single B process. Since the traditional procedure for starting a new executable includes making an unnecessary copy of a process, QNX provides a more efficient set of `spawn()` functions for loading executables from disk. The advantage to the traditional method is that the `exec()` step does not have to be done. A single executable program may “expand” to multiple copies of itself, each performing a different task. This method is generally termed the fork threading model, but it is a misnomer since a “thread” usually refers to an independent part of a process that operates in the same memory space as its creator. All process created with `fork()` are in separate protected memory spaces.

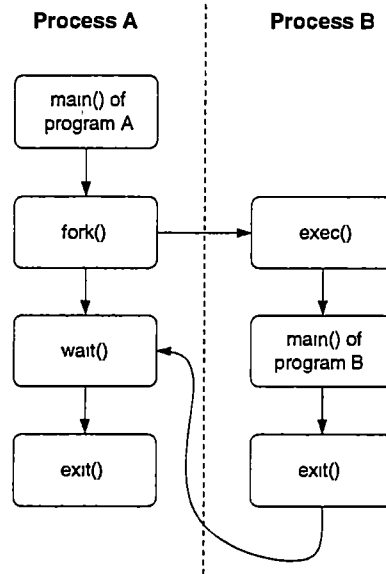


Figure 2.23 UNIX Process Creation

The Tree Process Model

The POSIX process model is arranged like a tree. That is, every process except the first one, has a parent. Any process can be a parent, all that is required is that it have at least one child process. A process can be both a parent and a child. A child process can have a sibling process if they share a common parent. The lineage of a system's processes fan out like the branches of tree, shown in Figure 2.24. There is a special relationship between parent and child processes. When a child process finishes, either normally or abnormally, the parent process is notified. The parent may find out how a child process exited by calling one of the `wait()` functions. If a child process dies and the parent does not call `wait()` the child process remains in an altered state. A process in this state is called a "zombie" because it is neither running nor completely finished. Another situation is when a parent process dies before its children. In most cases where certain provisions have been made, any child processes will be terminated along with the parent. In other cases, the child processes are reparented or "adopted" by the special first process. This technique is used by long

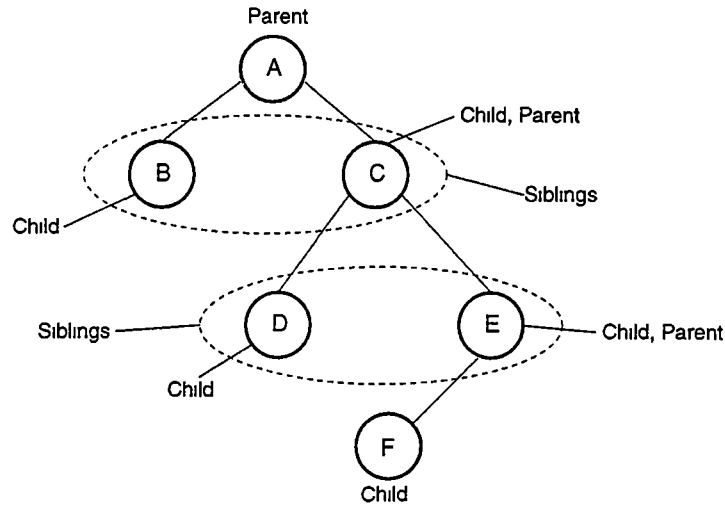


Figure 2.24: UNIX Process Tree Relationship

running processes, termed “dæmons”, to break out of the process tree. When referring to a special UNIX process, dæmon is usually spelled the old-fashioned way to relate to the term’s original meaning, a guardian spirit, rather than the current definition of an evil entity or devil.

2.5.3 Interprocess Communication

Since all processes operate in separate memory spaces there needs to be some provision for transferring or sharing information. QNX supports many forms of IPC, the primary being message passing. With QNX message passing, processes communicate with functions named Send, Receive, and Reply (S/R/R), see Figure 2.25. This form of IPC is synchronous because the two communicating processes wait for responses from each other. QNX also offers an asynchronous version of message passing where confirmation is not required. POSIX standardized forms of IPC such as shared memory, signals, and pipes are also supported by QNX.

Shared memory is a descriptive term for a form of IPC in which a section of memory is shared between multiple processes. A protected memory based operating system such as

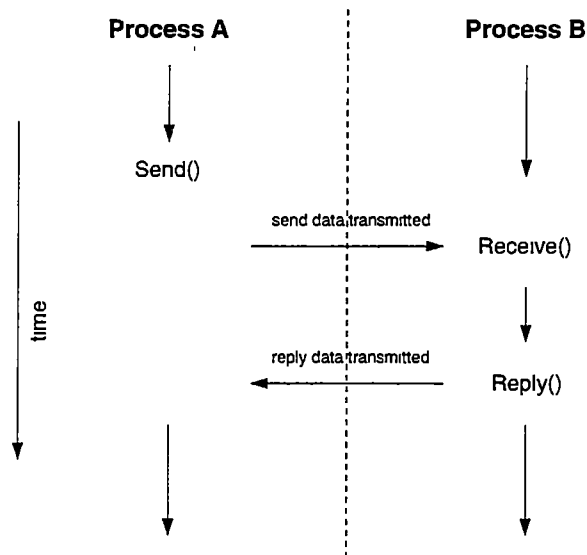


Figure 2.25 QNX Send-Receive-Reply IPC Mechanism

QNX isolates the memory spaces of each process. The shared memory mechanism puts a window in the wall that separates processes. Shared memory looks to a running process just like non-shared memory. Since shared memory is basically regular memory, there is no inherent synchronization for accesses to shared memory. Other forms of synchronization like semaphores or mutexes can be used in conjunction with shared memory. Shared memory is a very fast form of IPC mostly because it is so primitive.

Signals are a traditional UNIX form of IPC. Signals are used extensively internally in a POSIX system: process termination is induced by SIGTERM, invalid memory references cause the delivery of SIGSEGV, a child termination notice is indicated by SIGCHLD, and SIGFPE is sent as an indication of a math error (division by zero, overflow). Only two signals are reserved for application specific uses. Signals are also not queued. These limitations generally make signals a poor choice for a user defined IPC scheme. A specification in POSIX 1003.1b, real time signals, removes these two limitations, but QNX4 does not support this feature.

Pipe IPC is one of the characteristic features of UNIX. Pipes are so named because they

open a one-way communication channel between two separate processes. A single pipe can only be used with two processes. One process writes to its end while the other only reads from it. Multiple pipes can be used to connect more than two processes. While pipe data travels through the monolithic kernel in a traditional UNIX system, QNX implements pipes with a Pipe I/O manager. This manager translates pipe function calls to QNX S/R/R messages.

2.5.4 Real Time Features

The definition of a real time system can be quite broad [10]. At the most basic level, a real time system is one that meets a specified temporal deadline on time. That definition would hold for both a real time stock ticker system and a real time anti-lock braking system (ABS) for an automobile. While the time deadline for a stock ticker may be anywhere from a few seconds to a few minutes, the deadline for an ABS might be in the microsecond range. To distinguish between the two extremes, the terms "hard" and "soft" are used to qualify the term "real time", but even these terms are really only relative. It might be possible that an assembly line robot must synchronize with the rest of the assembly line at a hard real time rate of 10 Hz and a video game might need to synchronize sound and video at a soft real time rate of 60 Hz. Since the speed of operation might not best determine if a system is hard or soft, another qualification is added to hard real time systems. A hard real time system is one that must meet its deadline on time or there is a high risk of loss of life or property.

QNX was designed with hard real time in mind. To that end it supports many features necessary to reach temporal goals. First, the small microkernel features very fast task switching and interrupt handling. These functions are the main responsibilities of the QNX kernel since most other services have been moved to user space tasks. Also, since the kernel itself is never scheduled to run, it can "get out of the way" of demanding hard real time tasks.

Another feature of QNX from the POSIX 1003.1b specification is process priority levels. Prioritizing allows more important tasks to preempt less important tasks. In QNX a lower priority task will not run if there is a runnable higher priority task as long as some sort of shared resource doesn't cause a priority inversion. A priority inversion happens when a low priority task is able to prevent a higher priority task from running. For example, tasks of different priorities might both use a semaphore. If the low priority task obtains the semaphore, it can prevent the higher priority task from running. Priority inversion is usually undesirable.

In addition to a priority parameter, processes can also have a scheduling method specified.

- **FIFO scheduling** Under FIFO, First In First Out, scheduling a process will continue to run until it either voluntarily relinquishes control or is preempted by a higher priority task. Real time tasks that cannot afford to be interrupted are best run with FIFO scheduling.
- **Round-Robin scheduling** Round-robin scheduling is the same as FIFO scheduling except that a process may also be preempted if it consumes more than a single time slice of CPU time. In QNX a time slice is defined as 100 milliseconds.
- **other scheduling** While there are well defined POSIX specifications for FIFO and round-robin scheduling, the "other" scheduling method is intentionally vendor specific. QNX defaults to an adaptive scheduling method where priorities are adjusted in response to CPU use. While adaptive scheduling is ideal for an interactive multi-user system, it cannot guarantee response times necessary for hard real time applications.

The combination of a responsive microkernel, priority levels, and scheduling allows QNX to be used reliably in hard real time situations with properly written software.

Chapter 3

Control Algorithms

The UTK FutureCar has two main operating modes, ZEV and HEV. In ZEV, Zero Emissions Vehicle, mode the vehicle acts like a purely electric vehicle. The on-board battery pack is not recharged, hence, the control algorithm is considered to be “charge depleting.” The Unique Mobility SR218 traction motor is the only torque providing drive train component. In HEV, Hybrid Electric Vehicle, mode all drive train components optionally come into play. If the SOC is high, HEV mode acts similarly to ZEV mode in that the engine is not operational and the traction motor provides all of the required driving power. If SOC is low or the driver torque request is significantly high, the engine is operational along with the generator. Because the engine/generator combination is used to restore the battery pack, HEV mode is a “charge sustaining” mode. The main efficiency improvement over a conventional vehicle comes from operating the engine only when needed.

3.1 ZEV Mode

In ZEV mode, the requested torque is directly proportional to the position of the driver’s accelerator pedal. The traction motor, as shown in Figure 2.9, acts as a constant torque device at low rotational speeds and as a constant power device at higher speeds. To

linearize the traction motor torque the maximum torque available must be found. If the motor speed is less than the transition speed, ≈ 2100 rpm, the maximum motor torque in units of N m is

$$T_{max} = 240 \quad (3.1)$$

Otherwise the maximum motor torque is given by

$$T_{max} = \frac{P_{max}}{\omega_{motor} \left(\frac{\pi}{30}\right)}, \quad (3.2)$$

where $P_{max} = 53,000$ Watts and ω_{motor} is measured in rpm. With the maximum available torque known the torque request in N-m for the motor is

$$T_{req} = \min\left(\frac{accelpedal}{r_{wheel}}, T_{max}\right), \quad (3.3)$$

where *accelpedal* is interpreted in units of N m and

$$r_{wheel} = \frac{T_{wheel}}{T_{motor}} \approx 7.59 \quad (3.4)$$

Ideally the generator should not operate in ZEV mode, but due to speed limitations of the over-running clutch, the generator must be operated when the vehicle travels at speeds higher than 55 mph. The desired speed of the generator is -3000 rpm while the vehicle is traveling at or over 55 mph where the negative sign indicates a reverse rotational direction. There will be a hysteresis with a width of 10 mph which is used to determine when the generator should be turned off. That is, when the vehicle speed drops below 55 mph, the generator will not be turned off until vehicle speed has dropped below 45 mph. In pseudo-code the algorithm for controlling the generator while in ZEV mode is given in Figure 3.1.

Traction motor braking should be directly proportional to brake line pressure. This should allow some of the energy lost during vehicle braking to be recovered and stored in

```

if (vehicle speed > 55 mph) and (generator is not spinning)
  enable generator
  set generator desired speed to -3000 rpm
  set generator regenerating level to zero
end

if (vehicle speed is < 45 mph) and (generator is spinning)
  disable generator
end

if vehicle speed is between 45 mph and 55 mph
  maintain the current state of the generator
end

```

Figure 3.1 ZEV generator algorithm

the battery pack

Proportioning the traction motor while in reverse follows the same rules as ZEV with the exception that the motor direction is reversed. Precautions should be made to prevent any drive train damage due to switching traction motor direction at speed.

3.2 Hybrid Mode Fundamentals

3.2.1 Throttle Position

During the course of drive train component testing, very little data on the CNG converted engine was collected. To compensate, the influence of throttle position on engine torque is estimated using a torque correction factor. This factor is multiplied by the maximum possible torque for a given engine speed. The equation

$$C_{torque}(throttlepos, \omega_{eng}) = 1 - e^{-[throttlepos - k_2 \left(\frac{\omega_{eng} - 800}{800}\right)]k_1}, \quad (3.5)$$

where ω_{eng} is measured in rpm, $throttlepos$ is a percent value and the values

$$k_1 = 10, k_2 = 0.05 \quad (3.6)$$

describe this torque correction factor. The constant k_1 modifies the sharpness of the curve and k_2 adjusts the influence of varying engine speed on the torque correction factor. This equation is illustrated in Figure 3.2 for four engine speeds, 800, 1200, 2000, and 4000 rpm. Torque correction factors are bounded by 1 and zero. With this correction factor, the experimental data for the CNG converted engine with wide open throttle can be manipulated into an estimated engine torque map, shown in Figure 3.3. This map is used to estimate engine torque output from engine speed and throttle position. The engine map shows that there is a local maximum torque output for a given speed and throttle position. It will be assumed that the point of maximum torque output corresponds to the point of maximum operating efficiency for a given speed. The speed at which torque is a maximum depends upon throttle position. For a given engine speed there exists a specific throttle position which will optimize engine efficiency. The relationship between engine speed and throttle position can be found by optimizing engine torque with respect to speed and solving for throttle position.

3.2.2 Throttle-Up Procedure

The throttle-up procedure is used during re-engagement of the engine from idle to a working throttle position. Proper re-engagement of the engine is critical. Improper re-engagement of the engine can result in shock loading and possible breakage of transmission components, primarily the over-running clutch. The over-running clutch connects the engine crank shaft to the transmission. The inner part of the clutch is attached to the engine, and the outer part is connected to the transmission. This arrangement allows the outer portion of the clutch to spin faster than the engine crankshaft. This, in turn, prevents the generator from driving the engine faster than it would normally rotate. This setup may present a problem if, during the throttle-up, the engagement of the engine occurs too quickly. If the engine is brought from idle to operating speed too quickly, the over-running clutch may be shock loaded beyond its capacity. In order to avoid this, throttle position will be increased as a

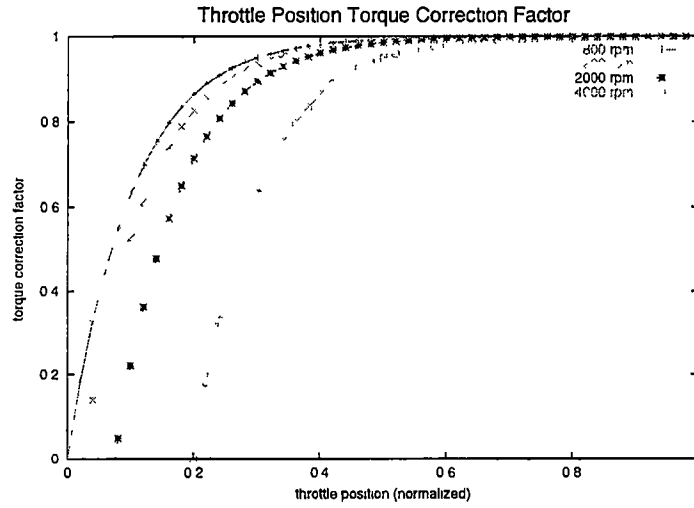


Figure 3 2 Throttle Position Torque Correction Factor

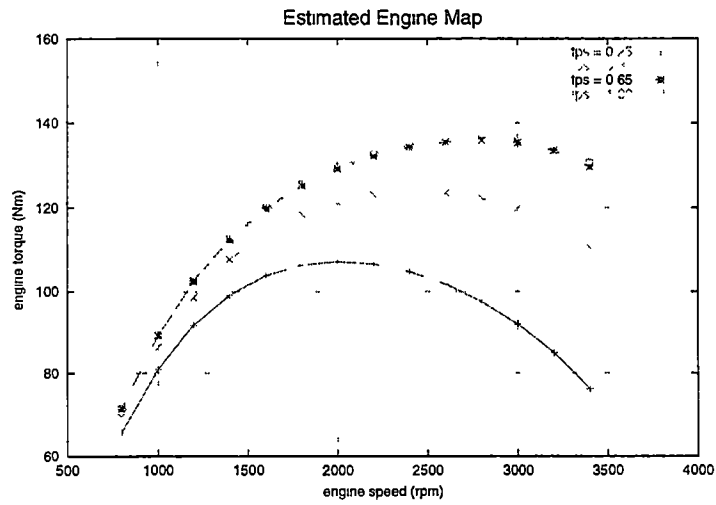


Figure 3 3 Estimated Engine Map

function of time during engine engagement. The equation governing the rate of increasing throttle is

$$throttle(t) = throttle_{final} \frac{\alpha^{\frac{\beta}{t_f} t} - 1}{\alpha^\beta - 1}, \quad (3.7)$$

where

$$\alpha = \frac{throttle_{max} + 1}{throttle_{max}} \quad (3.8)$$

and $throttle_{final}$ is the final working throttle position, t_f is the time duration of the throttle-up procedure in seconds, and the two factors, α and β , affect the shape of the curve. The factor β changes the sharpness of the bend. A small β makes the function more linear, and a large β increases the bend in the function. Figure 3.4 shows throttle position for three values of beta, 1, 7, and 14. During throttle-up the following will take place

- The generator will be spinning with the desired speed equal to the actual speed with zero torque on the shaft. The engine will be operating at idle, throttle position = 0. The outer portion of the over-running clutch is spinning faster than the engine crankshaft.
- The engine throttle will be increased as a function of time, following Equation 3.7. During this time, engine speed will be increased while there is no load on the engine.
- At some time, the engine speed will match the speed of the outer portion of the over-running clutch. This represents the engagement of the engine into the drive train.
- With the engine now engaged, torque is produced by both the engine and the generator. Engine speed will increase, and the absolute value of the generator speed will decrease and deviate from the desired speed. Engine torques will increase as the throttle is opened, and generator torque will increase as deviation of actual speed from desired speed increases. This transient behavior will continue until the torque of the engine and generator are balanced and throttle position has reached the maximum working position.

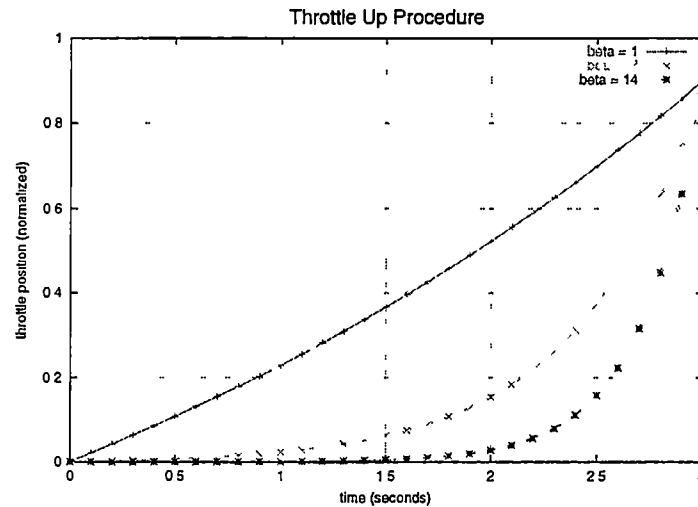


Figure 3.4 Throttle Up Procedure

- The throttle-up procedure will be complete when the actual throttle position is equal to the maximum working position

3.2.3 The Generator

In order to maintain quasi-static equilibrium and control within the gear train while the engine is running, the generator torque and engine torque must be linearly related by the gear ratio below

$$\frac{T_{eng}}{T_{gen}} = \frac{27 + 57}{27} \approx 3.111 \quad (3.9)$$

From the manufacturer's data, Figure 2.8, a torque-speed map for the generator can be estimated by a simple linear relationship. Above 6000 rpm the available torque from the generator quickly drops off. Because of this, the speed operation of generator must be limited to values between -6000 rpm and 6000 rpm. If the generator is operating outside of ± 6000 rpm, its torque will be considered zero. Speed dependent efficiency can be approximated as changing linearly from 1000 rpm to 6000 rpm. At 6000 rpm the efficiency is approximately 90% and at 1000 rpm, approximately 60%.

Since the generator is speed controlled, torque is produced by the generator when the actual speed of the generator is different from the requested speed. If the generator is converting mechanical power to electrical power, then a difference of 800 rpm or greater between the desired speed and the actual speed will cause the generator to produce full negative torque as shown in Figure 3 5

3.2.4 Electric Take-off

At slow speeds, less than 5 mph, the ratio between generator speed and engine speed is high, ≈ 3.11 . The engine speed must therefore be kept low to protect the generator from spinning too fast. This will be accomplished by trimming engine throttle to idle when the vehicle is not moving, and increasing throttle as vehicle speed increases. This requires that the main traction motor provide all of the torque required during low vehicle speed. Figure 3 6 shows generator speed given vehicle speed for four different engine speeds. Clearly, if the generator is to be maintained within a safe operating region, engine speed must be lower than 2000 rpm when the vehicle is nearly stationary.

To enforce this limitation on engine speed, throttle must be trimmed during low vehicle speeds as shown in Figure 3 7

3.3 Hybrid Mode

The equations graphed in Figure 3 8 are used during hybrid mode to control drive train component speeds based on vehicle speed. The generator slows down during increasing vehicle speed. Since the generator speed drops to an inefficient operating region after 60 mph, its regeneration capabilities are greatly compromised. To compensate the vehicle has two "regimes" of operation. In the transition from the "low speed" regime to the "high speed" regime the generator has to switch its direction of rotation. After the switch, the generator speed increases with an increase in vehicle speed. The generator presumably has

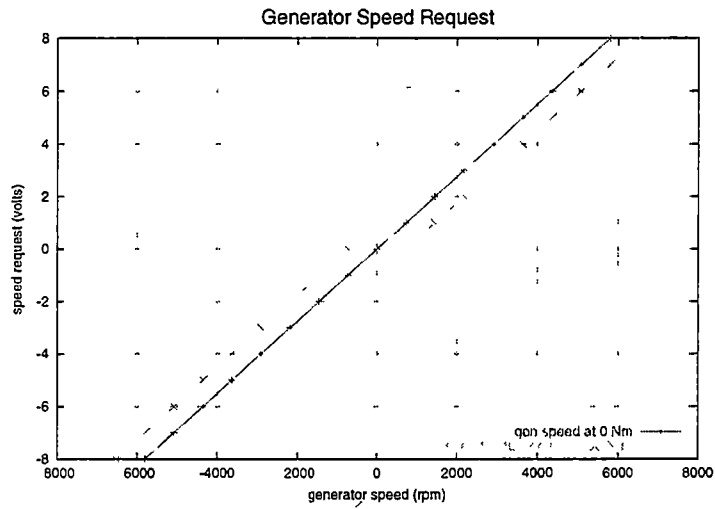


Figure 3.5 Generator Speed Request and Torque Relation

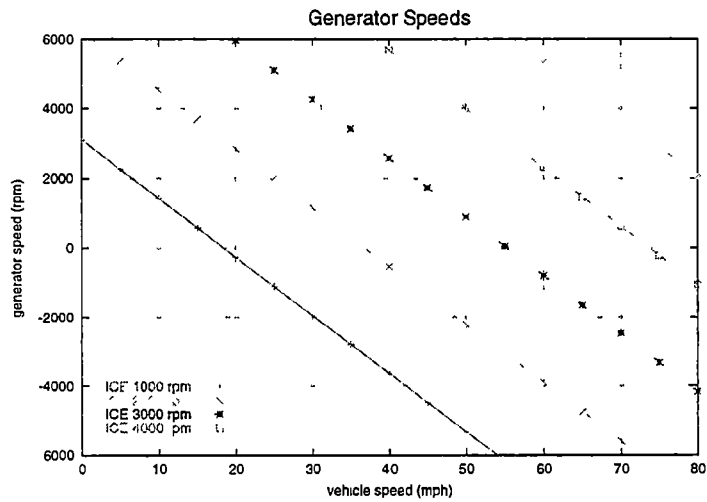


Figure 3.6 Generator speed as a function of vehicle speed and engine speed

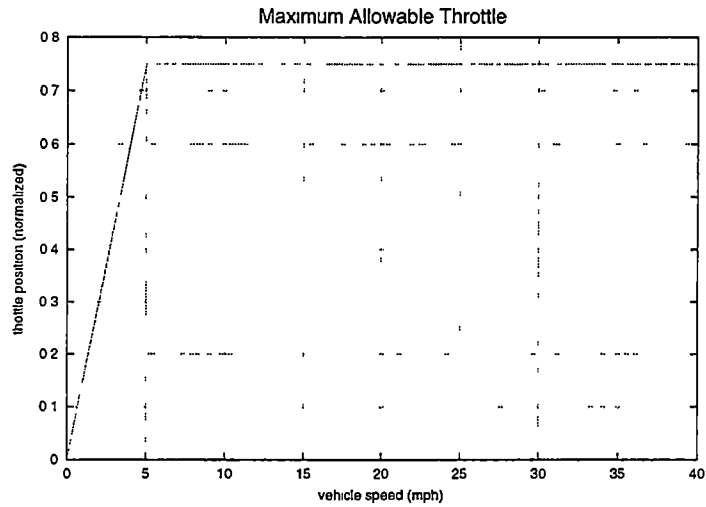


Figure 3 7 Maximum allowable throttle for given vehicle speed

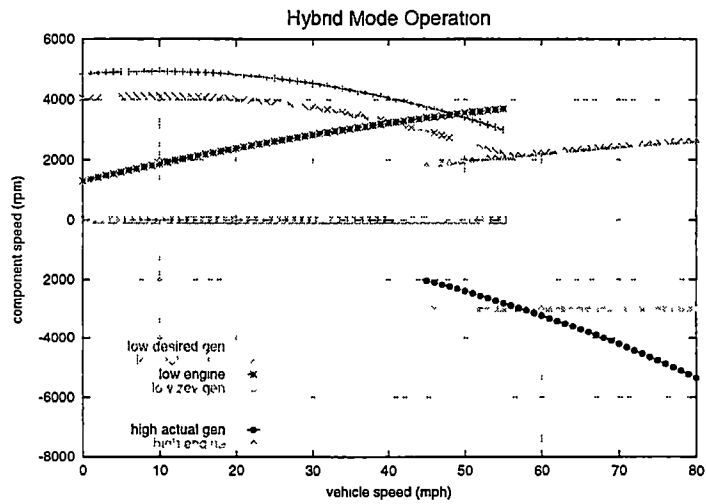


Figure 3 8 Hybrid mode drive train component speeds

enough torque capability to force the engine to a certain speed regardless of engine throttle position and speed. The gear ratio relating the two is

$$\frac{T_{eng}}{T_{gen}} = \frac{84}{27} \approx 3.111 \quad (3.10)$$

The generator can produce at least 48 N·m of torque throughout its operating range, and the engine can produce about 140 N·m at its peak. Because of the gearing, the effective generator torque that the engine could “feel” is about 150 N·m, (48*3.111)

The generator desired speed command will be the primary method of drive train control. The following steps will be used

- Measure vehicle speed
- Calculate desired generator speed based on vehicle speed
- Request this speed from generator while also requesting full regeneration.
- Monitor the engine speed to insure that it is operating within the desired range
- Determine torque on the generator based on generator current readings.
- Measure driver request for torque from accelerator pedal.
- Calculate and request torque from the traction motor that will cause the traction motor to transmit or absorb power to or from the wheels

The accelerator pedal responsiveness is ideally constant whether the operational mode is HEV or ZEV. Since there is a greater amount of torque available to the drive wheels when in HEV mode, the accelerator pedal needs to be desensitized. One way of doing this is to infer the additional torque available from the generator current signal. The torque request for the traction motor would then be

$$T_{motreq}(Nm) = T_{maxreq} - T_{driverreq} - I_{gen} \cdot k_t \cdot k_g \quad (3.11)$$

where all torque units are N m and I_{gen} is in units of amperes. The constant k_t in units of $\frac{Nm}{A}$ is an estimation of generator torque for a corresponding generator current. The constant k_g is dimensionless and represents the generator to wheel gear ratio

3.3.1 Hybrid Regime Transition

Different control equations are used for the two different regimes of vehicle speed. The greatest difference between the two is the change in direction of rotation of the generator. There is also a change of about 2000 rpm in the speed of the engine when crossing from one regime to the other. There is one set of equations for low vehicle speed (0-55 mph)

$$\omega_{gen\ desired\ low} = a_2(V_{spd})^2 + a_1(V_{spd}) + a_0 \quad (3.12)$$

where V_{spd} is the vehicle speed in mph, $\omega_{gen\ desired\ low}$ is in units of rpm, and the constants are

$$a_2 = -0.933, \quad a_1 = 17.82, \quad a_0 = 4044 + 800, \quad (3.13)$$

and

$$\omega_{eng\ desired\ low} = c_2(V_{spd})^2 + c_1(V_{spd}) + c_0 \quad (3.14)$$

where V_{spd} is the vehicle speed in mph, $\omega_{eng\ desired\ low}$ is in units of rpm, and the constants are

$$c_2 = -0.3, \quad c_1 = 60, \quad c_0 = 1300 \quad (3.15)$$

The set of equations for high vehicle speed (45-80 mph) is

$$\omega_{gen\ desired\ high} = b_2(V_{spd})^2 + b_1(V_{spd}) + b_0 \quad (3.16)$$

where V_{spd} is the vehicle speed in mph, $\omega_{gen\ desired\ high}$ is in units of rpm, and the constants are

$$b_2 = -0.777, b_1 = 2.264, b_0 = -650 - 800, \quad (3.17)$$

and

$$\omega_{eng\ desired\ high} = d_2(V_{spd})^2 + d_1(V_{spd}) + d_0 \quad (3.18)$$

where V_{spd} is the vehicle speed in mph, $\omega_{eng\ desired\ high}$ is in units of rpm, and the constants are

$$d_2 = -0.25, d_1 = 55, d_0 = -180 \quad (3.19)$$

There is a 10 mph overlap between the two regimes to prevent the generator from cycling directions quickly. The algorithm for switching from the low regime to the high regime is

- Set engine throttle to idle
- Wait for throttle to come to idle position.
- Set the generator desired speed to zero
- Wait for generator to spin down
- Set generator request to operate in high regime (Equation 3.16)
- After generator speeds up again, begin the throttle-up procedure

The algorithm for switching from the high regime to low is basically the same except that the desired generator equation is Equation 3.12.

3.3.2 Transition from ZEV to HEV Mode

There are both high and low operating regimes for ZEV and HEV operating modes. The low end regime for ZEV is defined by having the generator disabled, and the high end is defined by having the generator enabled and spinning at -3000 rpm. The range and overlap of the

high and low regimes of ZEV and HEV modes are identical. This fact greatly simplifies transition between the two modes. If the vehicle is traveling at a speed less than 45 mph and there needs to be a mode switch from ZEV to HEV, the procedure to follow is

- Set the generator speed to the low regime desired speed.
- Set the generator regen level to full
- Insure that the generator has reached the desired speed
- Start the engine
- Begin the throttle-up procedure

This makes the assumption that the driver will continue at a speed less than 55 mph. If this assumption is incorrect, a transition from low regime HEV to high regime HEV will need to be made soon after the ZEV to HEV transition.

If the vehicle speed is in the transition region, between 45 and 55 mph, and operating in high end regime ZEV, the procedure for switching to HEV is

- Set the generator desired speed to the high regime speed
- Set the generator regen level to full.
- Insure that the generator has reached the desired speed.
- Start the engine
- Begin the throttle-up procedure

This anticipates that if the vehicle is operating at high speed in ZEV mode, the driver will most likely continue to travel at high speed in HEV mode. If the generator direction is already in the correct direction for high speed operation, another transition, from low regime HEV to high regime HEV, is not required. Similarly, if the vehicle speed is between

45 and 55 mph and the operational switch is from low regime ZEV to HEV, the generator should be set to follow the low regime path

Chapter 4

Software Details

4.1 Overview

The code to implement the FutureCar dual hybrid electric vehicle system controller is all written in the C programming language with the QNX supplied Watcom compiler. The code is broken down into two function libraries, two control programs, and a few monitoring and diagnostic programs. To manage the complicated build process of all of the code the *make* [14] utility is used. As such, there is a special file, a *Makefile*, for each program, library, and subdirectory. Each software component is designed to be able to be built independently of all other components save for library dependencies, but there is also a top level Makefile that allows for building the complete system.

4.2 IPC Library: libfclient

A library of functions, named libfclient, provides interprocess communication services for all control and monitoring applications. Libfclient provides wrapper functions for accessing a single shared memory segment. The shared memory segment has provisions for saving 3600 snapshots of all the time varying variables. If the data is saved every second, this amounts

to the previous hour's information. The shared memory segment amounts to approximately 800 kilobytes.

Shared memory was chosen because it is the fastest and most efficient method of sharing a large data set among multiple processes. Accesses to shared memory are inherently asynchronous so processes that need to synchronize need to do so manually. Also, the asynchronous nature of shared memory means that it might be corrupted if two processes try to write to the same shared memory. More specifically, memory may become corrupted if a shared variable is only partially written when the writer process is preempted. One way to prevent corruption would be to use one or more semaphores to guarantee mutual exclusion, but this can make things more complicated and may slow performance if the number of semaphores is not sufficient. To prevent corruption of shared memory, the `fcntl` library uses only native sized variables. Under QNX on i386 Intel-like hardware the native size is 32 bits or 4 bytes, the size of an integer or single precision floating point number. By using 32-bit sized variables only, all writes are guaranteed to be atomic operations. Also all `fcntl` applications use FIFO scheduling to prevent unnecessary preemption. It is possible that a certain write operation may be preempted so that a write of multiple values may not be an atomic operation. This would not corrupt single values, but the time from which a certain value was last updated might not be the same as the rest. At a high enough refresh rate, this difference becomes negligible.

4.2.1 Shared Memory contents

The contents of the shared memory segment are defined in a header file, `fcvar.common.h`, listed in Appendix E.1.3. There are many sections to this data structure. The naming convention for structure members that represent values read from the hardware is a single character prefix, either an 'i' or an 'o' which distinguishes between input and output values relative to the system controller and a mixed case description. The naming convention for structure members that are used to convert or calibrate voltage readings to engineering

units follows that of the hardware values with the addition of a suffix, either 'Sen' or 'Off', that is used to distinguish whether the value is a linear slope sensitivity value or a linear y-intercept offset. Digital hardware values follow the same naming convention as the analog hardware values. Table 4.1 lists the contents of the shared memory values in detail.

The shared memory segment is arranged to work as a circular buffer. Periodically the active section of the structure is advanced and the calculated values, digital hardware values, and analog hardware values are stored in a history.

4.2.2 libfclient functions

For an application to use the fclient library it has to include the file `fclient.h`, shown in Appendix E.1.2. This header file gives prototypes for the functions available in the library.

`fclient_create_shm()`

The function `fclient_create_shm()` takes no arguments and returns a pointer to the newly created shared memory segment. This function should only be used in a single application designated as the manager of the shared memory segment. The function removes any previously present segment, creates a new one, and initializes it to contain all zeros. If the function fails it returns a value of `NULL`.

`fclient_delete_shm()`

The function `fclient_delete_shm()` takes no arguments and returns an integer to indicate whether the operation was successful. Because a shared memory segment may exist after the process that created it exits, this function allows for the orderly removal of the shared memory used. A value of 1 indicates that the function failed while a value of zero indicates success.

Table 4 1 Partial listing of fclient shared memory contents

Member Name	Data Type	Description	Units
SOC	float	battery state of charge value	normal
SOCcat	integer	battery state of charge category enumeration	-
mode	integer	vehicle operation mode enumeration value	-
vehicle_speed	float	vehicle speed	mph
vehicle_distance	float	vehicle distance traveled since power-on	miles
kwh_volts	float	battery pack voltage measured from kWh meter	volts
kwh_amps	float	battery pack current measured from kWh meter	amps
IceReqState	integer	requested state of engine enumeration value	-
IceState	integer	actual state of engine enumeration value	-
running	integer	control program active indicator	-
hybrid_regime	integer	hybrid mode indicator	-
hybrid_transition	integer	hybrid sub-mode indicator	-
iEmMotorSpeed	float	traction motor speed feedback	rpm
iGenMotorSpeed	float	generator motor speed feedback	rpm
iIceEngineSpeed	float	engine speed feedback	rpm
iAccelPedalLevel	float	driver accelerator pedal position	normal
iBrakePedalLevel	float	brake line pressure	normal
iActualEmTorque	float	traction motor inverter torque feedback	N m
iFuelPressure	float	CNG fuel tank pressure/level	psi
iEmCurrent	float	traction motor current draw	amps
iGenCurrent	float	generator motor current draw	amps
iBattPackVoltage	float	high voltage bus potential from EM inverter	volts
iEmRotorTemp	float	traction motor rotor temperature	Celsius
iEmInvTemp	float	traction motor inverter temperature	Celsius
iTpsFeedback	float	engine throttle position feedback	normal
oGenSpeedReq	float	generator motor speed request	rpm
oGenRegenLimit	float	generator motor regeneration request	normal
oEmAccelReq	float	traction motor torque request	normal
oEmBrakeReq	float	traction motor regeneration request	normal
oIceThrottlePos	float	engine throttle position request	normal
iPark	integer	park shifter position indicator	-
iReverse	integer	reverse shifter position indicator	-
iNeutral	integer	neutral shifter position indicator	-
iDriveSport	integer	drive "sport mode" shifter position indicator	-
iDriveEcon	integer	drive shifter position indicator	-
iZEV	integer	drive "ZEV mode" shifter position indicator	-
iHvac	integer	A/C on/off switch indicator	-
iEmTempWarn	integer	traction motor high temperature warning indicator	-
iEmControllerReady	integer	traction motor ready indicator	-
iEmFaultIndicator	integer	traction motor general fault indicator	-
iEmOvertempIndicator	integer	traction motor over temperature indicator	-
iGenTempWarn	integer	generator motor high temperature warning indicator	-
iGenControllerReady	integer	generator motor ready indicator	-
iGenFaultIndicator	integer	generator general fault indicator	-
iGenDirectionIndicator	integer	generator direction indicator	-
iIceFaultIndicator	integer	engine "check engine light" indicator	-
oIceStarter	integer	engine starter motor switch	-
oTecEnable	integer	engine controller enable switch	-
oEmEnable	integer	traction motor system enable switch	-
oEmDirection	integer	traction motor direction switch	-
oGenEnable	integer	generator motor system enable switch	-
oThrottlePwrCycle	integer	remote throttle body power switch	-

fclient_open_shm()

The function `fclient_open_shm()` takes a single argument and returns a pointer to the shared memory segment upon success. The single argument, `mode`, is used to indicate what sort of permissions are requested when dealing with the shared memory segment. `Mode` can be one of two symbolic constants: `O_RDONLY` or `O_RDWR`. A value of `O_RDONLY` indicates that the client only wants read access to the shared memory. A client that opens the shared memory segment with the flag `O_RDONLY` cannot modify the contents, either on purpose or inadvertently. This is ideal for monitoring or logging applications that only need to view the contents of the shared memory but make no changes. If `mode` is `O_RDWR`, the application is requesting full read and write access to the shared memory segment. If the function fails, it returns a value of `NULL`.

die(), warn(), notice()

The macros `die()`, `warn()`, and `notice()` are used to either print or log a message. These macros take a single argument, a pointer to a character string. The macros construct a message line of the format given in Table 4.2. This format is helpful in diagnosing problems because it gives the time the function was called, the file name and line number where the function was called, and the unique process identifier of the task that used it. The `die()` macro will also cause the calling process to quit. The message is written to the `STDERR` file stream which defaults to displaying on a terminal but can be redirected to a log file on disk or a network socket. It is the responsibility of the calling application to set up any redirections before calling any of these macros.

4.3 Task Primitives Library: libftask

To simplify dealing with multiple tasks the `libftask` library provides a few task related functions and some time functions.

Table 4.2 die(), warn(), and notice() output format

time stamp	process ID	file name	line number	user message
------------	------------	-----------	-------------	--------------

4.3.1 Time functions

Two functions exist in the library to simplify dealing with time issues. Specifically, two functions are available, `ftask_delay()` and `ftask_gettime()`.

`ftask_delay()`

This function is a wrapper for the POSIX 1003.1b `nanosleep()` function. A single parameter is required, a double precision floating point value that represents the number of seconds the calling process wishes to be suspended. The function returns a single integer that can be either a zero or -1. If the return value is a zero then at least the time specified has elapsed. The actual time in suspension might be greater than the time requested because of the granularity of the system timer. The default system timer updates at a rate of 100 Hz and the kernel cannot make scheduling adjustments faster than that. This means that the delay value may be rounded up by up to the inverse of the system timer frequency. By default this is $\frac{1}{100}^{th}$ of a second or 10 milliseconds. If the return value is -1, the function has either been interrupted by a signal or all system timers are in use. The value of the global variable `errno` determines which exception has occurred.

`ftask_gettime()`

The `ftask_gettime()` function takes no arguments and returns a double precision floating point number that represents the number of seconds since January 1, 1970. It is a wrapper for the POSIX 1003.1b `clock_gettime()` function. Like the `ftask_delay()` function the precision of the return value cannot be greater than that of the system timer tick, by default 10 milliseconds. A double precision float return value is required if any meaning is to be

had from calling `ftask_gettime()` twice within the range of the timer tick.

4.3.2 Task management functions

To simplify task management, `libftask` provides a multitude of functions. The semantics of the `ftask` functions are intentionally based on the POSIX P1003.1c *Pthreads*[13] specification. Since QNX4 does not support POSIX light-weight processes, `pthread`s, the combination of the shared memory routines in `libfclient` and the task management functions in `libftask` can provide similar functionality.

Most `ftask` functions' first argument is a pointer to a data type *ftask*. This parameter is a sort of "handle" to a unique process created by `libftask` functions. While the internal structure of an *ftask* is visible, it is not meant to be manipulated directly. One of the more curious parameters to some `libftask` functions end in `_func`. These function parameters are actually names of other functions.

`Libftask` has two features that are unique. Tasks may be "triggered". Triggering is a simple form of IPC based on the QNX native function `Trigger`. A task may voluntarily suspend, or block, itself by calling the `ftask_trigger_block()` function. Another process that has access to the *ftask* handle that identifies the trigger blocked task may use the `ftask_trigger()` function to unblock the voluntarily blocked task. No data is transferred with a trigger, only the triggered task's runnable state is modified. The `ftask_trigger()` function does not cause the calling process to block. This makes triggering an asynchronous form of IPC.

The other unique feature of `libftask` is that a created task may be attached to a periodic timer. This is actually a wrapper for the rather complicated method of attaching a process to the main timer interrupt. A task created with the ability to attach to the timer interrupt calls `ftask_periodic_timer_block()` to voluntarily suspend itself. On the next timer interrupt the operating system will allow the `ftask_periodic_timer_block()` function to return, allowing the task to run. A loop that uses the `ftask_delay()` function will execute

with a period that includes the time for the loop to run in addition to the time specified. A loop that uses `ftask_peridic_timer_block()` will run at a rate independent of the time required to execute the contents of the loop as long as executing the contents of the loop do not exceed the specified timer frequency.

ftask_init()

The parameter list to `ftask_init()` is quite extensive so it will be listed in its entirety here.

```
int ftask_init(ftask *ft,
              int policy,
              int priority,
              void (*start_func)(void *)
              void *start_func_arg,
              void (*cleanup_func)(int),
              int allow_trigger,
              int allow_periodic_timer,
              int periodic_timer_hz),
```

The purpose of the `ftask_init()` function is to initialize the *ftask* passed as the first parameter.

ftask *ft This parameter is a pointer to an *ftask* data type. It is used as the “handle” to uniquely identify the task that will eventually be created.

int policy The policy parameter is one of the symbolic constants, `SCHED_FIFO`, `SCHED_RR`, or `SCHED_OTHER` that represent FIFO, round-robin, or adaptive scheduling.

int priority The priority parameter is an integer value that represents a valid scheduling priority number. It should be noted that under QNX the highest priority for a non *super-user* task is 19. A value higher than this requires additional privilege.

void (*start_func)(void *) The `start_func` parameter is the name of a function that will be the entry point for the newly created task. This is analogous to the `main()` function entry point of all C language programs.

void *start_func_arg This parameter will be passed as the only argument to the created task’s entry function. The void pointer type can be typecast to a variable of any type.

If the entry function requires no argument this parameter may be set to the symbolic constant `NULL`.

void (*cleanup_func)(int) the `cleanup_func` argument is, like the `start_func` argument, a name of a function to call when the task is stopped. If no cleanup function is needed for a particular task, this parameter may be specified as `NULL`.

int allow_trigger This is really a boolean value that specifies whether the `ftask` triggering functionality is required for this task. A value of zero means no triggering ability is requested and any non-zero value means that triggering is required for this task.

int allow_periodic_timer Another boolean value, this parameter specifies if the ability to attach to a periodic timer is required. It should be noted that in QNX this functionality requires *super-user* privileges.

int periodic_timer_hz The `periodic_timer_hz` argument is only valid for a task that has a non-zero `allow_periodic_timer` value. The value is a specification of the frequency, in Hz, that a task can be scheduled to run. If periodic behavior is not needed it is safe to set this value to zero.

The return value of `ftask_init()` will be zero to indicate success while a value of -1 will be returned to indicate that there is some sort of inconsistency in specified arguments.

ftask_create()

The `ftask_create()` function takes a single argument, a pointer to an *ftask* type that has been previously initialized with the `ftask_init()` function. If `ftask_init()` had not previously been used to initialize the *ftask* the results are undefined. This is where a new task is actually created, internally it calls the `fork()` function to create the new task.

ftask_delete()

This function is used to stop a previously created task. It causes the named task to jump to the function named as the `cleanup_func` argument to `ftask_init` and exit. A return value of zero means that the task was successfully requested to quit. Even if this function does return zero, that does not mean that the requested task has quit. It only means that the task has received the instruction to quit. A non-zero return value means that the command failed.

ftask_destroy()

This function is similar to `ftask_delete` function except that the specified *ftask* is forcibly stopped. No internal cleanup will be done for a task that is destroyed. A return value of zero means that the task was successfully destroyed while a non-zero return value indicates that the function failed.

ftask_trigger()

Calling this function will allow a task that is voluntarily in a blocked state to become unblocked and continue.

ftask_trigger_block()

The `ftask_trigger_block()` function causes the calling process to be put in a suspended, or blocked, state. The calling process may become unblocked if either another process uses `ftask_trigger()` to unblock it or a signal is delivered.

ftask_wait_on_tasks()

This function puts the calling process in a suspended state in which it will be restored if one of the processes it created with `ftask_create()` exits. The single parameter which is a

pointer to type *ftask* will contain information for the task that ended. This function may be interrupted by the delivery of a signal.

ftask_same()

The `ftask_same()` function is used to compare known *ftasks* to the *ftask* returned by `ftask_wait_on_tasks()`. The return value is 1 if the two tasks refer to the same process and zero if they are different.

In addition to the task management functions described previously, `libftask` also has a few convenience functions that are used internally but can be useful for other purposes.

ftask_sched_adjust_self()

This function takes two arguments, an integer specifying scheduling policy and an integer specifying scheduling priority. It is used internally by `ftask_create` to adjust the scheduling parameters of newly created tasks. This function is a wrapper around the POSIX 1003.1b `sched_setscheduler()` function.

ftask_register_cleanup_self()

The `ftask_register_cleanup_self()` function is used internally by `ftask_create()` to register a signal handler for `SIGTERM`. The only parameter is the name of the function to call on reception of the `SIGTERM` signal.

ftask_register_reread_self()

This function is identical to `ftask_register_cleanup_self()` except that a handler for the `SIGHUP` signal is specified. `SIGHUP` is traditionally used to indicate the controlling terminal connection has “hung up”. Daemon processes, since they do not have controlling

terminals, traditionally redefine this signal to indicate that a rereading of configuration files has been requested.

4.4 Hardware Interface: fcard

The program that handles all hardware I/O is called *fcard*, the *fcar* daemon. A daemon, in traditional UNIX terminology, is a process that is designed to run for long periods of time and perform a very specific service. Daemons break away from their parent processes and usually redirect any diagnostic messages to a log file. The main reason for using a daemon to do all hardware I/O is one of mostly one of simplicity. *Fcard*'s main purpose is to read shared memory, write the specified values from shared memory to the hardware, read in values from the hardware, and put the read in data into shared memory. *Fcard* acts as the manager of the common shared memory segment, see Figure 4.1. *Fcard* is not a device driver in the traditional sense.

Device drivers under QNX can take on many forms. There are basically three ways to access hardware under QNX. The first involves interfacing with the QNX Dev manager. This is the most complicated way because Dev drivers have to keep track of many data structures just to communicate with Dev in addition to controlling dedicated hardware. The second method is what QNX calls an *iomanager*. This method is simplified because registering with the Dev manager is not needed. This leaves the driver to concentrate on controlling its own hardware. Both of the previous device driver methods are termed 'POSIX' drivers in QNX documentation because they involve making a special 'node file' in the */dev* directory. This */dev* node is supposed to act similarly to a regular disk file. One of the strengths of traditional UNIX systems is that almost everything can be considered a 'file'. That is, POSIX 1003.1 functions like *open()*, *read()*, *write()*, and *close()* operate the same on a file on disk, an interprocess communication pipe, a serial port, and even a network connected socket. While this sort of abstraction is very useful for complicated hardware, it

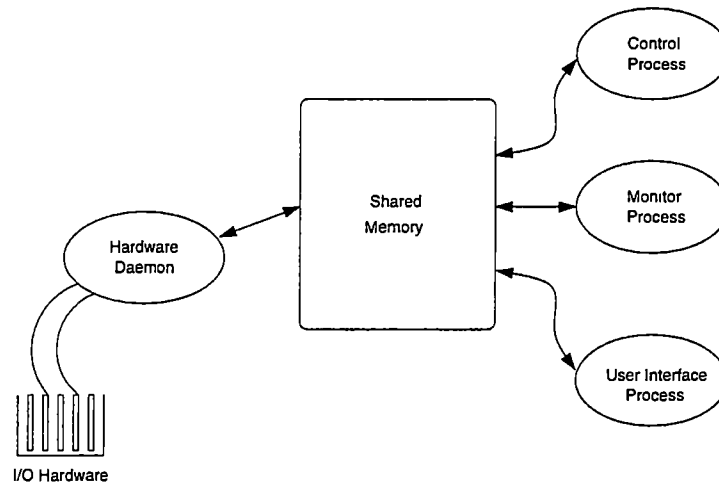


Figure 4 1 Shared memory connection

is overkill for the relatively simple I/O hardware used in the UTK FutureCar. The third type of device driver for QNX is no device driver. Because all device drivers under QNX are designed to be run outside of the microkernel, all of the facilities for accessing hardware directly are readily available to regular user processes. All that is required to do privileged hardware access is to link the program with privilege level 1 and run as the *super-user*.

4.4.1 Features

The fcard program has many features that make it very capable

- It has the ability to read calibration and configuration values from a text file on disk. Because configuration file reading is handled by a dedicated task, changes can be made while the program is running without interrupting regular operation.
- A command line option can be specified to disable hardware I/O but otherwise operate normally. This feature allows for easy debugging of the non-hardware I/O parts of the program on a machine that does not have the Octagon hardware installed.

- It features a sort of ‘task manager’ that can restart hardware I/O tasks if one fails due to a math error or other exception. The task manager also can tell if a task is being restarted too often and halt the system.
- A command line option can be specified to restrict accesses to the QNX filesystem. This option is most useful if `fcard` is used in an embedded environment where either the filesystem is not available when it starts, or there is no filesystem.

4.4.2 Operation

During normal operation, `fcard` operates as four separate, cooperating tasks as shown in Figure 4.2. It uses the `fcntl` library, discussed in Section 4.2, for IPC and the `ftask` library, discussed in Section 4.3, extensively for task management.

The reasoning behind using four separate tasks is based on time. First, the task handling the Octagon hardware, shown in Figure 4.3, has to interface reliably and quickly with the hardware. This hardware task attaches to the system timer and loops at a rate of 100 Hz. Since tests show that the time to do one loop is only about 1 millisecond, the hardware task is idle most of the time.

The task that handles reading the kWh meter, shown in Figure 4.4, only operates at 1 Hz because that is the speed at which the meter reports data to the serial line. The configuration file reader, shown in Figure 4.5, is idle most of the time. It only has to respond to requests to reread the file. While it might be possible to join the kWh meter reading and configuration file readers to reduce resource requirements, having them as separate tasks makes them simpler to manage and understand.

The manager task, shown in Figure 4.6, ideally does nothing and is idle all the time after starting all the other tasks. It only handles exceptions such as premature process death, configuration file relaying, and the orderly shutdown of the `fcard` tasks. In a resource constrained environment the task manager could also join with the kWh meter task and the configuration file reader at the detriment to simplicity.

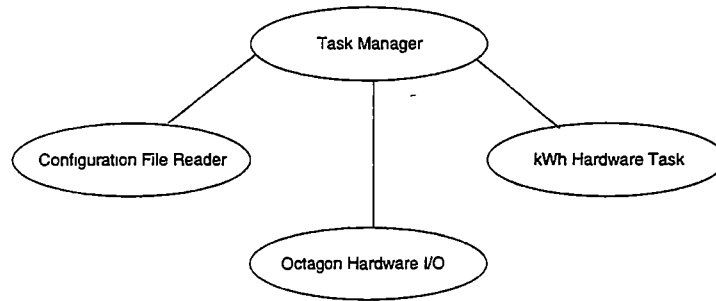


Figure 4 2 Fcard Process Tree

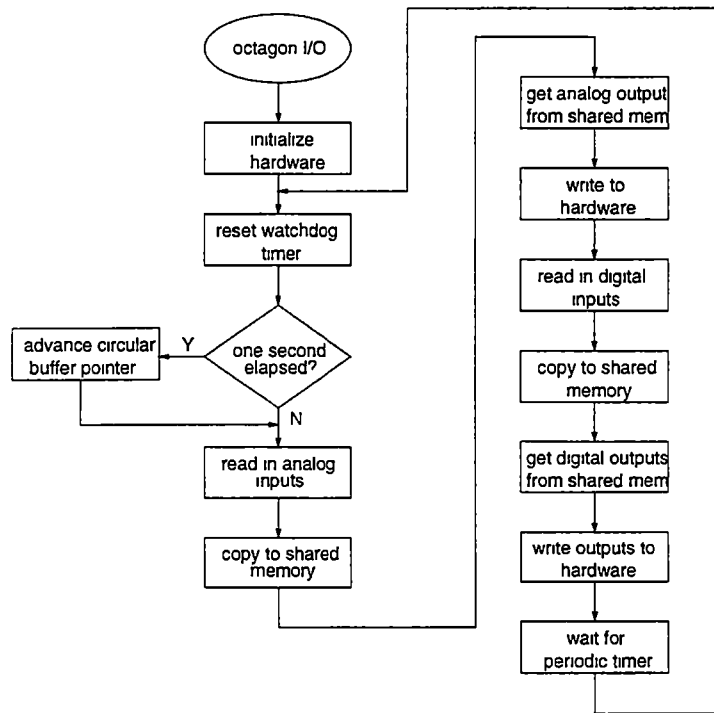


Figure 4 3 Fcard Octagon Hardware I/O Task Flowchart

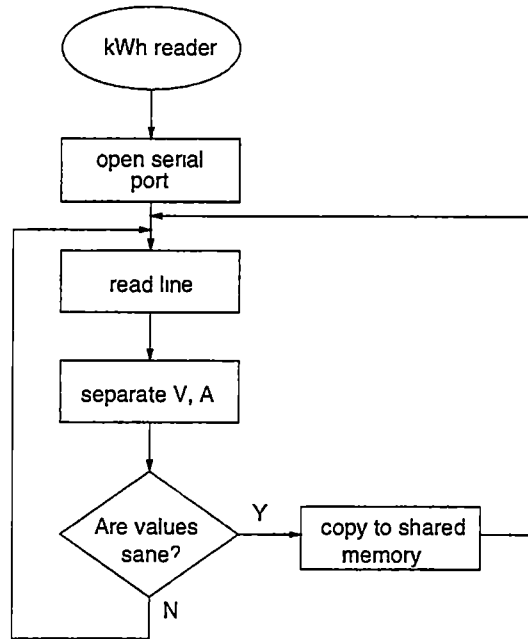


Figure 4 4 Fcard kWh Meter Reader Task Flowchart

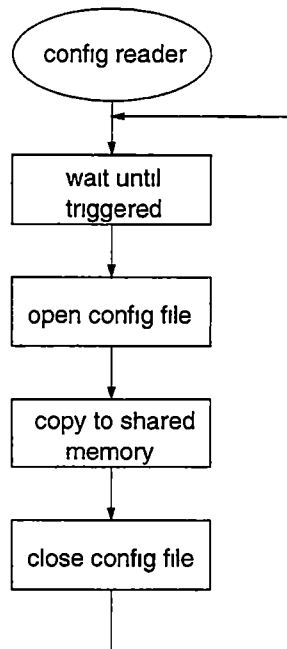


Figure 4 5 Fcard Config File Reader Task Flowchart

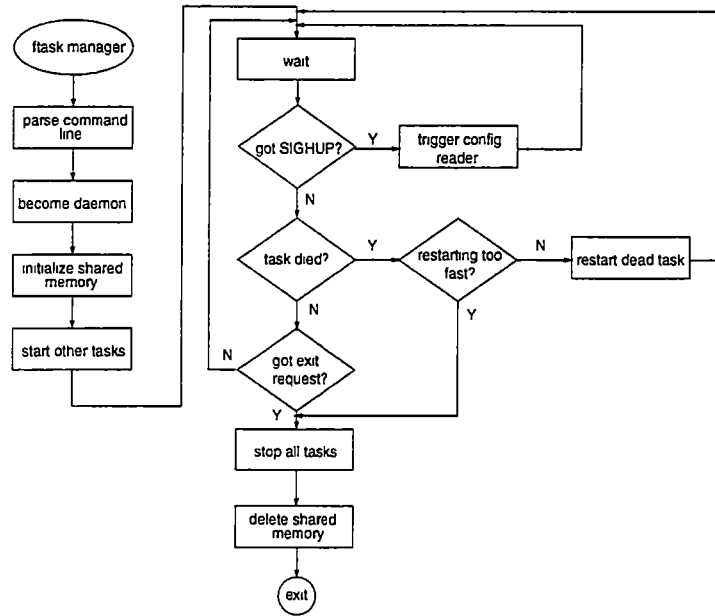


Figure 4 6 Fcard Task Manager Flowchart

Fcard's most difficult task is managing I/O to the Octagon cards. Since there is no real QNX device driver for the cards a custom solution was written. Access to the cards is accomplished with IN and OUT assembly instructions. The 5710 Multifunction I/O card has the ability to assert an interrupt line on the bus, but this feature is not used. The 5710 card's lengthiest operation is performing an analog-to-digital (A/D) conversion, but this only takes a few microseconds. Instead of introducing the complexity of using an interrupt to indicate that the A/D conversion is finished, the card is simply polled, that is, it is read in a hard loop to check for completion. This might seem to be an unnecessary waste of CPU time, but, in practice, it has little effect on the overall operation of the system. Even polling every A/D channel the two 5710 cards in succession uses very little CPU time.

4.5 Control Program: fcar

Fcar is the name of the main control program. The listing for fcar is given in Appendix D.2. One of the main features of this overall control system is the fact that hardware I/O is done independently from any control decisions. This has the benefit of making the working control code syntactically simpler, since, instead of breaking away from the control algorithm to fetch data from the hardware and then returning, the control algorithm just assumes that the variables in shared memory are automatically updated in a timely manner.

4.5.1 Overview

The fcar control program, like fcard, is a multithreaded application. Upon execution it creates four additional tasks, shown in Figure 4.7, to handle different control requirements. Fcar is similar in structure to the fcard program. It also uses the fclient IPC library and the ftask library for process primitives. Multiple tasks are used because of temporal issues. Each fcar task makes control decisions with differing time requirements. The ICE controller task, responsible for starting and stopping the engine, uses simple delays to sample engine speed to determine whether ignition has occurred at a rate on the order of 4 Hz. The ICE controller task must be able to operate independently from the main control loop. The main control loop has to remain responsive to driver requests even while in the transition time required to physically start or stop the engine. The throttle controller task is responsible for ensuring a smooth engine engagement by ramping the engine throttle in a controlled manner. It operates using simple delays at a rate on the order of 10 Hz. Since the service provided by the throttle controller task only happens after the ICE controller has successfully started the engine, it is possible that these two tasks could be merged. The SOC calculator task operates at approximately 1 Hz, and the main control task, the mode selector task, runs at 100 Hz by hooking into the system timer interrupt.

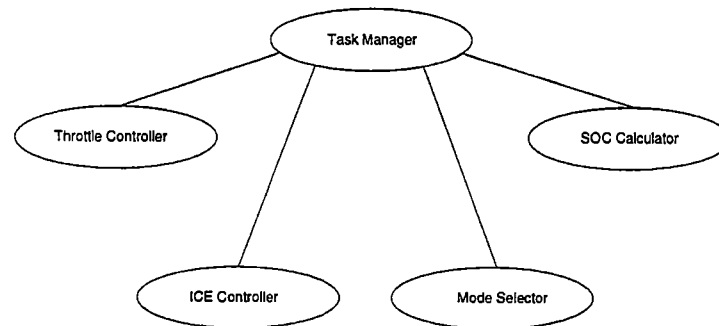


Figure 4.7 Fcar Process Tree

4.5.2 Task details

SOC calculator

The SOC calculator task whose flow chart is shown in Figure 4.8 implements the SOC estimation method discussed in Section 2.4. As the flow chart shows the procedure is simple. All that the task has to do is go into a loop with a delay that makes it recalculate approximately every second. Looping any faster would probably not result in a better estimation because the underlying kWh meter hardware only reports the information used by the SOC calculator at approximately 1 Hz. The listing for the SOC calculator task is in Appendix D.2.9.

Throttle controller

The throttle controller task has the responsibility of ensuring a smooth engagement of the engine. The flowchart for the task, shown in Figure 4.9, shows that this task is a very simplistic one. The remote throttle body unit is controlled by writing a normalized floating point value to shared memory. While there is a throttle position feedback value available, the task does not use it. The Mikuni unit proved to be very responsive to position requests in preliminary tests, so in the interest of simplicity, no feedback control algorithm is used.

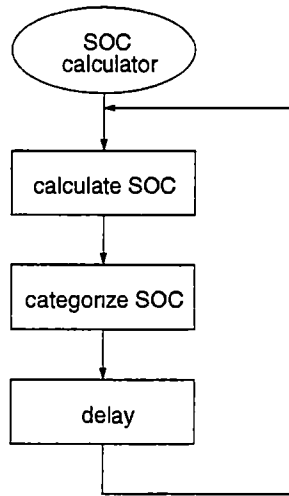


Figure 4 8 Fcar SOC Calculator Task Flowchart

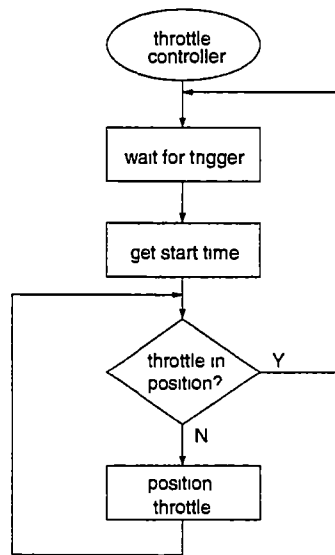


Figure 4 9 Fcar Throttle Controller Task Flowchart

ICE controller

The ICE controller task, shown in Figure 4.10, is responsible for starting and stopping the engine. The mode selector task actually makes the decision when to start and stop the engine. To start the engine, the TEC-2 ECU is turned on which then also enables the CNG fuel tank solenoids, the remote throttle body is powered and the throttle position is set to idle, and finally the starter motor is engaged. To determine if the engine has started, the engine speed is monitored. If the engine speed is greater than the speed induced by the starter, the engine is considered successfully started. The CNG fuel lines on the UTK FutureCar are controlled with a master quarter-turn valve. If this valve is inadvertently left closed and the engine is unable to start, the ICE controller task will give up after a few tries and keep a record that the engine is unable to be started. Since it might be possible for a driver to fix a minor fuel problem, the record of a “dead engine” is cleared by the mode selector task if the shifter is moved to the neutral position.

To stop a running engine the remote throttle body is put in the idle position and turned off and ECU power is removed. When the TEC-2 ECU power is removed, ignition and fuel delivery are stopped. While this method of stopping an engine is fairly traditional, unburned fuel emissions could be reduced by only removing the fuel supply and letting ignition continue until the engine starves. Preliminary tests show that for the engine used, fuel in the lines runs out after about 15 seconds. This was considered to be too long so the method of removing the ignition was chosen.

Mode selector

The fcar mode selector task is the core of the control system. It is the main implementation of the dual hybrid planetary drive train algorithm. As such, it is fairly complicated. The flowcharts for this task have been broken up based on mode. The mode selector task operates by looping at a rate determined by the main system timer interrupt. The entry point of the task begins by reading the driver’s mode request with the shifter position. After reading the

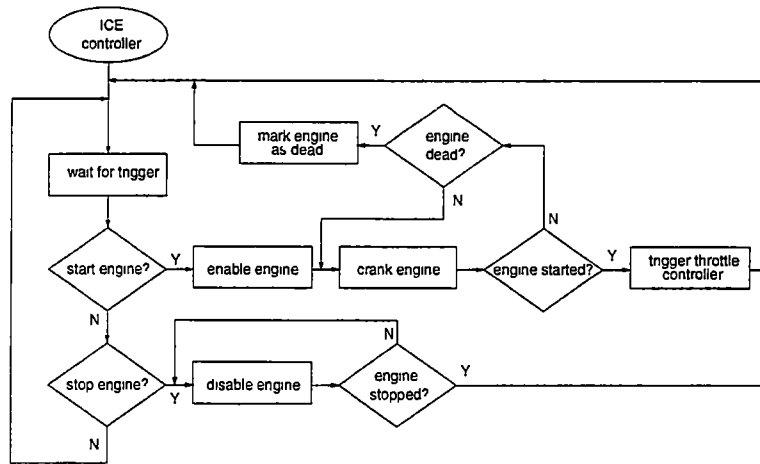


Figure 4 10 Fcar ICE Controller Task Flowchart

shifter, the variable mode in shared memory, or more specifically `s->cv[s->active].mode`, is assigned. The current mode of operation is stored because functions for proportioning drive train components are used for multiple modes. This can be seen when the mode selected is ZEV, zero emissions vehicle, electric power only. The entry point of the mode selector task is shown in Figure 4 11.

In ZEV mode, Figure 4.12, the first thing done is the enabling of the two motor controllers. If the controllers do not report back that they are ready, nothing else happens. In order to protect the drive train from any shocks resulting from “shifting” from forward to reverse, vehicle speed is checked to make sure that a direction change doesn’t happen at speed. If the driver chooses a forward “gear” when traveling in reverse at speed, the control system will override this choice and maintain the current direction until the vehicle speed has decreased. After this safety check is done, the main traction motor is confirmed to be operating in the forward direction. As previously mentioned, the mode variable is checked to see if the function is being called while really in ZEV mode. To comply with zero emissions, a request is sent to the ICE controller task to shut down the engine. Finally the control requests for the traction and generator motors based on the driver requests and

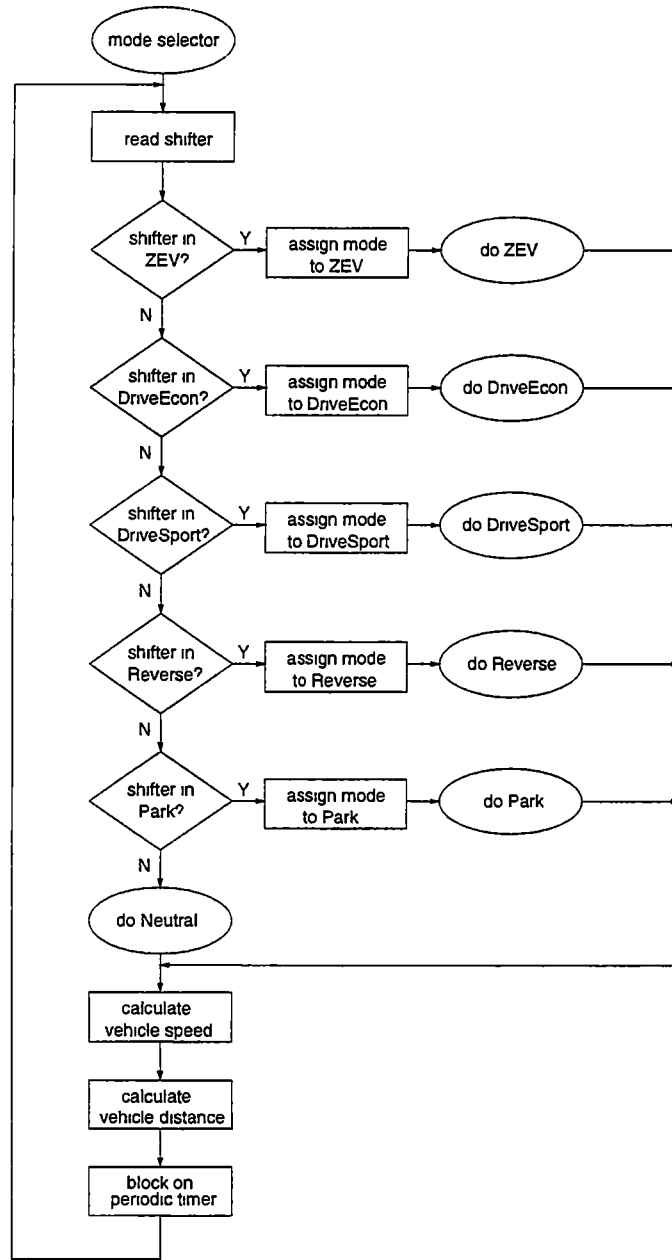


Figure 4 11 Fcar Mode Selector Task Entry Point Flowchart

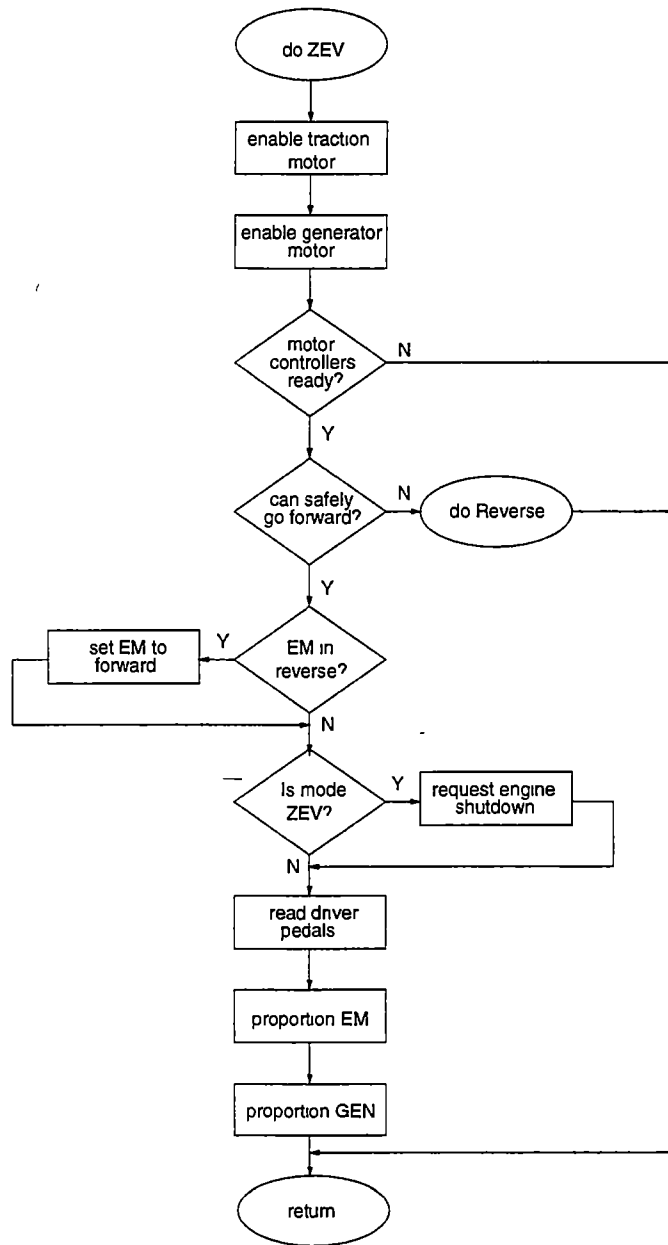


Figure 4 12 Fcar Mode Selector Task, ZEV Function Flowchart

current vehicle conditions are calculated and written to shared memory

Reverse mode, Figure 4 13, is very similar to ZEV mode save for the direction setting on the main traction motor The main difference between ZEV and Reverse modes is that in reverse, the generator is disabled This implies that the engine is not functioning also Reverse for the UTK FutureCar can only be done with electric power because there is no normal reversing gear in the transmission that allows the engine to spin in the normal direction while going backwards.

Park mode in the first iteration of the control code would request full regeneration from the traction motor in an attempt to prevent motion The physical transmission does not have a mechanical gear lock for parking and the traction motor used in the FutureCar does not have very good low speed regeneration characteristics This results in having to use the emergency parking brake to maintain the vehicle on a grade Since park mode is ineffective, neutral mode, Figure 4 14, is used for both park and neutral shifter settings. The main function of neutral mode is to disable all drive train components While in neutral, the system also resets the “dead engine” condition mentioned in the previous ICE controller section

The two “Drive” positions available on the shifter, DriveEcon and DriveSport, were originally designated as such for two different hybrid operation modes The DriveEcon position is used for normal hybrid electric vehicle, HEV, operation while the DriveSport position is only used for testing purposes HEV, or DriveEcon, mode, shown in Figure 4.15, is the normal vehicle operating mode This mode is similar to the previously mentioned modes in that it sets the direction of the traction motor It also makes requests from the ICE based on SOC If SOC is high, the engine is not required and the ZEV function is used to proportion the two drive train motors If SOC is low, the engine is used to provide additional drive torque to the wheels and recharge the battery pack at the same time Code exists that attempts to do hybrid regime transitions, discussed in Section 3 3 1, but it is untested

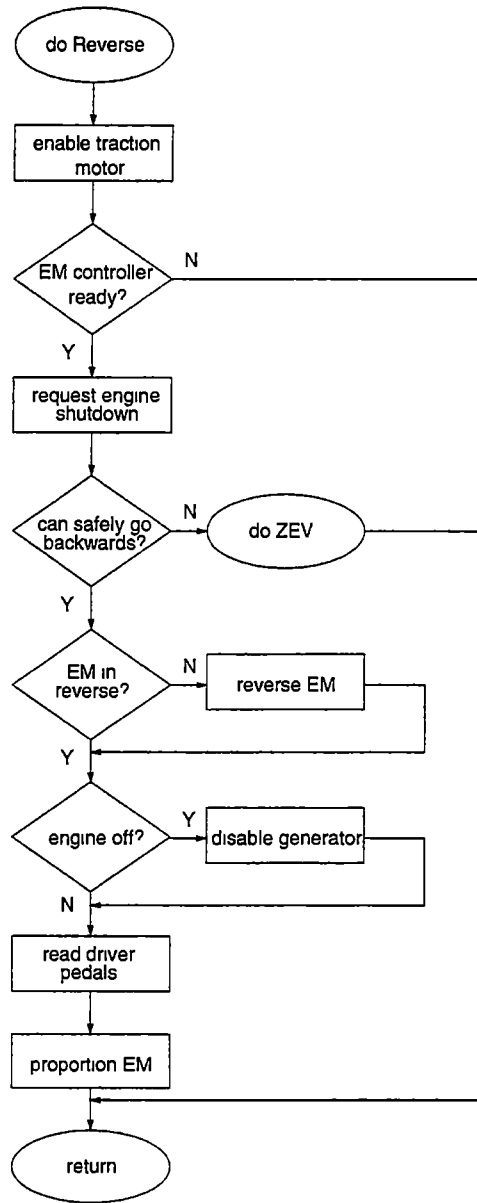


Figure 4 13 Fcar Mode Selector Task, Reverse Function Flowchart

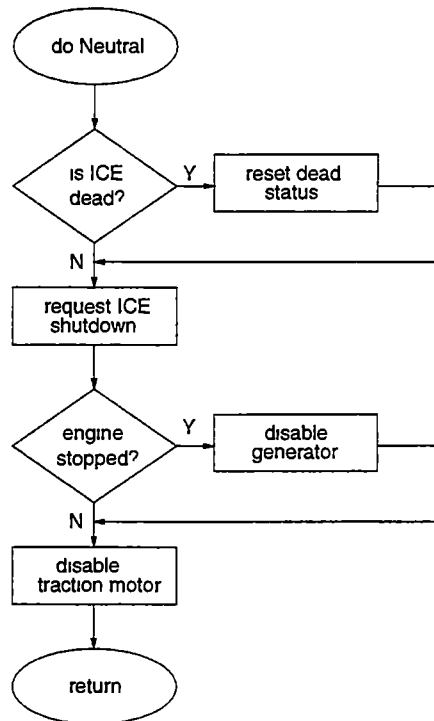


Figure 4 14 Fcar Mode Selector Task, Neutral Function Flowchart

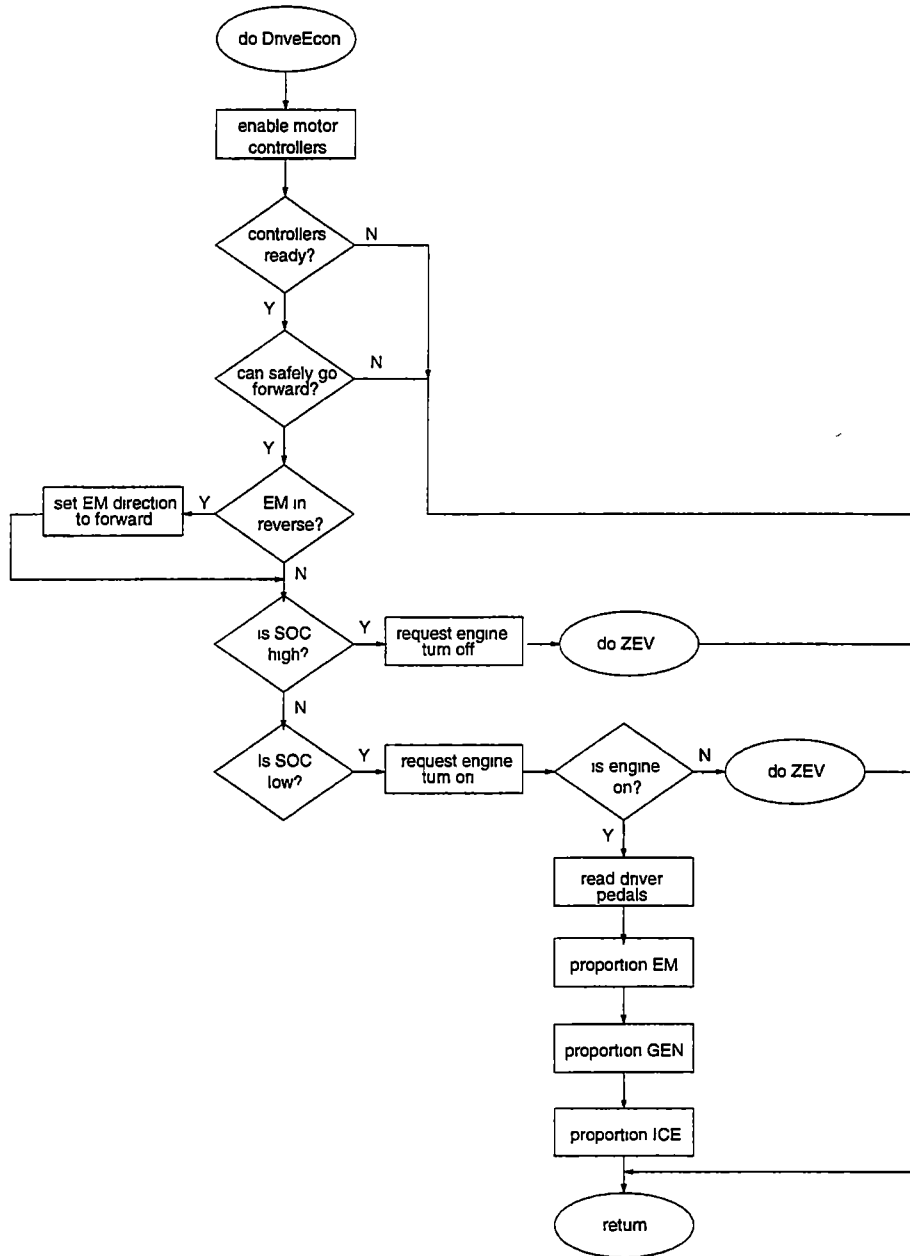


Figure 4 15 Fcar Mode Selector Task, DriveEcon Function Flowchart

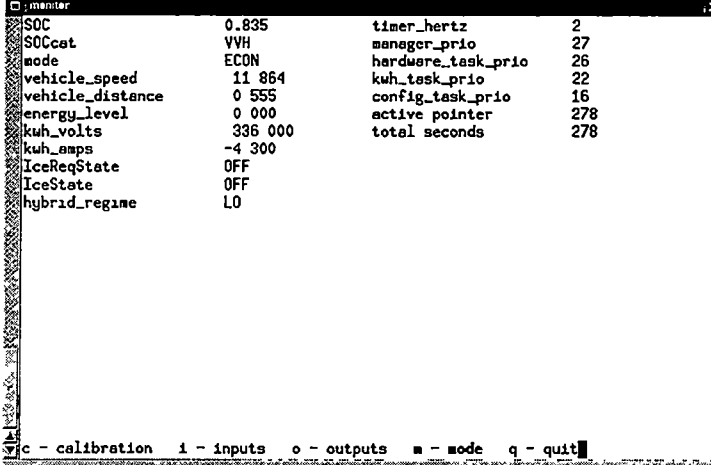
4.6 Monitoring and Diagnostics

Debugging a real time control system is a difficult task. Since many operations are time dependent, using a traditional source code debugger is difficult at best. What is needed is a way to examine the running system in normal operation. The control programs do produce some diagnostic messages, but these are usually just simple notices or fatal errors. The overhead of printing, either to a connected terminal or a file, is prohibitive while maintaining the high frequency loop rates needed by the critical control tasks. The choice of using a large shared memory segment to hold current and past data makes it simpler to diagnose problems. To this end a few custom diagnostic tools were written to examine the running system.

4.6.1 Shared Memory Monitor: *mon*

A simple *curses* based tool, *mon*, displays the entire contents of the active portion of the shared memory segment used by all controller tasks. The interface is text based and can run on any terminal with an addressable cursor. There are four screens of information available. The first is a listing of all the calibration values used by *fcard* to convert voltages to engineering units. The next screen shows all of the digital and analog inputs read from the Octagon hardware. After that is a screen that shows all of the output values. The final screen, shown in Figure 4.16, displays all other values including many state variables used by the *fcar* control program and the battery pack information read from the kWh meter over a serial connection. Values are updated on screen every half second.

Switching between screens is done either with the arrow keys or with the 'c', 'i', 'o', or 'm' keys that represent the calibration, inputs, outputs, and mode screens respectively. Exiting the application is done by pressing the 'q' key. The control computer on the car has *mon* installed and it can be run by using the username "mon" at the QNX login prompt.



```

mon
SOC          0.835          timer_hertz      2
SOCcost      VVH             manager_prio     27
mode         ECON           hardware_task_prio 26
vehicle_speed 11.864         kwh_task_prio   22
vehicle_distance 0.555         config_task_prio 16
energy_level 0.000         active_pointer   278
kwh_volts    336.000       total_seconds    278
kwh_amps     -4.300
IceReqState  OFF
IceState     OFF
hybrid_regime LO

c - calibration  i - inputs  o - outputs  m - mode  q - quit

```

Figure 4.16 Mon shared memory viewer tool

4.6.2 Data Trends Viewer: trends

While the *mon* tool displays only the active values in shared memory, another tool, *trends*, is used to view a graph of the time history of selected variables in shared memory. This program is a CGI, Common Gateway Interface, program that is meant to be run by a web server. The freely available Apache web server was compiled for QNX and runs at boot up on the Octagon control computer. This allows any graphics capable web browser to be used to view time histories. The program is not much more than a wrapper application that uses the *fcntl* library to access the shared memory segment and then use the freely available Gnuplot program to make the actual images that are sent to a connected web browser. An example session is shown in Figure 4.17.

4.6.3 Data Logging: flogger

One disadvantage to using *trends* is that the shared memory contents are not saved when the control computer is turned off. To remedy this problem, the *flogger*, or *fcar* logger, application was written. This data logger samples the contents of shared memory and periodically writes the data to the onboard M-Systems DiskOnChip 2000 non-volatile flash

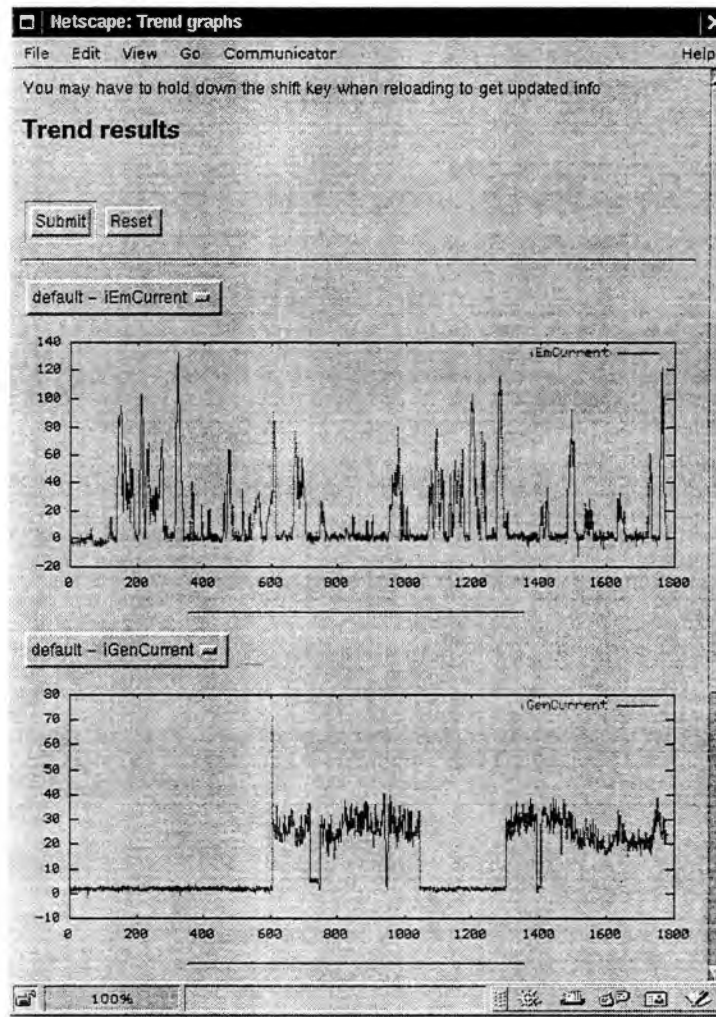


Figure 4.17: Trends shared memory history viewer tool

memory device. Data is cached in memory, 10 seconds worth by default, and then flushed to "disk." Since disk write operations are done only when the cache is full, data corruption is minimized that could occur due to the removal of power during a write. The application saves a log file in text format with commas separating the individual fields. At a sample rate of 1 Hz, the text file generated is approximately one megabyte for every hour in operation.

Chapter 5

Conclusions

The UTK FutureCar control system is functional. The vehicle can operate in both ZEV mode and a charge sustaining HEV mode. Unofficial preliminary tests indicate that HEV mode results in a fuel efficiency of approximately 30 miles per gallon of gasoline equivalence with an untuned system. There is room for improvement.

5.1 Improvements

The UTK FutureCar vehicle as a whole is largely untested. The most extensive use of the vehicle happened during the test drives used to collect data for state of charge calculations. Since most of the vehicle control code was developed specifically for the FutureCar competition in a hurried manner, many aspects of the original design were not implemented.

In HEV mode, switching regimes has not been tested. While there is some code available that would do the necessary transition, it has not been tested. The main reason behind this is the good chance of breaking some transmission component. During initial HEV testing the planetary gear set did get spun fast enough to seize one of the planet gears. This was mainly due to an overly aggressive engine throttle setting that, even though the manufacturer's specifications indicate otherwise, allowed the engine to overpower the

generator. Other contributing factors to this transmission failure were lack of sufficient lubrication and operator error. Since HEV regime transitions take place at a relatively high vehicle speed, 45-55 mph, any testing of regime switching would best be done on a chassis dynamometer where, if any problems arise, damage could be minimized. Since the transmission components are largely custom made, replacement parts might have a significant lead time.

“Sport Mode”, an HEV mode that, in addition to low SOC conditions, uses the engine/generator to increase available torque in response to aggressive driver requests has not been implemented. In this mode the engine will be turned on even at a high SOC. This is unimplemented mainly because of time constraints.

The drivability of the vehicle does not follow the stock vehicle. While compensations are made to make ZEV and HEV modes feel the same to the driver, there is a significant “boost” during HEV operation. The main factor to equalizing the driver feel comes from the generator current feedback signal. After some SOC testing the generator current signal provided by the generator inverter was suspect. The generator current signal does not follow the pack current measured from the kWh meter when it should.

The state of charge algorithm, discussed in Section 2.4, has performed well in simulations, but has not actually been integrated as a controlling factor into the rest of the control system. The current system, as of this writing, only displays the estimated SOC of the battery pack, but leaves the responsibility of starting and stopping the engine to the driver. For a fully automatic system that starts and stops the engine based on SOC, the decision needs to be made by the computer.

The rather small battery pack capacity, 13 A h, seems to be insufficient for long periods of electric-only operation. While this capacity was deemed to be adequate in the original design, the high current environment to which the pack is subjected severely decreases the amount of available energy. It is possible, probably without much modification, to replace the Hawker Genesis 13 A h cells with 16 A h cells because they are almost physically

identical. While the added capacity might not add much to the overall energy storage capacity, it would be an improvement.

The driver interface is sub-optimal. While the vacuum fluorescent display module does provide useful information in a unique manner, the original design specified using the stock analog speedometer in addition to the the VFD for driver feedback. There is code available that would operate the large analog gauge, but it is also untested. The main reason for this is the high current, 25 mA, required to drive the gauge coils. The Octagon analog output cards are only specified to source 5 mA of current and initial attempts at amplifying the signal failed. Since the choice was made not to risk controller hardware damage with a direct connection, this feature was left undone. The original interface specification called for a small LCD screen mounted in the dashboard. Since QNX does feature a very small and capable graphics system, Photon, a graphical driver interface could be added to the current system. Since a source for LCD panels small enough to fit in an automobile dashboard and a good place to put a video screen in the dash were never found, this idea stagnated.

The system controller has a long power-up sequence. While the code written for controlling the vehicle has negligible start up time, the time from the key switch being moved to the "on" position to a responsive vehicle is about 30 seconds long. This is due mostly to the Octagon CPU card's PC compatible BIOS. While a PC compatible platform was chosen to speed development and make control code changes quick, the regular boot-up sequence takes far longer than the time to start the engine of a conventional vehicle. There are remedies for this from the QNX manufacturer in the form of eliminating the PC compatible BIOS start-up code, but this has not been rigorously investigated. Since the control code is largely portable, it could be moved to a different computing hardware and software environment, one more suited to embedded work, without much trouble. Completely abandoning the PC hardware might be the best remedy to the slow boot problem.

5.2 Future Work

Future work would primarily involve more testing and performance evaluation of the vehicle as a whole. The control system code, while not complete to the original specification, is very functional for the majority of the tasks required of a hybrid electric vehicle. What is needed is more fine tuning of the existing code to coax more efficiency out of the system.

Some tasks to be done in the future would be more sensible to try than others. Probably the easiest improvement to make would be changing the responsibility of the hybrid mode engine operation from the driver to the SOC algorithm included in the code. While the algorithm might need some small changes to simulate a person's decisions, this is easily done. The next important step would be the completion of the HEV regime switching code. Since the "high" regime code is untested, the vehicle speed has been limited to about 55 mph. To reach highway cruising speeds, the switch to the "high" regime must be made.

In the process of developing a control system for a dual hybrid electric vehicle, perhaps something greater has come of it. The control code was designed in a framework so that it could be used outside of its current implementation, independent of both the Octagon hardware used and the QNX operating system. A possible next step for this research would be to adapt the control system code to an entirely different computational environment on a different vehicle.

Bibliography

Bibliography

- [1] K L Barfield A microprocessor control system for a hybrid electric vehicle Master's thesis, University of Tennessee, Knoxville, 1994
- [2] Cruising Equipment Co , Seattle, Washington USA *Installation Manual, Kilowatt-Hour+2 Meter*, March 1995
- [3] Electromotive Inc , Manassas, Virginia USA, <http://www.electromotive-inc.com>. *Total Engine Control (TEC) engine management unit*
- [4] B O Gallmeister *POSIX 4 Programming for the Real World* O'Reilly and Associates, Inc , 1995
- [5] Hawker Energy Products Inc , Warrensburg Missouri USA *Genesis Application Manual*, third edition, December 1997
- [6] Hawker Energy Products Inc , Warrensburg Missouri USA *Genesis Selection Guide*, second edition, February 1998
- [7] F Hayes-Roth, D A. Waterman, and D B Lenat *Building Expert Systems* Addison-Wesley Publishing Company, Inc , 1983
- [8] X He Hybrid electric vehicle simulations and evaluation Master's thesis, University of Tennessee, Knoxville, 1997
- [9] B W Kernighan and D M Ritchie *The C Programming Language, Second Edition*. Prentice Hall, Inc , 1988
- [10] P. Laplante *Real-Time Systems Design and Analysis* IEEE Press, 1992.
- [11] D Lewine *POSIX Programmer's Guide* O'Reilly and Associates, Inc , 1991.
- [12] H W Muller *Epicyclic Drive Trains · Analysis, Synthesis, and Applications* Wayne State University Press, 1982
- [13] B Nichols, D Buttlar, and J P Farrell *Pthreads Programming*. O'Reilly and Associates, Inc , 1996

- [14] A Oram and S Talbott *Managing Projects with make* O'Reilly and Associates, Inc., 1991
- [15] QNX Software Systems Ltd , Kanata, Ontario Canada *QNX OS System Architecture*, 1993
- [16] J D Taylor An evaluation of the effects of increased exhaust gas recirculation on a dedicated natural gas vehicle conversion Master's thesis, University of Tennessee, Knoxville, 1997
- [17] L H Tsoukalas and R E Uhrig *Fuzzy and Neural Approaches in Engineering*. John Wiley and Sons, Inc , 1997
- [18] B E Tucker The development and implementation of a control system for a hybrid electric vehicle Master's thesis, University of Tennessee, Knoxville, 1997
- [19] Unique Mobility Inc , Golden, Colorado USA, [http //www uqm com](http://www.uqm.com). *Electric motors for alternative fuel vehicles*
- [20] K Yamaguchi, S Moroto, K Kobayashi, M Kawamoto, and Y Miyaishi Development of a new hybrid system - dual system *Society of Automotive Engineers paper 960231*, 1996

Appendices

Appendix A

Basic Control Code Modification

QNX is a self-hosted operating system. That is, the development tools, compiler, assembler, debugger, etc., run on QNX. This differs from most real time operating systems that require a cross-compiler that runs on a host computer to generate an executable that is loaded on the target computer. While QNX can be self-hosted, the FutureCar control system code is actually compiled on a small laptop computer and then copied to the Octagon computer installed in the vehicle.

The QNX native networking protocol, FLEET networking, makes it simple to share the resources of two computers. The development laptop computer, node #3, connects to the Octagon computer, node #1, over thinnet coax ethernet, also known as 10-base-2 ethernet. A thinnet ethernet topology was chosen because it is physically smaller than thicknet, 10-base-5, and doesn't require extra hub hardware that a 10-base-T network might require. To make any changes to the control code the following steps have to be made:

- Plug the laptop AC adaptor into the power strip in the center console and power up the laptop. The laptop can boot multiple operating systems so choose the QNX operating system when prompted.
- Switch the vehicle key switch to the accessory position. In the accessory position the

control computer in the trunk will be powered up in about 30 seconds. The VFD display will change to indicate that the control computer has successfully booted.

- Attach the ethernet cable that comes out of the center console to the laptop computer.
- Login to the laptop with the username `matt`. There is no password required.
- For new code to be installed on the control computer, the network connection between the two has to be working. This can be checked with either the command `alive` or `sin net`.
- There are multiple versions of the control code on the laptop. The directories holding the control code are labeled with a date format of the form `yyyymmdd` where `yyyy` indicates the year, `mm` the month, and `dd` the day. There is one subdirectory under this date coded directory, `99`. Inside this `99` subdirectory is the actual control code, which will be referred to as the “cc” directory.
- After changes are made to code in the `cc` directory, the executables need to be remade. This is done by issuing the command `make all` from the `cc` directory. Optionally, the command `make clean` can be issued to remove any stale object files or executables before running `make all`.
- Running `make all` only rebuilds the executables. The command `make install` has to be run to actually copy the required executables to the control computer. Since the `make install` command requires copying files to a “privileged” area of the control computer, the command `su` has to be run first. The correct sequence of commands to install a new version of the code is `make all ; su ; make install ; exit`.
- After running `make install`, rebooting the control computer will make the new code run automatically on boot.

The source code in the `cc` directory is split further into more subdirectories descriptively named for their intended purpose. These subdirectories are

cgi This directory holds further subdirectories for Common Gateway Interface programs that are meant to be run by the web server that starts automatically at boot. The most useful of these small programs, `trends.cgi`, is described later in more detail.

fcar The `fcar` subdirectory holds the main control code. If changing the behavior of the vehicle is required, this is the place to look.

fcard This directory holds the code for the `fcar` daemon. This is where changes have to be made if hardware I/O requirements change.

flogger The code for the `fcar` data logger is here. More information on retrieving data collected by the data logger is given later.

libclient This directory holds the code for the library of functions used to share data among the many small programs that use the common shared memory segment.

libftask Functions for simplifying task management are here.

monitor The code for the `mon` shared memory monitoring program is here.

test The `test` directory holds code for many small test programs that can be used to check basic functionality of the system.

vfd This directory holds the code for the program that takes values from shared memory and displays it on the VFD connected to a serial port.

The process of adding or subtracting variables that will be stored in shared memory is not as simple as it should be. For this reason, there are some “dead” variables in the shared memory definition that were either never used or no longer used. Since changes have to be made in multiple places, it is easier to just leave some unused variables in the shared memory segment if memory is available.

The process of changing the contents of the shared memory used by all of the fcar programs has a few steps. First the contents of the file `libfclient/fcar_common.h` relative to the `cc` directory has to be changed. This file is the real definition of the contents of the shared memory. If the added variable represents a value that needs to be calibrated from a voltage signal to engineering units, additional entries for calibrating the new signal have to also be added to `libfclient/fcar_common.h`. All calibration is done in the file `fcard/octagon_10.c` so changes need to be made there if the added value needs calibration. Default calibration values should be added to the file `fcard/fcard.conf` and the code in `fcard/read.conf.c` should be updated to allow parsing the new calibration values. The running control system also has the ability to change calibration values with a web browser interface. If this ability is required, changes have to be made to the `calibrate.c` source code in the `cgi/calibrate` directory. There is also an ability to log data with a web browser interface. If this feature is needed, changes have to be made to the code in the `cgi/dumper` directory. Similarly, if the variable needs to be logged to disk with the automatic flogger data logger, changes have to be made in the `flogger` directory. Finally, if the new variable is to show up in the shared memory monitor program, changes have to be made in the `mon` subdirectory.

The programs `fcard`, `fcar`, and `vfd` are started in a shell script, `fcarstart`, that is called on boot from the file `/etc/config/sysinit.1` on the control computer. The process of executing `make install` puts all of the required control programs and configuration files in the correct places. Executables are placed in `/opt/fcar/bin`, configuration files are placed in `/opt/fcar/etc`, CGI programs are copied to `/usr/local/apache/cgi-bin`, and other web server files are placed in subdirectories of `/usr/local/apache` on the Octagon control computer.

Appendix B

Real time Trend Graph Viewing

While the control code can be updated using the native QNX networking protocols, the procedure for viewing CGI generated history graphs of shared memory variables requires the use of the TCP/IP (Transmission Control Protocol / Internet Protocol) capabilities of QNX. For TCP/IP networking to work, both the server and the client machines have to be configured. Both the control computer and the development laptop computer were registered to run on the UTK ethernet network. As such, the two computers have some fixed TCP/IP configuration parameters, given in Table B.1. This configuration allows both computers to be used on the UTK campus network, but, during normal use, the two computers will only communicate with each other on the two node "car network." Since the regular campus nameserver, the network host that converts name queries to numerical addresses (octagon engr utk edu → 128.169.100.192), is not available from the "car network," and neither host acts as a nameserver, each host must be referred to by their numerical IP address.

TCP/IP connectivity can be tested by using the `ping` command to see if the other host is reachable. Once this is done, a web browser can be started on the laptop computer. If the laptop was booted to QNX, the *Voyager* browser can be used. Since the

Table B 1 TCP/IP configuration parameters

	Octagon computer	Laptop computer
host IP address	128 169 100 192	128 169 100 180
host IP name	octagon	pip
domain	engr utk edu	engr utk edu
subnet mask	255 255 252 0	255 255 252 0
gateway host IP	128 169 100 1	128 169 100.1
nameserver host IP	128 169 50 100	128 169 50 100

QNX browser requires a graphical user interface, the QNX GUI, *Photon*, has to be started with the command `ph` before the browser can be started. After the GUI has started, the web browser can be started with the command `voyager`. If the laptop computer was booted into another operating system, the standard procedure for starting a web browser on that OS should be used. Once the web browser is up, it should be pointed to the address `http://128.169.100.192/cgi-bin/trends.cgi`. Usage of the trends CGI program is fairly self-explanatory. Three possible drop-down lists are provided to choose the shared memory variables' history to graph.

The web server, Apache `httpd`, is automatically started at boot on the octagon control computer from the `/etc/config/sysinit 1` shell script. In addition to the web server, telnet and ftp services are also enabled at boot on the control computer. One notable use of the telnet service is that the special login name 'mon' can be used without a password to automatically start the shared memory monitor program, discussed in Section 4.6.1. Additionally, files can be transferred to and from the octagon control computer without using QNX native networking by using the ftp service.

Appendix C

Retrieving Logged Data

The fcar logger program, *flogger*, is automatically started at boot from the shell script, `/opt/fcar/bin/fcarstart`. The logging output of the program is by default stored in a file `/opt/fcar/var/flogger.log` on the control computer. Since this file can grow rather large, quickly, it should be moved from the control computer regularly. The data logger by default samples the contents of the shared memory every second, but this rate can be increased or decreased with a command line parameter. The format of the log file is comma separated text with a UNIX end-of-line marker (single line feed character). This format can easily be converted to other formats for data analysis. The data logger program prints a text header describing each field of the log file each time the program is started.

Moving the log file to the laptop computer running QNX is probably the simplest method of retrieving logged data. The command `mv //1/opt/fcar/var/flogger.log flogger.log` would move the log file to the current directory on the laptop computer. It is generally safe to move the log file while the data logger program is running. If it is running it will create a new log file if the file is moved or deleted. Once on the laptop computer, the log file can be copied to a DOS FAT formatted floppy disk. The command sequence for this would be `su , Dosfsys & , cp flogger.log /dos/a/ , slay Dosfsys ; exit` if the log file is to be copied to the first floppy disk drive. Another more cross-platform

method of retrieving the log file would be to ftp the data from the control computer to another host. The ftp command "DEL" could be used to delete the log file after copying it with a "GET" command.

Appendix D

Control Code Listing

D.1 fcard: Hardware Control Dæmon

D.1.1 Makefile

```
# requires gnu make

CC           =      cc
DEFINES      =      -DDEBUG
INCLUDES     =      -I /libfclient -I /libftask
QUIET        =      -Q -wx
OPTS         =      -Oraillnextm -4r -fp3 -fp187
#DEBUG       =      -g
CFLAGS       =      $(QUIET) $(DEBUG) $(OPTS) $(DEFINES) $(INCLUDES)
LIBS         =      -L /libfclient -lfclient -L /libftask -lftask
LDFLAGS      =      $(QUIET) $(DEBUG) $(LIBS) -T 1
FILES        =      fcard c daemon c read_conf c
FILES        +=     hardware_io c initshm c
FILES        +=     parse_cmdline.c octagon_io c
FILES        +=     octagon_init c kwh_meter c
OBJS         =      $(FILES c= o)
OUT          =      fcard
DEPEND       =      makedepend
BINDIR       =      /opt/fcar/bin/
ETCDIR       =      /opt/fcar/etc/

all          $(OUT)

$(OUT)       $(OBJS)

dep          $(DEPEND) -- $(CFLAGS) -D__QNX__ -- $(FILES)

install      $(OUT)
             cp -f $(OUT) $(BINDIR)
             cp -f fcard conf $(ETCDIR)

pre          $(FILES)
             $(RM) pre c
             $(CC) -E $(CFLAGS) $< >> pre c
```



```

ass·          $(FILES)
              $(CC) -S $(CFLAGS) $<

clean
              rm -f $(OBJS) $(OUT) core fcard log fcard pid * err

```

D.1.2 fcard.h

```

/* fcard h */

#ifndef FCARD_H
#define FCARD_H

/* std includes */
#include <fcntl h>
#include <task h>

/* defines */
#define _POSIX_C_SOURCE 199309L
#define KWH_DEVICE "/1/dev/ser2"

/* where to put stuff */
#ifndef DEBUG
/* put and look for files in the current directory */
#define LOGFILE "fcard log"
#define PIDFILE "fcard pid"
#define DEFAULT_CONFIG_FILENAME "fcard conf"
#else /* DEBUG */
#define LOGFILE "/opt/fcar/var/fcard log"
#define PIDFILE "/opt/fcar/var/fcard pid"
#define DEFAULT_CONFIG_FILENAME "/opt/fcar/etc/fcard conf"
#endif /* DEBUG */

/* min seconds between restarts */
#define RESTART_THRESHOLD 2

/* scheduling and priority default defines */
#define FCARD_SCHED_POLICY SCHED_FIFO
/* parent needs higher priority so we can send signals, etc */
#define FCARD_MANAGER_PRIO 27 /* note qnx non-root max = 19 */
#define FCARD_HARDWARE_PRIO 26
#define FCARD_KWHREADER_PRIO 22
#define FCARD_CONFIG_PRIO 16

#define FCARD_HARDWARE_HZ 100

/* structures */
/* see fcar_common h in libfclient */

/* global variables */
extern char *config_filename,
extern volatile struct shared_hw_data *hd, /* big chunk of shared mem */
extern int embedded, /* if files and such are available */
extern int do_hardware_10, /* set to 0 for debugging non-hardware stuff */

/* function prototypes */
void daemonize(void),
void write_pid(void),
void exit_cleanly(int ignored),

```

```

void hardware_io_stop(int ignored),
void hardware_io(void *ignored),
void initshm(void),
void octagon_init(void),
void octagon_io(void),
void usage(void),
void parse_cmdline(int argc, char *argv[]),
void read_conf(void *stay),
void do_read_conf(void),
void do_reread(int ignored),

```

```

void read_kwh_meter_stop(int ignored),
void read_kwh_meter(void *unused),

```

```
#endif
```

D.1.3 octagon_io.h

```

/* octagon_io h */

#ifndef OCTAGON_IO_H
#define OCTAGON_IO_H
/* information specific to octagon I/O hardware */

#ifdef __QNX__
#include <sys/inline.h> /* qnx inline asm */
/* try to make IN and OUT a little more portable? */
#define in_8 inb
#define in_16 inw
#define in_32 inl
#define out_8(port,value) outb(port,value)
#define out_16(port,value) outw(port,value)
#define out_32(port,value) outl(port,value)
#endif

/* used for octagon 5066 cpu card hardware watchdog timer */
#define ENABLE_WATCHDOG() out_8(0x20c, in_8(0x20c) | 0x40)
#define PET_WATCHDOG() in_8(0x20c)
#define DISABLE_WATCHDOG() out_8(0x20c, in_8(0x20c) & ~0x40)

/* used in 5710 card initialization */
#define CR_OFFSET 0x0B /* 82C55 control register offset */

/* used in 5710 analog input */
#define CONVERT 0xFF /* MUX convert command */
#define CH_SELECT 0x09 /* analog input channel select offset */
#define HIGH8 0x02 /* bits 4 to 11 of read analog voltage (offset) */
#define LOW4 0x03 /* bits 0 to 3 of read analog voltage (offset) */

/***** analog input macros *****/
/* swap - exchange the high and low 8 bit registers, divide by 16 */
unsigned swap(unsigned val),
#pragma aux swap = "xchg ah,al" \
                  "shr eax,4" \
                  __parm __nomemory [eax] __value [eax] \
                  __modify __exact __nomemory [eax],

#define DELAY_PORT 0x80
#define DELAY_VALUE 0
#define delay_four_us() out_8(DELAY_PORT,DELAY_VALUE), \

```

```

        out_8(DELAY_PORT,DELAY_VALUE), \
        out_8(DELAY_PORT,DELAY_VALUE), \
        out_8(DELAY_PORT,DELAY_VALUE)

#define init_convert(port, ch) out_8(port+CH_SELECT, ch), \
        delay_four_us(), \
        out_8(port, CONVERT), \
        while((in_8(port) & 1) == 0)

#define get_conv(port) in_16(port+HIGH8)

#define getAnalog(a) swap(get_conv(a))

#define delay_five_us() out_8(DELAY_PORT,DELAY_VALUE), \
        out_8(DELAY_PORT,DELAY_VALUE), \
        out_8(DELAY_PORT,DELAY_VALUE), \
        out_8(DELAY_PORT,DELAY_VALUE), \
        out_8(DELAY_PORT,DELAY_VALUE)

/***** analog input macros *****/

/* used in 5710 analog output (offsets) */
#define DAC_CH_ZERO 0x0C
#define DAC_CH_ONE 0x0E

/* 5710 digital io modes */

/* port A port C */
/* ----- */
#define DMODE_BOTH_OUT 0x80 /* out out */
#define DMODE_AOUT_CIN 0x89 /* out in */
#define DMODE_AIN_COUT 0x90 /* in out */
#define DMODE_BOTH_IN 0x99 /* in in */

--

#define DPORTA 0x08
#define DPORTB 0x09 /* this still exists */
#define DPORTC 0x0A

/* used in 5750 analog output (offsets) */
#define ZERO_LEAST 0x0
#define ZERO_MOST 0x1
#define ONE_LEAST 0x2
#define ONE_MOST 0x3
#define TWO_LEAST 0x4
#define TWO_MOST 0x5
#define THREE_LEAST 0x6
#define THREE_MOST 0x7
#define FOUR_LEAST 0x8
#define FOUR_MOST 0x9
#define FIVE_LEAST 0xA
#define FIVE_MOST 0xB
#define SIX_LEAST 0xC
#define SIX_MOST 0xD
#define SEVEN_LEAST 0xE
#define SEVEN_MOST 0xF

/* base port addresses */
#define CARD0 0x100 /* slower 5710 card */
#define CARD1 0x110 /* faster 5710 card */
#define CARD2 0x120 /* 5750 card */

```

```

/* 5710 conversion macros */
#define INT2VOLTS(d) \
    ((10 OF * (float)(d) / 4095 OF) - 5 OF) /* gain 1 */
#define VOLTS2INT(f) \
    (unsigned)(((f) + 10 OF) / (20 OF / 4096 OF)) /* -10 to +10 vdc */

/* 5750 conversion macros (outputs only) */
#define FULLVOLTS2INT(f) \
    (unsigned)(((f) + 5 OF) / (10 OF / 4096 OF)) /* -5 to +5 vdc */

#define HALFVOLTS2INT(f) \
    (unsigned)((f) / (5 OF / 4096 OF)) /* 0 to +5 vdc */

/* analog output macros */
#define OUTPUT5710(card, ch, value) \
    out_16((card)+(ch), ((value) > 4095U) ? 4095U (value))

```

```

/***** private structures for octagon_io c *****/

```

```

struct io5710Integer
{
    unsigned AnalogIn[16],
    unsigned AnalogOut[2],
    unsigned DigitalInC[8],
    unsigned DigitalInA[8],
    unsigned DigitalOutC[8],
    unsigned DigitalOutA[8],
},

struct o5750Integer
{
    unsigned AnalogOut[8], /* octal-dac */
},

struct io5710Volts
{
    float AnalogIn[16],
    float AnalogOut[2],
},

struct o5750Volts
{
    float AnalogOut[8],
},

#endif

```

D.1.4 octagon_io_map.h

```

/* octagon_io_map h */

#ifndef OCTAGON_IO_MAP
#define OCTAGON_IO_MAP

/* how things are connected */
/* analog */
#define iEmMotorSpeedChannel vcard0 AnalogIn[1]
#define iGenMotorSpeedChannel vcard1 AnalogIn[1]

```

```

#define iIceEngineSpeedChannel          vcard1 AnalogIn[3]
#define iAccelPedalLevelChannel         vcard1 AnalogIn[5]
#define iBrakePedalLevelChannel         vcard1 AnalogIn[7]
#define iActualEmTorqueChannel          vcard1 AnalogIn[9]
#define iFuelPressureChannel            vcard1 AnalogIn[11]
#define iEmCurrentChannel               vcard1 AnalogIn[13]
#define iGenCurrentChannel              vcard1 AnalogIn[15]
/* #define iBattPackTempChannel          vcard0 AnalogIn[3]      */
#define iBattPackVoltageChannel         vcard0 AnalogIn[5]
#define iBattPackCurrentChannel         vcard0 AnalogIn[7]
#define iEmRotorTempChannel            vcard0 AnalogIn[9]
#define iEmInvTempChannel              vcard0 AnalogIn[11]
#define iTpsFeedbackChannel            vcard0 AnalogIn[13]

#define oGenSpeedReqChannel             vcard1 AnalogOut[0]
#define oGenRegenLimitChannel           vcard1 AnalogOut[1]
#define oEmAccelReqChannel              vcard2 AnalogOut[0]
#define oEmBrakeReqChannel              vcard2 AnalogOut[1]
#define oIceThrottlePosChannel          vcard2 AnalogOut[2]
#define oDisSpeedo0Channel              vcard2 AnalogOut[4]
#define oDisSpeedo1Channel              vcard2 AnalogOut[5]
#define oPwrToDigRacksChannel           vcard2 AnalogOut[7]

/* digital */
/* shifter info
 * pins 0 SHID1 PC-7
 *      1 SHID2 PC-6
 *      2 SHID3 PC-5
 *      3 SHID4 PC-4
 *
 *          conn pins
 * select  pos    0 1 2 3    decimal hex    hex << 4
 * -----
 * Park    A      0 0 0 0     0      0x0    0x00
 * Reverse B      0 1 0 0     4      0x4    0x40
 * Neutral C      0 0 1 0     2      0x2    0x20
 * DriveEcon D    0 0 0 1     1      0x1    0x10
 * DriveSport E   1 1 0 0    12     0xc    0xc0
 * ZEV      F     1 0 1 0    10     0xa    0xa0
 * (not used) G   1 0 0 1     9      0x9    0x90
 * (not used) H   1 0 0 0     8      0x8    0x80
 *
 */
#define iParkBitPattern                  0x00
#define iReverseBitPattern               0x40
#define iNeutralBitPattern               0x20
#define iDriveEconBitPattern             0x10
#define iDriveSportBitPattern            0xc0
#define iZEVBitPattern                   0xa0
#define iDynoBitPattern                   0x90

#define iHvacBit                          icard0 DigitalInC[1]
#define iEmTempWarnBit                    icard0 DigitalInC[0]
#define iEmControllerReadyBit             icard0 DigitalInA[6]
#define iEmFaultIndicatorBit              icard0 DigitalInA[7]
#define iEmOvertempIndicatorBit           icard0 DigitalInA[5]
#define iGenTempWarnBit                   icard0 DigitalInA[4]
#define iGenControllerReadyBit            icard0 DigitalInA[3]
#define iGenFaultIndicatorBit             icard0 DigitalInA[1]
#define iGenDirectionIndicatorBit         icard0 DigitalInA[2]
#define iIceFaultIndicatorBit             icard0 DigitalInA[0]

```

```

#define oIceStarterBit          1card1 DigitalOutA[3]
#define oTecEnableBit          1card1 DigitalOutA[4]
#define oEmEnableBit           1card1 DigitalOutA[7]
#define oEmDirectionBit       1card1 DigitalOutA[6]
#define oGenEnableBit          1card1 DigitalOutA[5]
#define oPwrSteeringEnableBit  1card1 DigitalOutA[2]
#define oThrottlePwrCycleBit   1card1 DigitalOutA[1]
#define oSmartChargerEnableBit 1card1 DigitalOutA[0]

#endif

```

D.1.5 fcard.c

```

/* fcard c */
#include <stdio.h>
#include <unistd.h>          /* unlink */
#include <stdlib.h>         /* exit */
#include <time.h>           /* time */
#include "fcard.h"

/* these can be static because they don't have to talk to each other */
static ftask *config_task, *hardware_task, *kwh_task, *ret,

int main(int argc, char *argv[])
{
    time_t t = 0, told = 0, tdelta,
    int     priority, hz, rv,

    /* parse command line */
    parse_cmdline(argc, argv),

    /* need to alloc some for tasks */
    config_task = (ftask*)calloc(1, sizeof(ftask)),
    hardware_task = (ftask*)calloc(1, sizeof(ftask)),
    kwh_task = (ftask*)calloc(1, sizeof(ftask)),
    ret = (ftask*)calloc(1, sizeof(ftask)),

    if ('config_task || 'hardware_task || 'kwh_task || 'ret )
    {
        die("calloc failed for task structs"),
    }

    /* detach etc */
    daemonize(),

    notice("log started"),

    if ('embedded)
    {
        /* record our pid */
        write_pid(),
    }

    /* setup shared memory */
    initshm(),

    /* try reading config file */
    read_conf(NULL),

```

```

/* setup scheduler parameters */
ftask_sched_adjst_self(FCARD_SCHED_POLICY,
                      hd->fcard_rv fcard_manager_prio),

/* startup config child task */
priority = hd->fcard_rv fcard_config_task_prio,
rv = ftask_init(config_task,
                FCARD_SCHED_POLICY, /* policy */
                priority,           /* priority */
                read_conf,          /* start_routine */
                (void *)1,          /* start_routine_arg */
                NULL,               /* cleanup_routine */
                1,                  /* allow_trigger */
                0,                  /* allow_periodic_timer */
                0),                /* periodic_timer_hz */

if (rv) die("ftask_init config_task"),

if (ftask_create(config_task) == -1)
    die("ftask_create(config_task)"),

/* start a new task to gather stats from kwh meter */
priority = hd->fcard_rv fcard_kwh_task_prio,
rv = ftask_init(kwh_task,
                FCARD_SCHED_POLICY, /* policy */
                priority,           /* priority */
                read_kwh_meter,     /* start_routine */
                NULL,               /* start_routine_arg */
                read_kwh_meter_stop, /* cleanup_routine */
                0,                  /* allow_trigger */
                0,                  /* allow_periodic_timer */
                0),                /* periodic_timer_hz */

if (rv) die("ftask_init kwh_task"),

if (ftask_create(kwh_task) == -1)
    die("ftask_create(kwh_task)"),

/* start a new task to talk to hardware */
priority = hd->fcard_rv fcard_hardware_task_prio,
hz = hd->fcard_rv fcard_hardware_timer_hz,

rv = ftask_init(hardware_task,
                FCARD_SCHED_POLICY, /* policy */
                priority,           /* priority */
                hardware_io,        /* start_routine */
                NULL,               /* start_routine_arg */
                hardware_io_stop,   /* cleanup_routine */
                0,                  /* allow_trigger */
                1,                  /* allow_periodic_timer */
                hz),                /* periodic_timer_hz */

if (rv) die("ftask_init hardware_task"),

if (ftask_create(hardware_task) == -1)
    die("ftask_create(hardware_task)"),

/* allow rereading config file */
ftask_register_reread_self(do_reread),

```

```

/* allow for exiting in an orderly manner */
ftask_register_cleanup_self(exit_cleanly),

while(hd)      /* shared mem still mapped */
{
    ftask_wait_on_tasks(ret),

    t = time(NULL),
    tdelta = t - told,

    if (tdelta < RESTART_THRESHOLD)
    {
        warn("respawning too fast must be a problem"),
        exit_cleanly(0),
    }

    if (ftask_same(ret, hardware_task))
    {
        warn("hardware I/O task died?, trying restart"),
        ftask_create(hardware_task),
        told = time(NULL),
    }

    if (ftask_same(ret, kwh_task))
    {
        warn("kwh task died?, trying restart"),
        ftask_create(kwh_task),
        told = time(NULL),
    }

    if (ftask_same(ret, config_task))
    {
        warn("config task died? trying restart"),
        ftask_create(config_task),
        told = time(NULL),
    }
}

/* won't get here */
return EXIT_FAILURE,
}

void exit_cleanly(int ignored)
{
    /* do cleanup stuff here */
    notice("fcard exiting reaping children"),

    /* first kill off known children */
    if (ftask_delete(hardware_task) != 0)
        warn("can't kill hardware_task"),

    if (ftask_delete(kwh_task) != 0)
        warn("can't kill kwh_task"),

    if (ftask_delete(config_task) != 0)
        warn("failed to kill config task"),

    /* get rid of shared memory segment */
    if (fclient_delete_shm() != 0)
        warn("fclient_delete_shm"),
}

```



```

    /* delete pid file */
    if (('embedded) && (unlink(PIDFILE) != 0))
        warn("unlink(PIDFILE)",

    /* just to be complete */
    free(ret),
    free(hardware_task),
    free(config_task),
    free(kwh_task),
    notice("all done"),

    exit(EXIT_SUCCESS),
}

void do_reread(int ignored)
{
    /* relay to config task */
    ftask_trigger(config_task),
}

```

D.1.6 parse_cmdline.c

```

/* parse_cmdline c */
#include <stdio h>
#include <stdlib h>
#include <string h>    /* strdup */
#include <sys/stat h>
#include <unistd h>    /* stat, getopt */
#include "fcard h"

/* real definition of config_filename */
char *config_filename,

/* real definition of embedded */
int embedded,

/* real definition of do_hardware_io */
int do_hardware_io,

void usage(void)
{
    printf("Usage fcard [-e] [-f config_file] [-d] [-h]\n"),
    printf("    -e enables embedded operation\n"),
    printf("    -f specifies the path to a config file\n"),
    printf("    -d disable actual hardware I/O (for testing)\n"),
    printf("    -h shows this usage message\n"),
    exit(EXIT_FAILURE),
}

/* look for various options */
void parse_cmdline(int argc, char *argv[])
{
    struct stat s,
    int c,
    int errflag = 0,
    config_filename = DEFAULT_CONFIG_FILENAME,
    embedded = 0,
    do_hardware_io = 1,

    while( (c=getopt(argc,argv, "ef hd")) != -1)

```

```

    {
        switch (c)
        {
            case 'e'
                embedded = 1,
                break,
            case 'f'
                config_filename = strdup(optarg),
                if (config_filename == NULL)
                    die("strdup"),
                break,
            case 'd'
                do_hardware_io = 0,
                break,
            case 'h'
                usage(),
                break,
            case '?'
                ++errflag,
                break,
        }
        if (errflag) usage(),
    }

    /* if not in "embedded" mode, config file should exist */
    if (!embedded)
    {
        if (stat(config_filename, &s) != 0)
            die("can't find specified config file"),
    }
    /* don't check for file otherwise, because regular filesystem's
     * not up yet */

    return,
}

```

D.1.7 daemon.c

```

/* daemon c */

#include <stdio.h>           /* freopen, fopen, fgets, sprintf */
#include <stdlib.h>         /* exit, atoi */
#include <unistd.h>         /* fork, setsid, chdir */
#include <sys/types.h>      /* fork, chdir, umask */
#include <sys/stat.h>       /* umask, stat */
#include <string.h>         /* memset */
#include <sys/types.h>      /* kill */
#include <signal.h>         /* kill */
#include "fcard.h"

/*
 * daemonize() run new process in background and
 * detach from controlling terminal
 */

void daemonize()
{
    /* immediately go to the background */
    switch (fork())
    {

```

```

        case 0 break,                /* child exits switch */
        case -1 die("fork"),         /* problem with fork */
        default exit(EXIT_SUCCESS), /* exit original process */
    }

    /* become new process group leader */
    if (setsid() < 0) /* would fail if invoked from a session leader */
        die("setsid"),

#ifdef DEBUG
    if ('embedded')
    {
        /* change to root dir to avoid problems
         * with mounted filesystems */
        if ( (chdir("/") < 0) die("chdir"),
    }
#endif

    /* reset umask so that we have total control over file creation */
    umask(0), umask(022),

#ifdef DEBUG
    if ('embedded')
    {
        /* reestablish standard file descriptors */
        if (freopen("/dev/null", "r", stdin) == NULL)
            die("reopen stdin"),

        if (freopen("/dev/null", "w", stdout) == NULL)
            die("reopen stdout"),

#ifdef DEBUG
        /* redirect stderr to our logfile */
        if (freopen(LOGFILE, "a", stderr) == NULL)
            die("reopen stderr"),
#endif
    }
#endif

    return,

}

/* dump our pid for easy killing */
void write_pid()
{
    FILE *pid_file,
    char buf[32],
    pid_t p,
    struct stat s,

    memset(&s, 0, sizeof(struct stat)),

    if ( (stat(PIDFILE, &s) == 0) && (s.st_size > 0) )
    {
        /* oops, file is already there */
        /* try to open it to see if that process is still running */
        if ((pid_file = fopen(PIDFILE, "r")) == NULL)
            die("can't open old pid file"),

        if ( fgets(buf, sizeof(buf), pid_file) == NULL)
            die("fgets"),
    }
}

```

```

    p = atoi(buf),
    if ((p != getpid()) && (kill (p, 0) == 0))
    {
        /* another daemon is running, exit NOW, not cleanly */
        sprintf(buf, "another pid (%d) is running", p),
        die(buf),
    }
    /* if we get here, the old process isn't running */
    /* just continue */
    fclose(pid_file),
}

if ((pid_file = fopen(PIDFILE, "w")) == NULL)
    die("can't write pid file"),

fprintf(pid_file, "%d\n", getpid()),

fclose(pid_file),

return,
}

```

D.1.8 initshm.c

```

/* initshm.c */

#include "fcard.h"

/* real definition of shared memory pointer */
volatile struct shared_hw_data *hd,

void initshm(void)
{
    if ((hd = fclient_create_shm()) == NULL) die("fclient_create_shm"),

    /* fill in fcard's shm with sensible default values */
    /* should be the same as fcard conf file */
    hd->fcard_rv fcard_hardware_timer_hz      = FCARD_HARDWARE_HZ,
    hd->fcard_rv fcard_manager_prio           = FCARD_MANAGER_PRIO,
    hd->fcard_rv fcard_hardware_task_prio     = FCARD_HARDWARE_PRIO,
    hd->fcard_rv fcard_kwh_task_prio         = FCARD_KWHREADER_PRIO,
    hd->fcard_rv fcard_config_task_prio      = FCARD_CONFIG_PRIO,

    hd->cal 1EmMotorSpeedSen                  = 2000 OF,
    hd->cal 1EmMotorSpeedOff                  = 0 OF,

    hd->cal 1GenMotorSpeedSen                  = -1450 OF,
    hd->cal 1GenMotorSpeedOff                  = 100 OF,

    hd->cal 1IceEngineSpeedSen                 = 3000 OF,
    hd->cal 1IceEngineSpeedOff                 = 0 OF,

    hd->cal 1AccelPedalLevelSen                 = 0 2326F,
    hd->cal 1AccelPedalLevelOff                 = -0 1628F,

    hd->cal 1BrakePedalLevelSen                 = 0 3333F,
    hd->cal 1BrakePedalLevelOff                 = 0 OF,

    hd->cal 1ActualEmTorqueSen                  = 100 OF,
    hd->cal 1ActualEmTorqueOff                  = -250 OF,

```

```

hd->cal 1FuelPressureSen          = 1000 OF,
hd->cal 1FuelPressureOff         = -1000 OF,

hd->cal 1EmCurrentNegSen         = 80 OF,
hd->cal 1EmCurrentNegOff        = -200 OF,

hd->cal 1EmCurrentPosSen         = 120 OF,
hd->cal 1EmCurrentPosOff        = -300 OF,

hd->cal 1GenCurrentSen           = 50 OF,
hd->cal 1GenCurrentOff           = 0 OF,

hd->cal 1BattPackTempSen         = 0 OF,
hd->cal 1BattPackTempOff        = 0 OF,

hd->cal 1BattPackVoltageSen      = 100 OF,
hd->cal 1BattPackVoltageOff     = 0 OF,

hd->cal 1BattPackCurrentSen      = 200 OF,
hd->cal 1BattPackCurrentOff     = -500 OF,

hd->cal 1EmRotorTempSen         = 40 OF,
hd->cal 1EmRotorTempOff        = 0 OF,

hd->cal 1EmInvTempSen           = 40 OF,
hd->cal 1EmInvTempOff          = 0 OF,

hd->cal 1TpsFeedbackSen          = 0 3401F,
hd->cal 1TpsFeedbackOff         = -0 2041F,

hd->cal 0GenSpeedReqSen          = 1450.0F,
hd->cal 0GenSpeedReqOff         = -100 OF,

hd->cal 0GenRegenLimitSen        = -0 10F,
hd->cal 0GenRegenLimitOff       = 0 OF,

hd->cal 0EmAccelReqSen           = 0 25F,
hd->cal 0EmAccelReqOff          = -0 125F,

hd->cal 0EmBrakeReqSen           = 0 25F,
hd->cal 0EmBrakeReqOff          = -0 125F,

hd->cal 0IceThrottlePosSen       = 0 339F,
hd->cal 0IceThrottlePosOff      = -0 220F,

hd->cal 0DisSpeedo0Amplitude     = -5 OF,
hd->cal 0DisSpeedo0PhaseAngle   = 1 57079632679489661923F,

hd->cal 0DisSpeedo1Amplitude     = 5 OF,
hd->cal 0DisSpeedo1PhaseAngle    = 0 OF,

hd->cal 0PwrToDigRacksSen        = 5 OF,
hd->cal 0PwrToDigRacksOff       = 0 OF,

```

```

}
```

D.1.9 fcard.conf

```

# fcard configuration file
```

```

# fcard runtime configuration values

fcard_hardware_timer_hz      100    # irq 0 toggles at rate of 100 Hz
#fcard_hardware_timer_hz    20      # split 100 Hz by 5 20 Hz
fcard_hardware_timer_hz      1       # 1 Hz

fcard_manager_prio           27
fcard_hardware_task_prio     26
fcard_kwh_task_prio          22
fcard_config_task_prio       16

# calibration values
##### inputs #####

1EmMotorSpeedSen             2000    # rpm per volt after vdiv
1EmMotorSpeedOff              0       # rpm @ 0 volts

1GenMotorSpeedSen            -1450   # rpm per volt after vdiv
1GenMotorSpeedOff             100      # rpm @ zero volts

1IceEngineSpeedSen           3000    # rpm per volt
1IceEngineSpeedOff            0       # rpm @ zero volts

1AccelPedalLevelSen          0 2326  # percent per volt
1AccelPedalLevelOff          -0 1628 # percent @ zero volts

1BrakePedalLevelSen          0 3333  # percent per volt
1BrakePedalLevelOff          0         # percent @ zero volts

1ActualEmTorqueSen            100     # N m per volt after vdiv
1ActualEmTorqueOff           -250     # N m @ 0 volts after vdiv

1FuelPressureSen             1000     # psi per volt
1FuelPressureOff             -1000   # psi @ zero volts

1EmCurrentNegSen              80      # amps per volt when neg after vdiv
1EmCurrentNegOff             -200    # amps @ 0 volts when neg after vdiv

1EmCurrentPosSen              120     # amps per volt when pos after vdiv
1EmCurrentPosOff             -300    # amps @ 0 volts when pos after vdiv

1GenCurrentSen                50      # amps per volt after vdiv
1GenCurrentOff                0       # amps @ zero volts

1BattPackTempSen              0       # degrees C per volt
1BattPackTempOff              0       # degrees C @ zero volts

1BattPackVoltageSen           100     # terminal voltage per volt
1BattPackVoltageOff           0       # terminal voltage @ zero volts

1BattPackCurrentSen           200     # amps per volt
1BattPackCurrentOff           -500    # amps @ zero volts

1EmRotorTempSen              40      # degrees C per volt
1EmRotorTempOff               0       # degrees C @ zero volts

1EmInvTempSen                 40      # degrees C per volt
1EmInvTempOff                  0       # degrees C @ zero volts

1TpsFeedbackSen               0 3401  # percent open per volt
1TpsFeedbackOff               -0 2041  # percent open @ zero volts

```

```
##### outputs #####
```

```
oGenSpeedReqSen      1450      # rpm per volt
oGenSpeedReqOff      -100      # rpm @ zero volts

oGenRegenLimitSen    -0 10     # percent regen per volt
oGenRegenLimitOff    0           # percent regen @ zero volts

oEmAccelReqSen       0 25      # % per volt
oEmAccelReqOff       -0 125    # % @ 0 volts

oEmBrakeReqSen       0 25      # percent per volt
oEmBrakeReqOff       -0 125    # percent @ zero volts

oIceThrottlePosSen   0 339     # percent open per volt
oIceThrottlePosOff   -0 220    # percent open @ zero volts

## for speedo eqn
## output voltage = Amplitude * sin( VehicleSpeedInMph * pi/92.5 + PhaseAngle)
## sin phase = 0    cos phase = pi/2

oDisSpeedo0Amplitude -5
oDisSpeedo0PhaseAngle 1 57079632679489661923 # pi/2

oDisSpeedo1Amplitude 5
oDisSpeedo1PhaseAngle 0

oPwrToDigRacksSen    5           # 1 == 5v out, 0 == 0v out
oPwrToDigRacksOff    0           # 0v out @ 0
```

D.1.10 read_conf.c

```
/* read_conf.c */
#include <stdio.h>      /* fopen, rewind, fgets */
#include <stdlib.h>     /* exit, atof, atoi */
#include <string.h>    /* strtok, strstr, */
#include "fcard.h"

/* note #key "stringifies" key, ( #key becomes "key" ) */
#define FINDFLOAT(key) if(strstr(s, #key) != NULL) \
    { \
        s = strtok(NULL, sep), \
        hd->cal key = atof(s), \
    }

#define FINDINT(key) if(strstr(s, #key) != NULL) \
    { \
        s = strtok(NULL, sep), \
        hd->fcard_rv key = atoi(s), \
    }

void read_conf(void *stay)
{
    if (stay)
    {
        /* config task sticks around waiting
        * for reread requests */
        notice("config reader ready"),
        while (hd)
        {
```

```

        ftask_trigger_block(),
        do_read_conf(),
    }
}
else
    /* read configuration first time around */
    do_read_conf(),
return,
}

void do_read_conf(void)
{
    FILE *file,
    const char sep[] = "\t", /* field delimiters */
    const char comment[] = "#", /* signifies comment */
    char *s, /* current key */
    char buf[256], /* line buffer */
    char msg[256], /* error message buffer */

    notice("reading config file"),

    /* open the config file */
    if ( (file = fopen(config_filename, "r")) != NULL)
    {
        /* make sure that we are at the beginning */
        rewind(file),

        /* read lines one at a time and process them */
        while ( fgets(buf, sizeof(buf), file) != NULL)
        {
            s = strtok(buf, sep),
            /* if a beginning comment, skip line */
            if (strstr(s, comment) != NULL) continue,

            /* don't need no stinking semicolons */
            /* inputs */
            FINDFLOAT(1EmMotorSpeedSen) else
            FINDFLOAT(1EmMotorSpeedOff) else
            FINDFLOAT(1GenMotorSpeedSen) else
            FINDFLOAT(1GenMotorSpeedOff) else
            FINDFLOAT(1IceEngineSpeedSen) else
            FINDFLOAT(1IceEngineSpeedOff) else
            FINDFLOAT(1AccelPedalLevelSen) else
            FINDFLOAT(1AccelPedalLevelOff) else
            FINDFLOAT(1BrakePedalLevelSen) else
            FINDFLOAT(1BrakePedalLevelOff) else
            FINDFLOAT(1ActualEmTorqueSen) else
            FINDFLOAT(1ActualEmTorqueOff) else
            FINDFLOAT(1FuelPressureSen) else
            FINDFLOAT(1FuelPressureOff) else
            FINDFLOAT(1EmCurrentNegSen) else
            FINDFLOAT(1EmCurrentNegOff) else
            FINDFLOAT(1EmCurrentPosSen) else
            FINDFLOAT(1EmCurrentPosOff) else
            FINDFLOAT(1GenCurrentSen) else
            FINDFLOAT(1GenCurrentOff) else
            FINDFLOAT(1BattPackTempSen) else
            FINDFLOAT(1BattPackTempOff) else
            FINDFLOAT(1BattPackVoltageSen) else

```



```

    FINDFLOAT(iBattPackVoltageOff) else
    FINDFLOAT(iBattPackCurrentSen) else
    FINDFLOAT(iBattPackCurrentOff) else
    FINDFLOAT(iEmRotorTempSen) else
    FINDFLOAT(iEmRotorTempOff) else
    FINDFLOAT(iEmInvTempSen) else
    FINDFLOAT(iEmInvTempOff) else
    FINDFLOAT(iTpsFeedbackSen) else
    FINDFLOAT(iTpsFeedbackOff) else
    /* outputs */
    FINDFLOAT(oGenSpeedReqSen) else
    FINDFLOAT(oGenSpeedReqOff) else
    FINDFLOAT(oGenRegenLimitSen) else
    FINDFLOAT(oGenRegenLimitOff) else
    FINDFLOAT(oEmAccelReqSen) else
    FINDFLOAT(oEmAccelReqOff) else
    FINDFLOAT(oEmBrakeReqSen) else
    FINDFLOAT(oEmBrakeReqOff) else
    FINDFLOAT(oIceThrottlePosSen) else
    FINDFLOAT(oIceThrottlePosOff) else
    FINDFLOAT(oDisSpeedo0Amplitude) else
    FINDFLOAT(oDisSpeedo0PhaseAngle) else
    FINDFLOAT(oDisSpeedo1Amplitude) else
    FINDFLOAT(oDisSpeedo1PhaseAngle) else
    FINDFLOAT(oPwrToDigRacksSen) else
    FINDFLOAT(oPwrToDigRacksOff) else

    FINDINT(fcard_hardware_timer_hz) else
    FINDINT(fcard_manager_prio) else
    FINDINT(fcard_hardware_task_prio) else
    FINDINT(fcard_kwh_task_prio) else
    FINDINT(fcard_config_task_prio) else
    if (strlen(s) == 1)
        continue, /* blank line */
    else
    {
        sprintf(msg,
            "unknown key %s in config",s),
        warn(msg),
    }
    continue,
}

fclose(file),
notice("done reading config file"),
}
else
warn("can't read config file, using defaults"),

return,
}

```

D.1.11 kwh_meter.c

```

/* kwh_meter c */
#include <stdio h> /* fdopen */
#include <stdlib h> /* atof, exit */
#include <string h> /* strtok */
#include <unistd h> /* close */
#include <sys/types h> /* open */

```

```

#include <sys/stat h> /* open */
#include <fcntl h> /* open */
#include "fcard h"

#define MAX_SANE_AMPS 300 OF
#define MAX_SANE_VOLTS 500 OF

/* sample kwh-meter line
 * time(sec), batt_kwhrs, batt_amps, batt_volts, batt_ahrs, apu_kwhrs, \
 * apu_amps, apu_ahrs, apu, cng_press, cng_temp
 */

static int kwh_fd,

void read_kwh_meter_stop(int ignored)
{
    if (kwh_fd) close(kwh_fd),
    notice("kwh reader stopped"),
    exit(EXIT_SUCCESS),
}

void read_kwh_meter(void *unused)
{
    FILE *kwh_in, /* kwh meter file */
    const char sep[] = ",", /* fields separated by commas */
    char *s,
    char buf[256],
    float amps, volts,

    /* if no hardware io is requested, just do nothing */
    if (!do_hardware_io)
    {
        notice("kwh_reader doing nothing"),
        while(1) ftask_delay(1 OF),
    }

    /* open the serial port to which the kwh meter is attached */
    if ((kwh_fd = open(KWH_DEVICE, O_RDONLY | O_NOCTTY)) == -1)
        die("open"),
    if ((kwh_in = fdopen(kwh_fd, "r")) == NULL)
        die("fdopen"),

    /* going to assume that the serial port
     * has already been setup correctly */

    notice("kwh_reader started"),

    /* this fgets blocks until a \n terminated line comes
     * in though the serial port (about 1 Hz ) */
    while ( (fgets(buf, sizeof(buf), kwh_in)) != NULL)
    {
        s = strtok(buf, sep),
        /* skip first two fields */
        s = strtok(NULL, sep),
        s = strtok(NULL, sep),
        amps = atof(s), /* third = amps */
        s = strtok(NULL, sep),
        volts = atof(s), /* fourth = volts */

        /* do a sanity check on the read values */
        if ((amps < -MAX_SANE_AMPS) || (amps > MAX_SANE_AMPS))

```

```

        {
            sprintf(buf, "read insane value for amps %f", amps),
            warn(buf),
        }else
            /* shared mem gets current value */
            hd->cv[hd->active] kwh_amps = amps,

        if ((volts < -MAX_SANE_VOLTS) || (volts > MAX_SANE_VOLTS))
        {
            sprintf(buf, "read insane value for volts %f", volts),
            warn(buf),
        }else
            /* shared mem gets current value */
            hd->cv[hd->active] kwh_volts = volts,

    }

    close(kwh_fd),
    die("fgets failed"),
}

```

D.1.12 hardware_io.c

```

/* hardware_io.c */

#include <string.h>          /* memcpy */
#include <stdlib.h>         /* exit */
#include <ftask.h>          /* ftask_periodic_timer_block */
#include "fcard.h"
#include "octagon_io.h"     /* DISABLE_WATCHDOG */

static unsigned loop_counter,

void hardware_io_stop(int ignored)
{
    if (do_hardware_io)
    {
        DISABLE_WATCHDOG(),
        notice("watchdog timer disabled"),
    }
    notice("hardware io stopped"),
    exit(EXIT_SUCCESS),
}

void hardware_io(void *ignored)
{
    notice("hardware_io started"),

    if(do_hardware_io)
    {
        octagon_init(),
        ENABLE_WATCHDOG();
        notice("watchdog timer enabled"),
    }

    while(1)
    {
        if (do_hardware_io)
        {

```

```

        PET_WATCHDOG(), /* fixed timeout is ~ 1 5 sec */
        octagon_io(),
    }
    else
    {
        if (!(++loop_counter %
            hd->fcard_rv fcard_hardware_timer_hz))
            /* do this every 1 second */
        {
            int oldpointer = hd->active,
            /* if at bottom, loop back to the top */
            if(hd->active == (SECONDS_TO_STORE-1))
                hd->active = 0,
            else
                hd->active++,

            /* ok, we moved the pointer to active data,
            * copy the old pointer's data to prevent
            * data loss (I hope this works)
            */
            memcpy(&hd->cv[hd->active],
                &hd->cv[oldpointer],
                sizeof(struct calculated_values)),

            memcpy(&hd->digital[hd->active],
                &hd->digital[oldpointer],
                sizeof(struct digital_hardware)),

            memcpy(&hd->analog[hd->active],
                &hd->analog[oldpointer],
                sizeof(struct analog_hardware)),

            hd->seconds++, /* increment total seconds */
        }
    }

    /* notice("loop"), */

    ftask_periodic_timer_block(), /* wait for the timer */

}
/* shouldn't get here */
}

```

D.1.13 octagon_init.c

```

/* octagon_init c */

#include "fcard h"
#include "octagon_io h"

/* setup octagon io cards */
void octagon_init()
{
    /* card0 set port B to all outputs */          /* 5710 */
    /* out_8(CARDO+CR_OFFSET, 0x80), */

    /* card0 do a dummy read */
    (void)in_8(CARDO+HIGH8),
    (void)in_8(CARDO+LOW4),
}

```

```

/* card0 setup digital io mode */
out_8(CARD0+CR_OFFSET, DMODE_BOTH_IN),

/* card1 set port B to all outputs */          /* 5710-1 */
/* out_8(CARD1+CR_OFFSET, 0x80), */

/* card1 do a dummy read */
(void)in_8(CARD1+HIGH8),
(void)in_8(CARD1+LOW4),

/* card1 setup digital io mode */
out_8(CARD1+CR_OFFSET, DMODE_BOTH_OUT),

/* turn off the digital outputs */
out_8(CARD1+DPORTA, ~0),
out_8(CARD1+DPORTC, ~0),

/* card2 no initialization necessary */        /* 5750 */
}

```

D.1.14 octagon_io.c

```

/* octagon_io c */

#define __INLINE_FUNCTIONS__ 1
#include <stdio h>

/* note void *memcpy(void *dest, const void *src, size_t n) */

#include <string h>          /* memcpy */
#include <math h>           /* sin */
#include "fcard h"         /* shared mem pointer definition */
#include "octagon_io h"    /* octagon hardware defines */
#include "octagon_io_map h" /* map channels/bits to named measurements */

#ifndef PI
#define PI 3.14159265358979323846
#endif

/* calibration macros - requires ansi cpp */
#define CALIBRATE(what)      private_analog in what = \
                             private_cal what##Sen * what##Channel + private_cal what##Off

#define DECALIBRATE(what)    what##Channel = \
                             (private_analog out what - private_cal what##Off) / \
                             private_cal what##Sen

#define DECALIBRATESPEED0(what) what##Channel = \
                             private_cal what##Amplitude * sin (private_analog out what * PI/92.5 \
                             + private_cal what##PhaseAngle)

static unsigned loop_counter,

/* there are a few distinct steps to octagon_io

```

```

*
* 1 get integer input data from hardware
* 2 convert input to floating point voltage
* 3 convert float voltage to calibrated values
* 4 share the calibrated inputs
*
* 5 get requested calibrated output values from shared mem
* 6 convert float calibrated values to voltages
* 7 convert voltages to integer values
* 8 output integer values
*
* 9 monkey with digital stuff
*/

void octagon_10()
{
    struct io5710Integer    icard0,
    struct io5710Integer    icard1,
    struct o5750Integer     icard2,

    struct io5710Volts     vcard0,
    struct io5710Volts     vcard1,
    struct o5750Volts      vcard2,

    struct digital_hardware private_digital,    /* named values */
    struct analog_hardware private_analog,     /* named values */
    struct calibration_values private_cal,      /* named values */

    unsigned d, d1, d2,    /* temp vars */
    int oldpointer,

    /* circulate active pointer every 1 seconds */
    d = hd->fcard_rv fcard_hardware_timer_hz,
    if (!(++loop_counter % d))
    {
        oldpointer = hd->active,
        /* if at bottom, loop back to the top */
        if(hd->active == (SECONDS_TO_STORE-1))
            hd->active = 0,
        else
            hd->active++,

        /* ok, we moved the pointer to active data,
        * copy the old pointer's data to prevent
        * data loss (I hope this works)
        */
        memcpy(&hd->cv[hd->active], &hd->cv[oldpointer],
            sizeof(struct calculated_values)),

        memcpy(&hd->digital[hd->active], &hd->digital[oldpointer],
            sizeof(struct digital_hardware)),

        memcpy(&hd->analog[hd->active], &hd->analog[oldpointer],
            sizeof(struct analog_hardware)),

        /* notice("moved pointer"), */
        hd->seconds++, /* increment total seconds in operation */
    }
}

```

```

/* get a private copy of the calibration values */
memcpy(&private_cal, &hd->cal, sizeof(struct calibration_values)),

/***** 1 us *****/

/***** begin analog input *****/

/* 1 do the analog inputs - differential mode */
init_convert(CARD0, 1), 1card0 AnalogIn[1] = getAnalog(CARD0),
/* init_convert(CARD0, 3), 1card0 AnalogIn[3] = getAnalog(CARD0), */
init_convert(CARD0, 5), 1card0 AnalogIn[5] = getAnalog(CARD0),
init_convert(CARD0, 7), 1card0 AnalogIn[7] = getAnalog(CARD0),
init_convert(CARD0, 9), 1card0 AnalogIn[9] = getAnalog(CARD0),
init_convert(CARD0, 11), 1card0 AnalogIn[11] = getAnalog(CARD0),
/* init_convert(CARD0, 13), 1card0 AnalogIn[13] = getAnalog(CARD0), */
init_convert(CARD0, 15), 1card0 AnalogIn[15] = getAnalog(CARD0), */

init_convert(CARD1, 1), 1card1 AnalogIn[1] = getAnalog(CARD1),
init_convert(CARD1, 3), 1card1 AnalogIn[3] = getAnalog(CARD1),
init_convert(CARD1, 5), 1card1 AnalogIn[5] = getAnalog(CARD1),
init_convert(CARD1, 7), 1card1 AnalogIn[7] = getAnalog(CARD1),
init_convert(CARD1, 9), 1card1 AnalogIn[9] = getAnalog(CARD1),
init_convert(CARD1, 11), 1card1 AnalogIn[11] = getAnalog(CARD1),
init_convert(CARD1, 13), 1card1 AnalogIn[13] = getAnalog(CARD1),
init_convert(CARD1, 15), 1card1 AnalogIn[15] = getAnalog(CARD1),

/***** 16*30us, total = 481 us *****/

/* 2 convert input integers to voltages */
vcard0 AnalogIn[1] = INT2VOLTS(1card0 AnalogIn[1] ),
/* vcard0 AnalogIn[3] = INT2VOLTS(1card0 AnalogIn[3] ), */
vcard0 AnalogIn[5] = INT2VOLTS(1card0 AnalogIn[5] ),
vcard0 AnalogIn[7] = INT2VOLTS(1card0 AnalogIn[7] ),
vcard0 AnalogIn[9] = INT2VOLTS(1card0 AnalogIn[9] ),
vcard0 AnalogIn[11] = INT2VOLTS(1card0 AnalogIn[11]),
/* vcard0 AnalogIn[13] = INT2VOLTS(1card0 AnalogIn[13]), */
vcard0 AnalogIn[15] = INT2VOLTS(1card0 AnalogIn[15]), */

vcard1 AnalogIn[1] = INT2VOLTS(1card1 AnalogIn[1] ),
vcard1 AnalogIn[3] = INT2VOLTS(1card1 AnalogIn[3] ),
vcard1 AnalogIn[5] = INT2VOLTS(1card1 AnalogIn[5] ),
vcard1 AnalogIn[7] = INT2VOLTS(1card1 AnalogIn[7] ),
vcard1 AnalogIn[9] = INT2VOLTS(1card1 AnalogIn[9] ),
vcard1 AnalogIn[11] = INT2VOLTS(1card1 AnalogIn[11]),
vcard1 AnalogIn[13] = INT2VOLTS(1card1 AnalogIn[13]),
vcard1 AnalogIn[15] = INT2VOLTS(1card1 AnalogIn[15]),

/***** 1us, total = 482us *****/

/* 3 convert voltages to engineering units */
/* this is where the io mapping happens y = mx + b*/

CALIBRATE(1EmMotorSpeed),
CALIBRATE(1GenMotorSpeed),
CALIBRATE(1IceEngineSpeed),
CALIBRATE(1AccelPedalLevel),
CALIBRATE(1BrakePedalLevel),
CALIBRATE(1ActualEmTorque),
CALIBRATE(1FuelPressure),

```

```

/* special piecewise */
private_analog in.iEmCurrent =
    (iEmCurrentChannel < 2 5F) ? /* 2 5v == zero amps */
    (private_cal iEmCurrentNegSen * iEmCurrentChannel
     + private_cal iEmCurrentNegOff)
    (private_cal iEmCurrentPosSen * iEmCurrentChannel
     + private_cal iEmCurrentPosOff),

CALIBRATE(iGenCurrent),
/* CALIBRATE(iBattPackTemp), */
CALIBRATE(iBattPackVoltage),
CALIBRATE(iBattPackCurrent),
CALIBRATE(iEmRotorTemp),
CALIBRATE(iEmInvTemp),
CALIBRATE(iTpsFeedback),

/***** 1us, total = 483us *****/

/* 4 share input data */
memcpy(&hd->analog[hd->active] in, &private_analog in,
       sizeof(struct input_analog_hardware)),

/***** 1us, total = 484us *****/

/***** end of analog input *****/

/***** begin analog output *****/

/* 5 get calibrated requested output values */
memcpy(&private_analog out, &hd->analog[hd->active] out,
       sizeof(struct output_analog_hardware)),

/***** 1us, total = 485us *****/

/* 6 convert float calibrated values to voltages */
/* note 10 mapping used here x = (y-b)/m */
DECALIBRATE(oGenSpeedReq),
DECALIBRATE(oGenRegenLimit),
DECALIBRATE(oEmAccelReq),
DECALIBRATE(oEmBrakeReq),
DECALIBRATE(oIceThrottlePos),

DECALIBRATESPEED0(oDisSpeedo0),
DECALIBRATESPEED0(oDisSpeedo1),

DECALIBRATE(oPwrToDigRacks), /* really just on/off */

/***** 1us, total = 486us *****/

/* 7 convert voltages to integer values */
/* icard0 AnalogOut[0] = VOLTS2INT(vcard0.AnalogOut[0]), */
/* icard0 AnalogOut[1] = VOLTS2INT(vcard0.AnalogOut[1]), */
/* icard1 AnalogOut[0] = VOLTS2INT(vcard1.AnalogOut[0]), */
/* icard1 AnalogOut[1] = VOLTS2INT(vcard1.AnalogOut[1]), */

icard2 AnalogOut[0] = HALFVOLTS2INT(vcard2.AnalogOut[0]),
icard2 AnalogOut[1] = HALFVOLTS2INT(vcard2.AnalogOut[1]),
icard2 AnalogOut[2] = HALFVOLTS2INT(vcard2.AnalogOut[2]),
/* icard2 AnalogOut[3] = HALFVOLTS2INT(vcard2.AnalogOut[3]), */

icard2 AnalogOut[4] = FULLVOLTS2INT(vcard2.AnalogOut[4]),

```



```

icard2 AnalogOut[5] = FULLVOLTS2INT(vcard2 AnalogOut[5]),
/* icard2 AnalogOut[6] = FULLVOLTS2INT(vcard2 AnalogOut[6]), */
icard2 AnalogOut[7] = FULLVOLTS2INT(vcard2 AnalogOut[7]),

/***** 1us, total = 487us *****/

/* 8 output integer values */
/* 5710 stuff - one channel at a time */
/* OUTPUTS710(CARD0, DAC_CH_ZERO, icard0 AnalogOut[0]), */
/* OUTPUTS710(CARD0, DAC_CH_ONE , icard0 AnalogOut[1]), */
/* OUTPUTS710(CARD1, DAC_CH_ZERO, icard1 AnalogOut[0]), */
/* OUTPUTS710(CARD1, DAC_CH_ONE , icard1 AnalogOut[1]),

/* 5750 stuff */
/* channels 0 and 1 output in one out call */
d = 0,

/* unsigned "negative" numbers will wrap around
 * to somewhere between 2^31 and (2^32)-1
 */
d1 = (icard2 AnalogOut[0] > (1U<<31)) ? 0
      (icard2 AnalogOut[0] > 4095U) ? 4095U
      icard2 AnalogOut[0],

d2 = (icard2 AnalogOut[1] > (1U<<31)) ? 0
      (icard2 AnalogOut[1] > 4095U) ? 4095U
      icard2 AnalogOut[1],

d = d1 | (d2 << 16),
/* now bits should be aligned correctly */
out_32(CARD2+ZERO_LEAST, d),

/* channel 2 output as a single because 3 not used */
d = (icard2 AnalogOut[2] > (1U<<31)) ? 0
      (icard2 AnalogOut[2] > 4095U) ? 4095U
      icard2 AnalogOut[2],
out_16(CARD2+TWO_LEAST, d),

/* channels 4 and 5 output in one out call */
d = 0,

d1 = (icard2 AnalogOut[4] > (1U<<31)) ? 0
      (icard2 AnalogOut[4] > 4095U) ? 4095U
      icard2 AnalogOut[4],

d2 = (icard2 AnalogOut[5] > (1U<<31)) ? 0
      (icard2 AnalogOut[5] > 4095U) ? 4095U
      icard2 AnalogOut[5],

d = d1 | (d2 << 16),
out_32(CARD2+FOUR_LEAST, d),

/* channel 7 output as a single because 6 not used */
d = (icard2 AnalogOut[7] > (1U<<31)) ? 0
      (icard2 AnalogOut[7] > 4095U) ? 4095U
      icard2 AnalogOut[7],

out_16(CARD2+SEVEN_LEAST, d),

/***** end analog outputs *****/
/***** conservative 10*30us = 300us, total = 787us *****/

```

```

/* do digital inputs */
/***** port C inputs *****/
d1 = in_8(CARDO+DPORTC),
d1 = ~d1, /* bit "not" for reverse logic optoisolators */
icard0 DigitalInC[0] = (d1 & 0x01) ? 1 0,
icard0 DigitalInC[1] = (d1 & 0x02) ? 1 0,
icard0 DigitalInC[2] = (d1 & 0x04) ? 1 0,
icard0 DigitalInC[3] = (d1 & 0x08) ? 1 0,
icard0 DigitalInC[4] = (d1 & 0x10) ? 1 0,
icard0 DigitalInC[5] = (d1 & 0x20) ? 1 0,
icard0 DigitalInC[6] = (d1 & 0x40) ? 1 0,
icard0 DigitalInC[7] = (d1 & 0x80) ? 1 0,

d2 = in_8(CARDO+DPORTA),
d2 = ~d2,
icard0 DigitalInA[0] = (d2 & 0x01) ? 1 0,
icard0 DigitalInA[1] = (d2 & 0x02) ? 1 0,
icard0 DigitalInA[2] = (d2 & 0x04) ? 1 0,
icard0 DigitalInA[3] = (d2 & 0x08) ? 1 0,
icard0 DigitalInA[4] = (d2 & 0x10) ? 1 0,
icard0 DigitalInA[5] = (d2 & 0x20) ? 1 0,
icard0 DigitalInA[6] = (d2 & 0x40) ? 1 0,
icard0 DigitalInA[7] = (d2 & 0x80) ? 1 0,

/* map the inputs */
d = 0,
d = d1 & 0xF0, /* put last four bits of port C into d */

private_digital in iPark = (d == iParkBitPattern) ? 1 0,
private_digital in iReverse = (d == iReverseBitPattern) ? 1 0,
private_digital in iNeutral = (d == iNeutralBitPattern) ? 1 0,
private_digital in iDriveEcon = (d == iDriveEconBitPattern) ? 1 0,
private_digital in iDriveSport = (d == iDriveSportBitPattern) ? 1 0,
private_digital in iZEV = (d == iZEVBitPattern) ? 1 0,

private_digital in iHvac = iHvacBit,
private_digital in iEmTempWarn = iEmTempWarnBit,
private_digital in iEmControllerReady = iEmControllerReadyBit,
private_digital in iEmFaultIndicator = iEmFaultIndicatorBit,
private_digital in iEmOvertempIndicator = iEmOvertempIndicatorBit,
private_digital in iGenTempWarn = iGenTempWarnBit,
private_digital in iGenControllerReady = iGenControllerReadyBit,
private_digital in iGenFaultIndicator = iGenFaultIndicatorBit,
private_digital in iGenDirectionIndicator = iGenDirectionIndicatorBit,
private_digital in iIceFaultIndicator = iIceFaultIndicatorBit,

/* share the inputs */
memcpy(&hd->digital[hd->active] in, &private_digital in,
sizeof(struct input_digital_hardware)),

/* get the requested outputs */
memcpy(&private_digital out, &hd->digital[hd->active] out,
sizeof(struct output_digital_hardware)),

/* map the outputs */
oIceStarterBit = private_digital out oIceStarter,
oTecEnableBit = private_digital out oTecEnable,
oEmEnableBit = private_digital out oEmEnable,
oEmDirectionBit = private_digital out oEmDirection,
oGenEnableBit = private_digital out oGenEnable,

```

```
oPwrSteeringEnableBit = private_digital out oPwrSteeringEnable,
oThrottlePwrCycleBit = private_digital out oThrottlePwrCycle,
oSmartChargerEnableBit = private_digital out oSmartChargerEnable,

/* write the outputs */
d = 0,
if (icard1 DigitalOutA[0]) d |=0x01,
if (icard1 DigitalOutA[1]) d |=0x02,
if (icard1 DigitalOutA[2]) d |=0x04,
if (icard1 DigitalOutA[3]) d |=0x08,
if (icard1 DigitalOutA[4]) d |=0x10,
if (icard1 DigitalOutA[5]) d |=0x20,
if (icard1 DigitalOutA[6]) d |=0x40,
if (icard1 DigitalOutA[7]) d |=0x80,

d = ~d, /* bit not for rev logic optoisolators */
out_8(CARD1+DPORTA, d),

/***** digital in/out 3*5us, total = 802us *****/
/* conservatively guess that loop completes in 1ms */

return,
}
```

D.2 fcar: Main Control Program

D.2.1 Makefile

```

CC           =      cc
DEFINES      =
INCLUDES     =      -I /libfclient -I /libftask
QUIET       =      -Q -wx
OPTS        =      -Orailnextm -4r -fp3 -fp187
DEBUG       =      #-g
CFLAGS      =      $(QUIET) $(DEBUG) $(OPTS) $(DEFINES) $(INCLUDES)
LIBS        =      -L /libfclient -L /libftask
LIBS        +=     -lfclient -lftask
LDFLAGS     =      $(QUIET) $(LIBS)
FILES       =      fcar c ice_ctrl c mode_select c modes c motor_eqn c
FILES       +=     soc c misc c
FILES       +=     throttle_ctrl c
OBJS        =      $(FILES) c= o)
OUT         =      fcar
DEPEND      =      makedepend
BINDIR      =      /opt/fcar/bin/

all         $(OUT)

$(OUT)      $(OBJS)
            $(CC) $(LDFLAGS) $^ -o $@

dep
            $(DEPEND) -- $(CFLAGS) -D__QNX__ -- $(FILES)

install     $(OUT)
            cp -f $(OUT) $(BINDIR)

clean
            $(RM) $(OUT) $(OBJS) * err

```

D.2.2 fcar.h

```

/* fcar h */

#ifndef FCAR_H
#define FCAR_H

#include <fclient h> /* shared memory structure definition */
#include <ftask h> /* process control */

/* defines */
#define _POSIX_C_SOURCE 199309L

/* this is put here because it's a little counterintuitive and needs
 * to be consistent across different source files
 */
#define EMENABLE 0 /* em controller on */
#define EMDISABLE 1 /* em controller off */
#define GENENABLE 1
#define GENDISABLE 0

/* critical hybrid parameters */
// #define FULLGENREGEN 0 85F /* normal */
#define FULLGENREGEN 0 65F

```

```

#define          THROTTLE_MAX          0 28F  /* normal */

/* main loop execution speed */
#define          FCAR_TIMER_HZ          100  /* split 100 Hz clock */

/* process scheduling */
#define FCAR_SCHED_POLICY                SCHED_FIFO
#define FCAR_MANAGER_PRIO                24
#define FCAR_MODE_SEL_PRIO                23
#define FCAR_ICE_CTRL_PRIO                21
#define FCAR_THROTTLE_CTRL_PRIO          19
#define FCAR_SOC_CALC_PRIO                18

/* for signals */
#define TriggerICE()                      if (IceState != DEAD) \
                                         ftask_trigger(ice_task)

#define TriggerThrottleUp()                ftask_trigger(throttle_task)

/* global variables */
extern volatile struct shared_hw_data *s, /* points to fcard shared mem */
extern ftask *ice_task,                  /* ice control pid */
extern ftask *throttle_task,             /* throttle ramper upper */
extern ftask *mode_sel_task,             /* real worker */
extern ftask *soc_calc_task,             /* slowly calculates SOC */

/* function prototypes */
void exit_cleanly(int ignored),
void ice_control(void *ignored),
void mode_selector(void *ignored),
void mode_selector_quit(int ignored),
void soc_calculator(void *unused),
void read_shifter(void),
void do_zev(void),
void do_econ(void),
void do_sport(void),
void do_reverse(void),
void do_park(void),
void do_neutral(void),

void zev_motor(void),
void reverse_motor(void),
void hybrid_motor(void),

void do_misc(void),

void throttle_control(void *ignored),

float hybrid_em_accel(void),
float zev_em_accel(),
float hybrid_gen_speed(void),
float hi_regime_gen_speed_req(void),
float lo_regime_gen_speed_req(void),

float hybrid_throttle(void),
float em_brake(void),

#endif /* FCAR_H */

```

D.2.3 fcar.c

```

/* fcar c */

#include <stdlib h>          /* exit, calloc */
#include "fcar h"

/* real definition of shared mem pointer */
volatile struct shared_hw_data *s,

/* real definition of global tasks */
ftask *ice_task, *throttle_task, *mode_sel_task, *soc_calc_task, *ret,

int main(int argc, char *argv[])
{
    notice("fcar started"),

    /* open shared memory */
    if ((s = fclient_open_shm(0_RDWR)) == NULL)
        die("fclient_open_shm(0_RDWR)"),

    /* allocate memory for task structs */
    ret = (ftask *)calloc(1, sizeof(ftask)),
    ice_task = (ftask *)calloc(1, sizeof(ftask)),
    throttle_task = (ftask *)calloc(1, sizeof(ftask)),
    mode_sel_task = (ftask *)calloc(1, sizeof(ftask)),
    soc_calc_task = (ftask *)calloc(1, sizeof(ftask)),

    if(!ice_task || !throttle_task || !mode_sel_task ||
        !soc_calc_task || !ret)
        die("alloc for task structs failed"),

    /* indicate that the control program is running */
    s->cv[s->active] running = 1,

    /* setup our priority/scheduling scheme */
    ftask_sched_adjust_self(FCAR_SCHED_POLICY, FCAR_MANAGER_PRIO),

    /* setup for graceful exit */
    ftask_register_cleanup_self(exit_cleanly),

    /* the order of the following task creation is important,
     * a task will not be able to trigger another task that
     * was created after it */

    /* start task to handle throttle up */
    ftask_init(throttle_task,
               FCAR_SCHED_POLICY,      /* scheduling policy */
               FCAR_THROTTLE_CTRL_PRIO, /* scheduling priority */
               throttle_control,       /* start func */
               NULL,                   /* start func arg */
               NULL,                   /* cleanup func */
               1,                       /* allow trigger */
               0,                       /* allow per timer */
               0),                     /* per timer hz */
    ftask_create(throttle_task),

    /* start a task to handle asynchronous
     * engine starting/stopping */
    ftask_init(ice_task,
               FCAR_SCHED_POLICY,      /* scheduling policy */

```

```

        FCAR_ICE_CTRL_PRIO,      /* priority */
        ice_control,           /* start func */
        NULL,                  /* start func arg */
        NULL,                  /* cleanup func */
        1,                     /* allow trigger */
        0,                     /* allow per timer */
        0),                   /* per timer hz */
    ftask_create(ice_task),

    /* start a task to do real work based on mode */
    ftask_init(mode_sel_task,
        FCAR_SCHED_POLICY,     /* scheduling policy */
        FCAR_MODE_SEL_PRIO,    /* priority */
        mode_selector,         /* start func */
        NULL,                  /* start func arg */
        mode_selector_quit,    /* cleanup func */
        0,                     /* allow trigger */
        1,                     /* allow per timer */
        FCAR_TIMER_HZ),        /* per timer hz */
    ftask_create(mode_sel_task),

    /* finally create a task to calculate state of charge */
    ftask_init(soc_calc_task,
        FCAR_SCHED_POLICY,     /* scheduling policy */
        FCAR_SOC_CALC_PRIO,    /* priority */
        soc_calculator,       /* start func */
        NULL,                  /* start func arg */
        NULL,                  /* cleanup func */
        0,                     /* allow trigger */
        0,                     /* allow per timer */
        0),                   /* per timer hz */
    ftask_create(soc_calc_task),

    /* just wait */
    while (s)
    {
        ftask_wait_on_tasks(ret),

        if (ftask_same(ret, ice_task))
            notice("ice controller died?"),

        if (ftask_same(ret, mode_sel_task))
            notice("mode selector died?"),

        if (ftask_same(ret, throttle_task))
            notice("throttle controller died?"),

        if (ftask_same(ret, soc_calc_task))
            notice("SOC calculator died?"),

    }

    /* shouldn't get here */
    exit_cleanly(0),

    return 0,
}

void exit_cleanly(int ignored)
{

```

```

/* indicate that we've stopped */
s->cv[s->active] running = 0,

/* stop all know tasks */
if (ftask_delete(ice_task) != 0)
    warn("can't delete ice controller task"),

if (ftask_delete(mode_sel_task) != 0)
    warn("can't delete mode selector task"),

if (ftask_delete(throttle_task) != 0)
    warn("can't delete throttle controller task"),

if (ftask_delete(soc_calc_task) != 0)
    warn("can't delete SOC calculator task"),

/* not needed, */
free(ice_task),
free(mode_sel_task),
free(throttle_task),
free(soc_calc_task),
free(ret),

/* done */
exit(EXIT_SUCCESS),
}

```

D.2.4 mode_select.c

```

/* mode_select c */

#include <stdlib.h>          /* exit */
#include "fcar.h"

/* 10 macros used in this file */
#define ZEV                s->digital[s->active] in iZEV
#define DriveEcon          s->digital[s->active] in iDriveEcon
#define DriveSport         s->digital[s->active] in iDriveSport
#define Reverse            s->digital[s->active] in iReverse
#define Park                s->digital[s->active] in iPark
#define Neutral            s->digital[s->active] in iNeutral
#define mode                s->cv[s->active] mode
#define PwrToDigRacks      s->analog[s->active] out oPwrToDigRacks
#define hybrid_regime      s->cv[s->active] hybrid_regime

void mode_selector_quit(int ignored)
{
    notice("mode selector quitting"),
    exit(EXIT_SUCCESS),
}

void mode_selector(void *unused)
{
    notice("mode selector started"),

    /* turn on power to digital racks */
    PwrToDigRacks = ON,

```



```
/* set initial regime */
hybrid_regime = LO_REGIME,

/* wait for periodic timer */
while(s)
{
    /* read shifter */
    read_shifter(),

    /* do misc stuff */
    do_misc(),

    ftask_periodic_timer_block(),
}
/* shouldn't get here */
exit(EXIT_FAILURE),
}

void read_shifter()
{
    if (ZEV)
    {
        mode = ZEV_MODE,
        do_zev(),
    }
    else if (DriveEcon)
    {
        mode = ECON_MODE,
        do_econ(),
    }
    else if (DriveSport)
    {
        mode = SPORT_MODE,
        do_sport(),
    }
    else if (Reverse)
    {
        mode = REVERSE_MODE,
        do_reverse(),
    }
    else if (Park)
    {
        mode = PARK_MODE,
        do_park(),
    }
    else if (Neutral)
    {
        mode = NEUTRAL_MODE,
        do_neutral(),
    }
    else /* default */
    {
        /* this really doesn't work as intended, the
         * shifter switch pattern for park is the same
         * as if it were completely disconnected */
        mode = NEUTRAL_MODE,
        /* notice("can't read shifter"), */
        do_neutral(),
    }
}
```

```

    return,
}

```

D.2.5 modes.c

```

/* modes c */

#include "fcar h"

/* 10 macros used in this file */
#define EmAccelReq      s->analog[s->active] out oEmAccelReq
#define EmEnable       s->digital[s->active] out oEmEnable
#define EmSpeed        s->analog[s->active] in iEmMotorSpeed
#define EmDirection    s->digital[s->active] out oEmDirection
#define mode           s->cv[s->active] mode
#define IceReqState    s->cv[s->active] IceReqState
#define IceState       s->cv[s->active] IceState
#define GenEnable      s->digital[s->active] out oGenEnable
#define GenRegen       s->analog[s->active] out oGenRegenLimit
#define GenSpeedReq    s->analog[s->active] out oGenSpeedReq
#define EmBrakeReq     s->analog[s->active] out oEmBrakeReq
#define Hvac           s->digital[s->active] in iHvac
#define SOCcat         s->cv[s->active] SOCcat
#define SOC            s->cv[s->active] SOC
#define throttle       s->analog[s->active] out oIceThrottlePos
#define AccelPedal     s->analog[s->active] in iAccelPedalLevel
#define BrakePedal     s->analog[s->active] in iBrakePedalLevel
#define EmControllerReady s->digital[s->active] in iEmControllerReady
#define GenControllerReady s->digital[s->active] in iGenControllerReady
#define vehicle_speed  s->cv[s->active] vehicle_speed
#define hybrid_regime  s->cv[s->active] hybrid_regime

#define MAX_BACK_TO_FORWARD_SWITCH_SPEED 5 OF /* mph */
#define MAX_FORWARD_TO_BACK_SWITCH_SPEED 5 OF /* mph */

#define FORWARD 0 /* em direction */
#define REVERSE 1

#define EMREADY 1
#define GENREADY 1

#define MAX_PARK_EMBRAKE 1 OF /* normal */

void do_zev() /****** ZEV *****/
{
    /* make sure that traction motor and generator are enabled */
    EmEnable = EMENABLE,
    // GenEnable = GENENABLE,
    GenEnable = GENDISABLE,

    if (EmControllerReady != ON) return, /* note need feedback to work */
    // if (GenControllerReady != ON) return,

    /* insure that the vehicle is nearly
     * standing still or moving forward */
    if ((vehicle_speed < MAX_BACK_TO_FORWARD_SWITCH_SPEED) ||
        (EmDirection == FORWARD) )
    {
        if (EmDirection == REVERSE )

```

```

        EmDirection = FORWARD,

        /* check if this is the real ZEV mode */
        if (mode == ZEV_MODE)
        {
            /* make sure that engine is off */
            IceReqState = OFF,
            if (IceState == ON) TriggerICE(),

        }
        /* don't mess with engine/generator if being
        * called from a hybrid mode in transition */

        /* don't care about state of charge here */

        /* proportion traction motor and generator */
        zev_motor(),
    }
    else
        do_reverse(),
    return,
}

void do_reverse()      /***** REVERSE *****/
{
    /* insure that traction motor is enabled */
    EmEnable = EMENABLE,

    if (EmControllerReady != ON) return,    /* need feedback to work */

    /* make sure that engine is off */
    IceReqState = OFF,
    if (IceState == ON) TriggerICE(),

    /* don't care about state of charge */

    if ( (vehicle_speed < MAX_FORWARD_TO_BACK_SWITCH_SPEED) ||
        (EmDirection == REVERSE) )
    {
        if (EmDirection == FORWARD)
            EmDirection = REVERSE,

        /* set generator torque to zero */
        if (IceState == OFF) GenEnable = GENDISABLE,

        /* proportion traction motor */
        reverse_motor(),
    }
    else
        do_zev(),
    return,
}

void do_econ()        /***** ECON *****/
{
    EmEnable = EMENABLE,
    GenEnable = GENENABLE,

    if (EmControllerReady != ON) return,    /* need feedback to work */
    if (GenControllerReady != ON) return,
}

```

```

/* insure that the vehicle is nearly
 * standing still or moving forward */
if ((vehicle_speed < MAX_BACK_TO_FORWARD_SWITCH_SPEED) ||
    (EmDirection == FORWARD) )
{
    if (EmDirection == REVERSE ) EmDirection = FORWARD,
}
else
    return, /* do nothing */

/* if SOC is high or the A/C switch is off */
if ((Hvac == OFF))
{
    /* try to stop engine */
    IceReqState = OFF,
    if (IceState == ON)
    {
        TriggerICE(),
        do_zev(),
        return,
    }

    if (IceState == WORKING)
    {
        do_zev(),
        return,
    }

    /* if we can get here, engine should be off */

    /* check to see if transition from HEV
     * to ZEV is needed */

    do_zev(),
}
else
/* if SOC is low or A/C switch is on */
if ((Hvac == ON))
{
    /* try to start engine */
    IceReqState = ON,
    if (IceState == OFF)
    {
        TriggerICE(),
        do_zev(), /* fulfill regular driving requests */
        return,
    }
    else if (IceState == WORKING)
    {
        do_zev(),
        return,
    }

    /* engine should be on if we can get here */

    // TriggerThrottleUp(),

```

```

        /* check to see if transition from ZEV
        * to HEV is needed */

        /* do regular proportioning of motor and generator */
        hybrid_motor(),
    }
    return,
}

void do_sport()          /***** SPORT *****/
{
    /* warning stub function to do stationary testing by
    * proportioning gen and ice */

    if (Hvac)          /* try to start engine */
    {
        IceReqState = ON,
        if (IceState == OFF)
        {
            TriggerICE(),
            work_throttle(),
            return,
        }
        else if (IceState == WORKING)
        {
            work_throttle(),
            return,
        }
    }
    else                /* try to stop engine */
    {
        IceReqState = OFF,
        if (IceState == ON)
        {
            TriggerICE(),
            work_throttle(),
            return,
        }

        if (IceState == WORKING)
        {
            work_throttle(),
            return,
        }
    }

    work_throttle(),

    return,
}

void do_neutral()      /***** NEUTRAL *****/
{
    /* give the engine another chance at starting if it failed
    * to start (multiple times) before */
    if (IceState == DEAD)
    {
        notice("reseting dead engine"),
        IceState = OFF, /* reset */
    }
}

```

```

    /* make sure that engine is off */
    IceReqState = OFF,
    if (IceState == ON) TriggerICE(),

    /* set generator torque to zero after the engine stops */
    if (IceState == OFF)
    {
        GenEnable = GENDISABLE,
        GenRegen = 0 OF,
        GenSpeedReq = 0 OF,
    }

    /* set traction motor torque to zero */
    EmEnable = EMDISABLE,
    EmAccelReq = 0 OF,
    EmBrakeReq = 0 OF,

    return,
}

void do_park()          /***** PARK *****/
{
    #if 0
        EmEnable = EMENABLE,

        /* make sure that engine is off */
        IceReqState = OFF,
        if (IceState == ON) TriggerICE(),

        /* set generator torque to zero after the engine stops */
        if (IceState == OFF) GenEnable = GENDISABLE,

        /* set traction motor torque to full */
        EmBrakeReq = MAX_PARK_EMBRAKE,
    #else
        do_neutral(),
    #endif
    return,
}

void work_throttle()
{
    GenEnable = GENENABLE,
    /* GenEnable = GENDISABLE, */
    EmEnable = EMDISABLE,
    /* EmEnable = EMENABLE, */

    /*
    EmDirection = FORWARD,
    EmAccelReq = 0 10,
    */

    /* 1 1 map of accel pedal to engine throttle */
    throttle = AccelPedal,

    /* 1 1 map of brake to gen regen */
    GenRegen = BrakePedal,

    return,
}

```

D.2.6 motor_eqn.c

```

/* motor_eqn c */

#include "fcar h"
#include <math h> /* exp */
#include <signal h>

#ifndef PI
#define PI 3.14159265358979323846
#endif
#define MIN(a,b) (((a) < (b)) ? (a) (b))
#define MAX(a,b) (((a) > (b)) ? (a) (b))
#define RPM2RAD(a) ((a) * PI/30 0)

/* 10 macros */
#define EmSpeed s->analog[s->active] in iEmMotorSpeed
#define EmAccelReq s->analog[s->active] out oEmAccelReq
#define AccelPedal s->analog[s->active] in iAccelPedalLevel
#define EmBrakeReq s->analog[s->active] out oEmBrakeReq
#define BrakePedal s->analog[s->active] in iBrakePedalLevel
#define EngineSpeed s->analog[s->active] in iIceEngineSpeed
#define Tps s->analog[s->active] in iTpsFeedback
#define GenCurrent s->analog[s->active] in iGenCurrent
#define GenRegen s->analog[s->active] out oGenRegenLimit
#define GenSpeedReq s->analog[s->active] out oGenSpeedReq
#define GenSpeed s->analog[s->active] in iGenMotorSpeed
#define Hvac s->digital[s->active] in iHvac
#define vehicle_speed s->cv[s->active] vehicle_speed
#define hybrid_regime s->cv[s->active] hybrid_regime
#define hybrid_transition s->cv[s->active] hybrid_transition
#define Throttle s->analog[s->active] out oIceThrottlePos

/* constants */
#define MOTOR_TORQUE_CPOWER_CUTOFF 2108 OF /* rpm */
#define MOTOR_TORQUE_MAX 240 OF /* N m */

#define KT 0.429F /* N m/amp */
#define LO_REGIME_SPEED_THRESHOLD 60 OF /* mph */
#define HI_REGIME_SPEED_THRESHOLD 45 OF /* mph */

/* N m max combined torque at wheel */
#define VEHICLE_TORQUE_MAX 2400 OF
#define MOTOR_POWER_MAX 53000 OF /* Watts */
#define MOTOR_TO_WHEEL_RATIO 7.588F /* ratio */
#define REV_ACCEL_FACTOR 1 OF /* desensitize */

/* generator speed curve fit constants */
#define A2 -0.933F /* low end curve */
#define A1 17.82F
#define A0 (4044 OF + 800 OF)

#define B2 -0.778F /* high end */
#define B1 2.264F
#define B0 (-560 OF + 800 OF)
/* engine speed curve fit constants */
#define C2 -0.3F /* low end curve */

```

```

#define C1                60 0F
#define C0                1300 0F

#define D2                -0 25F      /* high end */
#define D1                55 0F
#define D0                -180 0F

#define WHEEL_TO_MOTOR_RATIO 0 131787032156F /* 1/7 588 */
#define GEN_TO_WHEEL_RATIO 1 66845878F /* (49*57)/(27*62) */

#define ZERO              0 0F
#define HI_REGIME_ZEV_GEN_SPEED -3000 0F /* rpm */

#define IDLE_THROTTLE     0 0F /* normal */
#define IDLE_THROTTLE_THRESHOLD 0 05F /* normal ~ 5% */
#define IDLE_ENGINE_THRESHOLD 1000 0F /* rpm */

#define ZERO_GEN_SPEED_REQ 0 0F /* rpm */
#define ZERO_GEN_SPEED_THRESHOLD 300 0F /* rpm */

#define EM_BRAKE_THRESHOLD 0 5F /* normal */

#define THROTTLE_MAX      (0 6F) /* normal */
#define INIT_THROTTLE    (0 2F) /* normal */
#define HI_SPD_ADD_THROTTLE (THROTTLE_MAX-INIT_THROTTLE)
#define THROTTLE_CONST_K1 (0 1F)

float hybrid_throttle()
{
    #if 0
        return (HI_SPD_ADD_THROTTLE *
                (1-exp(-vehicle_speed * THROTTLE_CONST_K1))
                + INIT_THROTTLE),
    #else
        if (vehicle_speed > 3 0)
            return (0 28F),
        else
            return (0 0F),
    #endif
}

/* only do regen braking when request is significantly high */
float em_brake()
{
    if (BrakePedal > EM_BRAKE_THRESHOLD)
    {
        return BrakePedal,
    }
    else
        return ZERO,
}

void zev_motor()
{
    /* fulfill accel request */
    EmAccelReq = zev_em_accel(), /* normal */
}

```



```

/* fulfill brake request */
EmBrakeReq = em_brake(),      /* normal */

if ( (vehicle_speed > LO_REGIME_SPEED_THRESHOLD) &&
      (hybrid_regime == LO_REGIME))
{
    hybrid_regime = HI_REGIME,
}

if ( (vehicle_speed < HI_REGIME_SPEED_THRESHOLD) &&
      (hybrid_regime == HI_REGIME))
{
    hybrid_regime = LO_REGIME,
}

if (hybrid_regime == LO_REGIME)
{
    GenSpeedReq = ZERO,
    GenRegen = ZERO,
}

if (hybrid_regime == HI_REGIME)
{
    // GenSpeedReq = HI_REGIME_ZEV_GEN_SPEED,
    // GenRegen = ZERO,
    GenSpeedReq = ZERO,
    GenRegen = ZERO,
}

}

void reverse_motor() /* note that em direction has already been handled */
{
    /* since it is not possible to go very fast in reverse,
     * no checks like in zev are made */

    /* fulfill accel request */
    EmAccelReq = AccelPedal * REV_ACCEL_FACTOR, /* normal */

    /* fulfill brake request */
    EmBrakeReq = BrakePedal, /* normal */

}

void hybrid_motor()
{
    float genspeed,

    /* proportion traction motor */
    EmAccelReq = hybrid_em_accel(), /* normal */
    /* fulfill brake request */
    EmBrakeReq = em_brake(), /* normal */

    /* check for transition from low to high regime */
    if ( (vehicle_speed > LO_REGIME_SPEED_THRESHOLD) &&
          (hybrid_regime == LO_REGIME))
    {
        if (hybrid_transition == UNDEFINED)

```

```

        hybrid_transition = SLOWING_DOWN,

if (hybrid_transition == SLOWING_DOWN)
{
    /* set engine throttle to idle */
    Throttle = IDLE_THROTTLE,
    if (Tps > IDLE_THROTTLE_THRESHOLD)
        /* throttle still in transition */
        return,

    /* throttle is in idle position if we can get here */
    if (EngineSpeed > IDLE_ENGINE_THRESHOLD)
        return, /* wait for engine to slow down */

    /* NOTE. ***** gen speed req needs to
     * happen before engine slow down */

    /* set gen desired speed to zero */
    GenSpeedReq = ZERO_GEN_SPEED_REQ,

    /* find abs(GenSpeed) */
    genspeed = (GenSpeed < 0) ? -GenSpeed   GenSpeed,

    if (genspeed > ZERO_GEN_SPEED_THRESHOLD)
        /* not slowed down yet */
        return,

    /* gen at zero speed and engine at idle at this point */
    hybrid_transition = SPEEDING_UP,
}

if (hybrid_transition == SPEEDING_UP)
{
    /* turn generator around */
    GenSpeedReq = hi_regime_gen_speed_req(),

    /* begin ramp up of engine */
    TriggerThrottleUp(),
}

/* transition done */
hybrid_regime = HI_REGIME,
hybrid_transition = UNDEFINED,
}
else
/* check for transition from high to low regime */
if ( (vehicle_speed < HI_REGIME_SPEED_THRESHOLD) &&
    (hybrid_regime == HI_REGIME))
{
    if (hybrid_transition == UNDEFINED)
        hybrid_transition = SLOWING_DOWN,

    if (hybrid_transition == SLOWING_DOWN)
    {
        /* set engine throttle to idle */
        Throttle = IDLE_THROTTLE,
        if (Tps > IDLE_THROTTLE_THRESHOLD)
            /* throttle still in transition */
            return,

        /* throttle is in idle position if we can get here */

```

```

        if (EngineSpeed > IDLE_ENGINE_THRESHOLD)
            return, /* wait for engine to slow down */

        /* set gen desired speed to zero */
        GenSpeedReq = ZERO_GEN_SPEED_REQ,

        /* find abs(GenSpeed) */
        genspeed = (GenSpeed < 0) ? -GenSpeed  GenSpeed,

        if (genspeed > ZERO_GEN_SPEED_THRESHOLD)
            /* not slowed down yet */
            return,

        /* gen at zero speed and engine at idle at this point */
        hybrid_transition = SPEEDING_UP,
    }

    if (hybrid_transition == SPEEDING_UP)
    {
        /* turn generator around */
        GenSpeedReq = lo_regime_gen_speed_req(),

        /* begin ramp up of engine */
        TriggerThrottleUp(),
    }

    /* transition done */
    hybrid_regime = LO_REGIME,
    hybrid_transition = UNDEFINED,

}
else
{
    /* dictate regular hybrid gen speed & regen */
    GenSpeedReq = hybrid_gen_speed(), /* rpm, with direction */
    GenRegen = FULLGENREGEN,
    Throttle = hybrid_throttle(), /* hack */
}

return,
}

float hybrid_em_accel()
{
    float max_em_torque, /* N m */
    float wheel_torque_req, /* N m */
    float torque_from_gen, /* N m */

    /* find maximum possible torque of motor at current speed */
    if (EmSpeed < MOTOR_CTORQUE_CPOWER_CUTOFF)
        max_em_torque = MOTOR_TORQUE_MAX,
    else
        max_em_torque = MOTOR_POWER_MAX / RPM2RAD(EmSpeed),

    /* find the drivers requested torque */
    wheel_torque_req = AccelPedal * VEHICLE_TORQUE_MAX,

    /* find out what the generator/engine is providing */
    torque_from_gen = GenCurrent * KT * GEN_TO_WHEEL_RATIO,

```

```

    /* proportion traction motor */
    return ((MIN(wheel_torque_req/MOTOR_TO_WHEEL_RATIO,
                MOTOR_TORQUE_MAX) - torque_from_gen)
           / MOTOR_TORQUE_MAX), /* normalized */
}

float zev_em_accel()
{
    float max_em_torque, /* N m */
          wheel_torque_req, /* N m */

    /* find maximum possible torque of motor at current speed */
    if (EmSpeed < MOTOR_CTORQUE_CPOWER_CUTOFF)
        max_em_torque = MOTOR_TORQUE_MAX,
    else
        max_em_torque = MOTOR_POWER_MAX / RPM2RAD(EmSpeed),

    /* fulfill accel request */
    wheel_torque_req = AccelPedal * VEHICLE_TORQUE_MAX,

    return (MIN(wheel_torque_req/MOTOR_TO_WHEEL_RATIO,
                MOTOR_TORQUE_MAX) / MOTOR_TORQUE_MAX), /* normalized */
}

float hybrid_gen_speed()
{
    float gen_speed_req, /* rpm */
          ice_speed, /* rpm */

    /* do normal operation */
    if (hybrid_regime == LO_REGIME)
    {
        /* calculate gen speed request */
        gen_speed_req = A2*vehicle_speed*vehicle_speed +
                       A1*vehicle_speed +
                       A0,

        /* calculate what we think the engine should be doing */
        ice_speed = C2*vehicle_speed*vehicle_speed +
                   C1*vehicle_speed +
                   C0,
    }

    if (hybrid_regime == HI_REGIME)
    {
        /* calculate gen speed request */
        gen_speed_req = B2*vehicle_speed*vehicle_speed +
                       B1*vehicle_speed +
                       B0,

        /* calculate what we think the engine should be doing */
        ice_speed = D2*vehicle_speed*vehicle_speed +
                   D1*vehicle_speed +
                   D0,
    }

    /* proportion generator */
    return gen_speed_req, /* rpm, with direction */
}

```

```

}

float hi_regime_gen_speed_req()
{
    float vspeed, gen_speed_req,

    vspeed = vehicle_speed,

    /* calculate gen speed request */
    gen_speed_req = B2*vspeed*vspeed +
                   B1*vspeed +
                   B0,

    return gen_speed_req,
}

float lo_regime_gen_speed_req()
{
    float vspeed, gen_speed_req,

    vspeed = vehicle_speed;

    /* calculate gen speed request */
    gen_speed_req = A2*vspeed*vspeed +
                   A1*vspeed +
                   A0,

    return gen_speed_req,
}

```

D.2.7 misc.c

```

/* misc c */

#include "fcar h"

#define vehicle_speed          s->cv[s->active] vehicle_speed
#define vehicle_distance      s->cv[s->active] vehicle_distance
#define EmSpeed                s->analog[s->active] in iEmMotorSpeed
#define GenSpeed               s->analog[s->active] in iGenMotorSpeed
#define Tec                    s->digital[s->active] out oTecEnable
#define PwrSteeringEnable      s->digital[s->active] out oPwrSteeringEnable
#define DisSpeedo0             s->analog[s->active] out oDisSpeedo0
#define DisSpeedo1            s->analog[s->active] out oDisSpeedo1
#define EmBrakeReq             s->analog[s->active] out oEmBrakeReq
#define BattPackVoltage        s->analog[s->active] in iBattPackVoltage
#define IceState                s->cv[s->active] IceState
#define ThrottleUp             s->cv[s->active] ThrottleUp
#define throttle                s->analog[s->active] out oIceThrottlePos

#define EM_SPEED_TO_VEHICLE_SPEED  96 5333 /* rpm per mph */
#define PWR_STR_OFF_SPEED          30 0F /* in vehicle mph */
#define PWR_STR_ON_SPEED           25 0F /* in vehicle mph */
#define SECONDS_IN_HOUR            3600 0F /* duh */
#define WAY_TOO_FAST_GEN_SPEED     6400 0F /* rpm */
#define IDLE_SAFETY_THROTTLE       0 0F /* normalized, closed */

/* smart charger turn off parameters */
#define SMART_CHARGE_OFF_VOLTAGE   340 0F /* pack voltage */
#define SMART_CHANGE_OFF_EM_BRAKE  0 20F /* em brake request */

```

```

#define MIN(a,b)      ( ((a) < (b)) ? (a)  (b))
#define MAX(a,b)      ( ((a) > (b)) ? (a)  (b))

static double t_old,

void do_misc()
{
    double delta_t,
    double t,

    t = ftask_gettime(),

    /* figure delta_t */
    if (t_old < 1 0)      /* first loop or it's January 1, 1970  ) */
        delta_t = 0 0,
    else
        delta_t = t - t_old,

    t_old = t,

    /* get vehicle speed in miles per hour */
    vehicle_speed = EmSpeed / EM_SPEED_TO_VEHICLE_SPEED,

    /* calculate distance travelled since boot */
    /* with Euler method discrete integration */
    vehicle_distance += vehicle_speed / SECONDS_IN_HOUR * delta_t,
    /* vehicle_distance = 0 0F, */

    /* control power steering pump */
    if ((PwrSteeringEnable == ON) && (vehicle_speed > PWR_STR_OFF_SPEED))
        PwrSteeringEnable = OFF,

    if ((PwrSteeringEnable == OFF) && (vehicle_speed < PWR_STR_ON_SPEED))
        PwrSteeringEnable = ON,

    /* control big analog speedometer
    * gauge in dash */
    DisSpeedo0 = DisSpeedo1 = vehicle_speed,
#if 0
    /* see if smart charger needs to be on */
    if ( (BattPackVoltage > SMART_CHARGE_OFF_VOLTAGE) ||
        (EmBrakeReq > SMART_CHANGE_OFF_EM_BRAKE) ||
        (IceState == ON) )

        SmartChargerEnable = OFF,      /* run accessories on aux-bat */
    else
        SmartChargerEnable = ON,      /* run dc/dc in parallel to aux-bat */

#endif

#if 0
    /* fix ThrottleUp */
    if (IceState == OFF)
        ThrottleUp = OFF,
#endif

#if 1
    /* cut engine if generator is speeding too fast */

```

```

    if (GenSpeed > WAY_TOO_FAST_GEN_SPEED)
    {
        /* cut the ignition, stop engine */
        Tec = OFF,
        throttle = IDLE_SAFETY_THROTTLE,
        notice("Gen spinning too fast, slowing engine"),
        /* TriggerThrottleUp(), */
    }

#endif
    return,
}

```

D.2.8 ice_ctrl.c

```

/* ice_ctrl c */

#include <stdlib.h>    /* exit */
#include "fcar.h"

/* macros used in this file */
#define mode          s->cv[s->active] mode
#define IceEngineSpeed s->analog[s->active] in iIceEngineSpeed
#define starter      s->digital[s->active] out oIceStarter
#define IceReqState  s->cv[s->active] IceReqState
#define IceState     s->cv[s->active] IceState
#define throttle     s->analog[s->active] out oIceThrottlePos
#define TEC          s->digital[s->active] out oTecEnable
#define ThrottlePwrCycle s->digital[s->active] out oThrottlePwrCycle
#define CRANKSEC     0 25F          /* 25 seconds */
#define STARTTHROTTLE 0 OF          /* normal */
#define MAXCRANKS    8              /* total 2 crank sec */
#define MINSTARTRPM  500 OF        /* rpm */
#define FUELSTARVESEC 2 OF         /* 2 seconds */
#define STOPTHROTTLE 0 OF         /* normal */
#define MINSTOPRPM   500 OF        /* rpm */
#define MAXFAILEDSTARTS 3

/* control engine state asynchronously from rest of control loop */
void ice_control(void *ignored)
{
    int failed_starts = 0,
        internal_state = IceState,

    notice("ice controller ready"),

    while(s) /* while shm is valid */
    {
        int i = 1,

        /* wait around until the engine needs to change state */
        ftask_trigger_block(),

        if (internal_state == DEAD)
        {
            /* been reset externally */
            internal_state = IceState,
            failed_starts = 0,
        }

        /* request to turn on the engine */
    }
}

```

```

if ((IceReqState == ON) && (IceState == OFF))
{
    notice("got engine start req"),

    /* start engine */
    IceState = WORKING,
    ThrottlePwrCycle = ON,
    throttle = STARTTHROTTLE,
    TEC = ON,

    /* try cranking the engine */
    while ( (1 <= MAXCRANKS) &&
            (IceEngineSpeed <= MINSTARTRPM) &&
            (IceReqState == ON) )
    {
        /* crank for a while */
        starter = ON,
        ftask_delay(CRANKSEC),
        1++,
    }

    //if (1)
    if (IceEngineSpeed >= MINSTARTRPM) /* success */
    {
        starter = OFF,
        notice("engine started ok"),

        failed_starts = 0,

        /* throttle up in completely
         * automatic fashion in HEV mode
         */

        if (mode == ECON_MODE)
        {
            /* wait a bit to let engine
             * stabilize */
            ftask_delay(0 5F),
            //TriggerThrottleUp(),
            //throttle = THROTTLE_MAX,
        }
        IceState = ON,
    }
    else
    {
        /* failed to start engine */
        starter = OFF,
        TEC = OFF,
        ThrottlePwrCycle = OFF,
        IceState = OFF,
        notice("failed to start engine"),
        /* note should probably keep a count of
         * failed engine start attempts and not
         * even try if not successful after
         * X attempts, update here it is */
        failed_starts++,

        if (failed_starts == MAXFAILEDSTARTS)
        {
            IceState = DEAD,

```



```

                                internal_state = DEAD,
                                }
                                }
}
else
/* request to turn off engine */
if ((IceReqState == OFF) && (IceState == ON))
{
    notice("got engine stop req"),

    /* stop engine */
    IceState = WORKING,
    throttle = STOPTHROTTLE,
    TEC = OFF,      /* starve the fuel */
    ThrottlePwrCycle = OFF,

    /* wait a bit to let things settle */
    ftask_delay(FUELSTARVESEC),

    if (IceEngineSpeed <= MINSTOPRPM)
    {
        /* it worked */
        IceState = OFF,
        notice("engine stopped ok"),
    }
    else
    {
        /* can't stop the engine */
        IceState = ON,
        notice("can't stop runaway engine"),
    }
}
else
/* if reqstate == actual state, do nothing */
if ( ((IceReqState == ON) && (IceState == ON)) ||
      ((IceReqState == OFF) && (IceState == OFF)) )
{
    notice("ouch, race condition? req==actual"),
}

}
/* should never get here */
notice("this should never get here"),
exit(EXIT_FAILURE),
}

```

D.2.9 soc.c

```

/* soc c */
#include <string h>      /* memset */
#include "fcar h"

/* macros used in this file */
#define VVHval          0 80
#define VHval           0 75
#define Hval            0 65      /* normalized */
#define Lval            0 35
#define VLval           0 25
#define VVlval          0 20
#define amps             (-(s->cv[s->active] kwh_amps)) /* reverse sign */

```

```

#define volts          s->cv[s->active] kwh_volts
#define SOC            s->cv[s->active] SOC
#define SOCcat        s->cv[s->active] SOCcat

void soc_calculator(void *unused)
{
    int i, n, loops = 0,
        SOCcat_old = -1, /* for first time */
        SOC_old = -1 0F,
        vhi, vmed, vlo,
        blend[14], sum, soc,

    /* threshold values used in "linear" interpolation */
    const double hi_thresh = 0 85,
        med1_thresh = 0 40,
        med2_thresh = 0 50,
        low1_thresh = 0 25,
        low2_thresh = 0 15,

    /* these are cubic equations */
    const double chi[] = { -5 8633e-05, /* aka coef100 */
        1 2505e-02,
        -9 8923e-01,
        3 4203e+02},

    const double cmed[] = { -4 7965e-05, /* aka coef60 */
        1 1156e-02,
        -9 8899e-01,
        3 3197e+02},

    const double clo[] = { -3 5965e-05, /* completely made up */
        0 8156e-02,
        -9 8899e-01,
        3 1532e+02},

    memset((void *)blend, 0, sizeof(blend)),
    n = (sizeof(blend) / sizeof(blend[0])), /* number of elements of blend */

    notice("soc calculator running"),

    while(s)
    {
        /* find hi, med, lo curves */
        vhi = chi[0]*amps*amps*amps +
            chi[1]*amps*amps +
            chi[2]*amps +
            chi[3],

        vmed = cmed[0]*amps*amps*amps +
            cmed[1]*amps*amps +
            cmed[2]*amps +
            cmed[3],

        vlo = clo[0]*amps*amps*amps +
            clo[1]*amps*amps +
            clo[2]*amps +
            clo[3],

        if (volts > vhi)
            soc = 1 0,
        else if ((volts <= vhi) && (volts > vmed))

```

```

        /* interpolate between h1 and med */
        soc = ((volts-vmed)/(vhi-vmed)) *
              (h1_thresh -medi_thresh) +
              medi_thresh,
    else if ((volts <=vmed) && (volts > vlo))
        /* interpolate between med and lo */
        soc = ((volts-vlo)/(vmed-vlo)) *
              (med2_thresh - low1_thresh) +
              low1_thresh,
    else
        /* assign to low */
        soc = low2_thresh,

/* could just let SOC=soc here, but averaging over
 * a few points seems to help */
sum = 0 0,
for (i=0,i<=(n-2),i++)
{
    blend[i] = blend[i+1], /* shift past values down */
    sum += blend[i],      /* keep a running total */
}
/* put current at end */
blend[n-1] = soc,
sum += soc,

if (loops >= n)
    /* use averaged */
    SOC = sum / (double)n,
else
    /* use immediate */
    SOC = soc,

/* categorize SOC */
/* if previous values are unknown, then assume SOC is
 * categorized to the next highest bin */
if ((SOCcat_old == -1) || (SOC_old == -1 0))
{
    SOCcat = VWL,
    if (SOC > VWLval) SOCcat = VL,
    if (SOC > VLval) SOCcat = L,
    if (SOC > Lval) SOCcat = H,
    if (SOC > Hval) SOCcat = VH,
    if (SOC > VHval) SOCcat = VVH,
    SOC_old = SOC,
    SOCcat_old = SOCcat,
}

/* now that the previous value is known, categorize the SOC */
if ( ((SOC_old > VWLval) && (SOC < VWLval)) ||
      ((SOC_old < VWLval) && (SOC > VWLval)) )
    SOCcat = VWL,
if ( ((SOC_old > VLval) && (SOC < VLval)) ||
      ((SOC_old < VLval) && (SOC > VLval)) )
    SOCcat = VL,
if ( ((SOC_old > Lval) && (SOC < Lval)) ||
      ((SOC_old < Lval) && (SOC > Lval)) )
    SOCcat = L,
if ( ((SOC_old > Hval) && (SOC < Hval)) ||
      ((SOC_old < Hval) && (SOC > Hval)) )
    SOCcat = H,
if ( ((SOC_old > VHval) && (SOC < VHval)) ||

```

```

        ((SOC_old < VHval) && (SOC > VHval))    )
        SOCcat = VH,
    if (    ((SOC_old > VVHval) && (SOC < VVHval)) ||
        ((SOC_old < VVHval) && (SOC > VVHval)) )
        SOCcat = VVH,

    SOC_old = SOC,
    SOCcat_old = SOCcat,

    loops++,

    /* SOC isn't very dynamic, kwh meter only runs at 1Hz */
    ftask_delay(1 0),
}
}

```

D.2.10 throttle_ctrl.c

```

/* throttle_ctrl c */

#include <math h>      /* pow */
#include "fcar h"

/* macros used in this file */
#define throttle      s->analog[s->active] out oIceThrottlePos
#define ThrottleUp    s->cv[s->active] ThrottleUp

/* constants */
#define THROTTLE_RAMP_TIME    (3 1) /* seconds */
#define THROTTLE_MAX         (0 30) /* normal */
#define THROTTLE_MAX         0 28
#define ALPHA                 ((THROTTLE_MAX + 1 0) / THROTTLE_MAX)
#define BETA_BEND             (7 0)
#define THROTTLE_HERTZ       (10 0)

/* control Mikuni remote throttle asynchronously */
void throttle_control(void *unused)
{
    double t_init, t,

    notice("throttle controller ready"),

    while(s) /* while shm is valid */
    {
        /* wait around till throttle up signal is passed */
        ftask_trigger_block(),

        notice("got throttle up req"),

        t = 0 0,
        t_init = ftask_gettime(),

        /* go into ramp up loop */
        while(t <= THROTTLE_RAMP_TIME)
        {
            /* set throttle position */
            throttle = THROTTLE_MAX *
            (pow(ALPHA, BETA_BEND* t/THROTTLE_RAMP_TIME) - 1 0)
            / (pow(ALPHA, BETA_BEND) - 1 0),

```

```
        /* delay */
        ftask_delay( 1 0/THROTTLE_HERTZ),

        t = ftask_gettime() - t_init,
    }

    notice("throttle up done"),
}
/* shouldn't get here */
}
```

Appendix E

Support Libraries Code Listing

E.1 libfclient: IPC Library

E.1.1 Makefile

```
# requires gnu make

CC           =      cc
AR           =      ar
ARFLAGS     =      qcr
RANLIB      =      true
DEFINES     =
INCLUDES    =
QUIET       =      -Q -wx
OPTS        =      -Orailnextm -4r -fp3 -fp187
#DEBUG      =      -g
CFLAGS      =      $(QUIET) $(DEBUG) $(OPTS) $(DEFINES) $(INCLUDES)
FILES       =      fclient_create_shm.c fclient_open_shm.c
FILES       +=     fclient_delete_shm.c s_die.c
OBJS        =      $(FILES).o
LIBOUT      =      libfclient.a
DEPEND      =      makedepend
LIBDIR      =      /opt/fcar/lib/

all          $(LIBOUT)

install      $(LIBOUT)
             cp -f $(LIBOUT) $(LIBDIR)

dep          $(DEPEND) -- $(CFLAGS) -D_QNX__ -- $(FILES)

clean        rm -f $(OBJS) $(LIBOUT) core * err

$(LIBOUT)   $(OBJS)
            $(AR) $(ARFLAGS) $@ $^
            $(RANLIB) $@
```

E.1.2 fclient.h

```
/* fclient.h */
```

```

#ifndef FCLIENT_INCLUDED
#define FCLIENT_INCLUDED

#include <fcntl h>          /* open mode types */
#include "fcar_common h"  /* shared_hw_data structure definition */

#ifndef NULL
#define NULL 0
#endif

/* process control feedback */
enum { NOTICE, WARN, FATAL },
#define die(m)                s_die(__FILE__, __LINE__, m, FATAL)
#define warn(m)               s_die(__FILE__, __LINE__, m, WARN)
#define notice(m)             s_die(__FILE__, __LINE__, m, NOTICE)

#ifdef __cplusplus
extern "C" {
#endif

/* function prototypes */
extern struct shared_hw_data * fclient_create_shm(void),
extern int fclient_delete_shm(void),
extern struct shared_hw_data * fclient_open_shm(mode_t mode),
extern void s_die(char *, int, char *, int),

#ifdef __cplusplus
}
#endif

#endif

```

E.1.3 fcar_common.h

```

/* fcar_common h */
/* info shared with other programs */

#ifndef FCAR_COMMON_H
#define FCAR_COMMON_H

/* shows up in /dev/shmem */
#define FCAR_SHARED_HW_DATA_NAME "fcar_shared_hw_mem"

/* for data logging */
#define SECONDS_TO_STORE 3600 /* 1 hour's worth */

/* ice modes */
enum { OFF=0, ON, WORKING, DEAD },

/* mode types */
enum { ZEV_MODE, ECON_MODE, SPORT_MODE, REVERSE_MODE, PARK_MODE, NEUTRAL_MODE },

/* state of charge category types */
enum { VVL, VL, L, H, VH, VVH },

/* hybrid mode regime types */
enum { LO_REGIME, HI_REGIME },

/* hybrid transition types */

```

```

enum { UNDEFINED=0, SLOWING_DOWN, SPEEDING_UP },

struct calculated_values
{
    float SOC,           /* state of charge */
    int SOCcat,         /* state of charge category */
    int mode,           /* "mode" like ZEV, SPORT, etc */
    float vehicle_speed, /* speed in mph */
    float vehicle_distance, /* distance vehicle travelled since boot */
    float energy_level, /* total combined energy */
    float kwh_volts,    /* battery pack voltage */
    float kwh_amps,     /* battery pack current */
    int IceReqState,   /* requested state of the engine */
    int IceState,      /* actual state of the engine */
    int running,       /* fcar control program is running */
    int hybrid_regime, /* different gen/eng proportioning */
    int hybrid_transition, /* changes during lo-h1, h1-lo */
},

struct fcard_runtime_values
{
    unsigned fcard_hardware_timer_hz,

    int fcard_manager_prio,
    int fcard_hardware_task_prio,
    int fcard_kwh_task_prio,
    int fcard_config_task_prio,
},

/*
 * xxxSen - linear slope sensitivity
 * xxxOff - linear y-intercept
 */
struct calibration_values
{
    /* inputs */
    float iEmMotorSpeedSen,
    float iEmMotorSpeedOff,
    float iGenMotorSpeedSen,
    float iGenMotorSpeedOff,
    float iIceEngineSpeedSen,
    float iIceEngineSpeedOff,
    float iAccelPedalLevelSen,
    float iAccelPedalLevelOff,
    float iBrakePedalLevelSen,
    float iBrakePedalLevelOff,
    float iActualEmTorqueSen,
    float iActualEmTorqueOff,
    float iFuelPressureSen,
    float iFuelPressureOff;
    float iEmCurrentNegSen,
    float iEmCurrentNegOff,
    float iEmCurrentPosSen,
    float iEmCurrentPosOff,
    float iGenCurrentSen,
    float iGenCurrentOff,
    float iBattPackTempSen,
    float iBattPackTempOff,
    float iBattPackVoltageSen,
    float iBattPackVoltageOff,
    float iBattPackCurrentSen,

```



```

float iBattPackCurrentOff,
float iEmRotorTempSen,
float iEmRotorTempOff,
float iEmInvTempSen,
float iEmInvTempOff,
float iTpsFeedbackSen,
float iTpsFeedbackOff,
/* outputs */
float oGenSpeedReqSen,
float oGenSpeedReqOff,
float oGenRegenLimitSen,
float oGenRegenLimitOff,
float oEmAccelReqSen,
float oEmAccelReqOff,
float oEmBrakeReqSen,
float oEmBrakeReqOff,
float oIceThrottlePosSen,
float oIceThrottlePosOff,
float oDisSpeedo0Amplitude, /* speedometer based on sine */
float oDisSpeedo0PhaseAngle,
float oDisSpeedo1Amplitude,
float oDisSpeedo1PhaseAngle,
float oPwrToDigRacksSen,
float oPwrToDigRacksOff,
},

struct input_digital_hardware
{
    unsigned iPark,
    unsigned iReverse,
    unsigned iNeutral,
    unsigned iDriveSport,
    unsigned iDriveEcon,
    unsigned iZEV,
    unsigned iHvac,
    unsigned iEmTempWarn,
    unsigned iEmControllerReady,
    unsigned iEmFaultIndicator,
    unsigned iEmOvertempIndicator,
    unsigned iGenTempWarn,
    unsigned iGenControllerReady,
    unsigned iGenFaultIndicator,
    unsigned iGenDirectionIndicator,
    unsigned iIceFaultIndicator,
},

struct output_digital_hardware
{
    unsigned oIceStarter,
    unsigned oTecEnable,
    unsigned oEmEnable,
    unsigned oEmDirection,
    unsigned oGenEnable,
    unsigned oPwrSteeringEnable,
    unsigned oThrottlePwrCycle,
    unsigned oSmartChargerEnable,
},

struct digital_hardware
{

```

```

    struct input_digital_hardware in,
    struct output_digital_hardware out,
},

struct input_analog_hardware
{
    float iEmMotorSpeed,
    float iGenMotorSpeed,
    float iIceEngineSpeed,
    float iAccelPedalLevel,
    float iBrakePedalLevel,
    float iActualEmTorque,
    float iFuelPressure,
    float iEmCurrent,
    float iGenCurrent,
    float iBattPackTemp,
    float iBattPackVoltage,
    float iBattPackCurrent,
    float iEmRotorTemp,
    float iEmInvTemp,
    float iTpsFeedback,
},

struct output_analog_hardware
{
    float oGenSpeedReq,
    float oGenRegenLimit,
    float oEmAccelReq,
    float oEmBrakeReq,
    float oIceThrottlePos,
    float oDisSpeedo0,
    float oDisSpeedo1,
    float oPwrToDigRacks,
},

struct analog_hardware
{
    struct input_analog_hardware in,
    struct output_analog_hardware out,
},

struct shared_hw_data
{
    struct calibration_values cal,
    struct fcard_runtime_values fcard_rv,
    struct calculated_values cv[SECONDS_TO_STORE],
    struct digital_hardware digital[SECONDS_TO_STORE],
    struct analog_hardware analog[SECONDS_TO_STORE],
    int active,
    unsigned seconds,
},

#endif

```

E.1.4 fclient_create_shm.c

```
/* fclient_create_shm c */
```

```

#include <fcntl h>          /* shm_open */
#include <sys/mman h>       /* shm_unlink, shm_open, mmap */

```

```

#include <sys/types h>          /* ltrunc */
#include <unistd h>             /* close */
#include <string h>            /* memset */
#include <sys/stat h>          /* umask */
#include "fclient h"

struct shared_hw_data *fclient_create_shm(void)
{
    int fd,                    /* temporary */
    struct shared_hw_data *d,

    /* get rid of any of our old shared memory if it exists */
    (void)shm_unlink(FCAR_SHARED_HW_DATA_NAME),

    umask(0),                  /* total control */

    /* create shared memory object */
    fd = shm_open(FCAR_SHARED_HW_DATA_NAME, O_RDWR | O_CREAT, 0644),

    if (fd == -1) return NULL,

    /* set the shared memory object size */
#ifdef __QNX__
    /* for some reason QNX likes to use ltrunc while the
     * POSIX 1003 1b spec calls for using ftruncate */

    if (ltrunc(fd, sizeof(struct shared_hw_data), SEEK_SET) == -1)
        die("ltrunc"),
    #else
    if (ftruncate(fd, sizeof(struct shared_hw_data)) == -1)
        die("ftruncate"),
    #endif

    /* map memory object to local space */
    d = mmap(0, sizeof(struct shared_hw_data),
             PROT_READ | PROT_WRITE | PROT_NOCACHE,
             MAP_SHARED, fd, 0),

    if (d == (void *)-1) return NULL,

    /* clean up close file descriptor */
    (void)close(fd),

    /* initialize struct to zero */
    memset((void *)d, 0, sizeof(struct shared_hw_data)),

    /* done */
    return d,
}

```

E.1.5 fclient_delete_shm.c

```

/* fclient_delete_shm c */

#include <sys/mman h>          /* shm_unlink */
#include "fclient h"

int fclient_delete_shm(void)
{
    if (shm_unlink(FCAR_SHARED_HW_DATA_NAME) != 0)
        return 1,
}

```

```

    return 0,
}

```

E.1.6 fclient_open_shm.c

```

/* fclient_open_shm.c */
#include <stdio.h>
#include <fcntl.h> /* shm_open */
#include <sys/mman.h> /* shm_open, mmap */
#include <unistd.h> /* close */
#include <time.h> /* nanosleep */
#include "fclient.h"

#define WAITNSEC 100000000L/4L /* 0.25 seconds */
#define NUMTRIES 16 /* try for 4 seconds */

/* this might be tricky fcard may not have made the shared
 * memory segment yet Here we'll keep trying for a few seconds
 * and bail out if it's not there after a few tries */

struct shared_hw_data *fclient_open_shm(mode_t mode)
{
    int fd,
    struct shared_hw_data *d,
    struct timespec ts = { 0, WAITNSEC },
    int i = 0,

    /* open shared memory */
    while ( ((fd=shm_open(FCAR_SHARED_HW_DATA_NAME, O_RDONLY, 0644))==-1)
            && (i < NUMTRIES) )
    {
        if (!(i % 4)) notice("fclient trying to connect"),
            i++,
            nanosleep(&ts, NULL),
    }

    if (fd == -1) return NULL,

    /* map memory object to local space */
    switch (mode)
    {
        case O_RDONLY
            d = mmap(0, sizeof(struct shared_hw_data),
                    PROT_READ | PROT_NOCACHE,
                    MAP_SHARED, fd, 0),
            break,
        case O_RDWR
            d = mmap(0, sizeof(struct shared_hw_data),
                    PROT_READ | PROT_WRITE | PROT_NOCACHE,
                    MAP_SHARED, fd, 0),
            break,
        default
            return NULL,
    }

    if (d == (void *)-1) return NULL,

    (void)close(fd),
    return d,
}

```

E.1.7 s_die.c

```

/* s_die.c */
#include <stdio.h>      /* stderr */
#include <stdlib.h>     /* exit */
#include <time.h>       /* timestamp stuff */
#include <unistd.h>     /* getpid */
#include <string.h>     /* strerror, strlen */
#include "fcntl.h"

void s_die(char *wherehappen, int line, char *message, int severity)
{
    char    tbuf[64],      /* where to put time stamp */
           lbuf[256],     /* output string */
           time_t t,

    t = time(NULL),
    strftime(tbuf, sizeof(tbuf), "%m/%d/%y %H %M %S", localtime(&t));

    switch (severity)
    {
        case NOTICE
            sprintf(lbuf, "%s [%d] notice (%s %d) %s\n",
                tbuf, getpid(), wherehappen, line, message),
            break,
        case WARN
            sprintf(lbuf, "%s [%d] Warn  (%s %d) %s\n",
                tbuf, getpid(), wherehappen, line, message),
            break,
        case FATAL
            sprintf(lbuf, "%s [%d] FATAL (%s %d) %s %s\n",
                tbuf, getpid(), wherehappen, line, message,
                strerror(errno)),
            break,
        default
            sprintf(lbuf, "%s [%d] Huh?  (%s %d) %s %s\n",
                tbuf, getpid(), wherehappen, line, message,
                strerror(errno)),
    }
    /*
    if (strlen(lbuf)-1 > sizeof(lbuf))
        severity = FATAL, */

    write(STDERR_FILENO, lbuf, strlen(lbuf)),

    if (severity == FATAL) exit(EXIT_FAILURE),

    return,
}

```

E.2 libftask: Task Primitives Library

E.2.1 Makefile

```

# requires gnu make

TRIGGER_METHOD = qnx
#TRIGGER_METHOD = sig
#TRIGGER_METHOD = pipe

CC           =      cc
AR           =      ar
ARFLAGS     =      qcr
RANLIB      =      true
DEFINES     =      -DDEBUG
INCLUDES    =
QUIET       =      -Q -wx
OPTS        =      -Orailnextm -4r -fp3 -fp187
#DEBUG      =      -g
CFLAGS      =      $(QUIET) $(DEBUG) $(OPTS) $(DEFINES) $(INCLUDES)
LIBFILES    =      ftask c ftask_time c timer_qnx c

ifeq ($(TRIGGER_METHOD),qnx)
LIBFILES    +=      trigger_qnx c
endif

ifeq ($(TRIGGER_METHOD),pipe)
LIBFILES    +=      trigger_pipe c
endif

ifeq ($(TRIGGER_METHOD),sig)
LIBFILES    +=      trigger_sig c
endif

LIBOBSJS    =      $(LIBFILES c= o)
LIBOUT      =      libftask a
DEPEND      =      makedepend
THISDIR     =      $(shell basename $$PWD)
QNX_HOST    =      greed
QNX_PARENTDIR =      -/

all          $(LIBOUT) #demos

$(LIBOUT)   $(LIBOBSJS)
            $(RM) $(LIBOUT)
            $(AR) $(ARFLAGS) $@ $^
            $(RANLIB) $@

demos       $(MAKE) -C demos

ftask_hint o ftask_hint c
            $(CC) $(CFLAGS) -zu -Wc,-s -c $<

dep         $(DEPEND) -- $(CFLAGS) -D__QNX__ -- $(FILES)

install all

clean       rm -f $(LIBOBSJS) $(LIBOUT) * err

```

```

$(MAKE) -C demos clean

rsync
    cd    && rsync -avzt $(THISDIR) $(QNX_HOST) $(QNX_PARENTDIR)

rmake    rsync
         rsh $(QNX_HOST) '(cd $(QNX_PARENTDIR)$(THISDIR) && make)

rclean
    cd    && rsync -avzt --delete $(THISDIR) $(QNX_HOST) $(QNX_PARENTDIR)

PHONY    demos dep clean install rsync rmake rclean

```

E.2.2 ftask.h

```

/* ftask h */

#ifndef FTASK_INCLUDED
#define FTASK_INCLUDED

#ifdef __QNX__
#include <sys/sched.h> /* sched_setscheduler, sched_yield */
#else
#include <sched.h>
#endif

#include <sys/types.h> /* pid_t */

struct ftask
{
    int policy,
    int priority,
    void (*start_func)(void *),
    void *start_func_arg,
    void (*cleanup_func)(int),
    int allow_trigger,
    int allow_periodic_timer,
    int periodic_timer_hz,
    pid_t fid, /* filled in after fork */
    union
    {
        pid_t trig_proxy, /* this is only used for qnx triggers */
        int pipefds[2], /* only used for pipe triggers */
    },
};

typedef struct ftask ftask,

#ifdef __cplusplus
extern "C" {
#endif

/* public function prototypes */
extern int ftask_init(ftask *ft, /* handle */
                    int policy, /* valid scheduling policy */
                    int priority, /* valid scheduling priority */
                    void (*start_func)(void *), /* task entry function */
                    void *start_func_arg, /* entry function argument */
                    void (*cleanup_func)(int), /* "destructor" */
                    int allow_trigger, /* triggering ability bool */

```

```

        int allow_periodic_timer,      /* use periodic timer bool */
        int periodic_timer_hz,        /* integer hz (if enabled) */

extern int ftask_delay(double sec),
extern int ftask_sched_adjust_self(int policy, int priority),
extern int ftask_register_cleanup_self(void (*cleanup_func)(int)),
extern int ftask_register_reread_self(void (*reread_func)(int)),
extern int ftask_delete(ftask *ft),
extern int ftask_destroy(ftask *ft),
extern int ftask_trigger(ftask *ft),
extern int ftask_trigger_block(void),
extern int ftask_create(ftask *ft),
extern int ftask_wait_on_tasks(ftask *ret),
extern int ftask_same(ftask *first, ftask *second),

extern int ftask_periodic_timer_block(void),

extern double ftask_gettime(void),

#ifdef __cplusplus
}
#endif

#endif

```

E.2.3 ftask_private.h

```

/* ftask_private h - internal stuff for libftask */

#ifndef FTASK_PRIVATE_INCLUDED
#define FTASK_PRIVATE_INCLUDED

#define FTASK_DELETE_SIG      SIGTERM
#define FTASK_DESTROY_SIG    SIGKILL
#define FTASK_TRIGGER_SIG    SIGUSR1
#define FTASK_REREAD_SIG     SIGHUP

#ifdef DEBUG
#include <stdio h>
#define DTRACE(x)             printf x
#else
#define DTRACE(x)
#endif

#endif

```

E.2.4 timer_qnx_private.h

```

/* ftask_hint_p h - private stuff */

#ifndef TIMER_QNX_PRIVATE_INCLUDED
#define TIMER_QNX_PRIVATE_INCLUDED

/* private function prototypes */
pid_t far timer_inthandler(void),
int timer_hint_attach(int divisor),
void timer_hint_detach(void),
void timer_register_cleanup(void),
int timer_get_timer_hz(void),

```



```
#endif
```

E.2.5 trigger_private.h

```
/* ftask_hint_p h - private stuff */
```

```
#ifndef TRIGGER_PRIVATE_INCLUDED
#define TRIGGER_PRIVATE_INCLUDED
```

```
/* private function prototypes */
int trigger_setup(ftask *ft),
void trigger_register_cleanup(void),
int parent_prefork(ftask *ft),
int parent_postfork(ftask *ft),
```

```
#endif
```

E.2.6 ftask.c

```
/* ftask c */
```

```
#include <stdlib.h> /* atexit */
#include <sys/types.h> /* fork, kill, wait */
#include <unistd.h> /* fork, getuid */
#include <signal.h> /* sigaction, kill */
#include <sys/wait.h> /* wait */
#include <errno.h>
#include "ftask.h"
#include "ftask_private.h"
#include "timer_qnx_private.h"
#include "trigger_private.h"
```

```
int ftask_init(ftask *ft,
              int policy,
              int priority,
              void (*start_func)(void *),
              void *start_func_arg,
              void (*cleanup_func)(int),
              int allow_trigger,
              int allow_periodic_timer,
              int periodic_timer_hz)
```

```
{
```

```
  if (ft)
  {
```

```
    ft->policy = policy,
    ft->priority = priority,
    ft->start_func = start_func,
    ft->start_func_arg = start_func_arg,
    ft->cleanup_func = cleanup_func,
    ft->allow_trigger = allow_trigger,
```

```
    /* this implementation doesn't allow both a
     * trigger and periodic timer, enforce that
     * here */
```

```
    if (allow_periodic_timer && allow_trigger)
```

```
    {
```

```
        DTRACE(("periodic timer and trigger not allowed\n")),
        goto bad_init,
```

```
    }

    if (allow_periodic_timer && (periodic_timer_hz <= 0))
    {
        DTRACE(("periodic timer cannot be negative\n")),
        goto bad_init,
    }

    ft->allow_periodic_timer = allow_periodic_timer,
    ft->periodic_timer_hz = periodic_timer_hz,
    /* fid filled in after fork */

    return 0,
}

bad_init
    errno = EINVAL,
    return -1,
}

/* scheduling parameters are implicitly inherited across fork */
int ftask_sched_adjust_self(int policy, int priority)
{
    struct sched_param sp,

    sp sched_priority = priority,

    return (sched_setscheduler(0, policy, &sp)),
}

/* signal handlers are also inherited */

/* handler for SIGTERM */
int ftask_register_cleanup_self(void (*cleanup_func)(int))
{
    struct sigaction act,

    act sa_handler = cleanup_func,
    sigemptyset(&act sa_mask),
    act sa_flags = 0,

    return(sigaction(FTASK_DELETE_SIG, &act, NULL)),
}

/* handler for SIGHUP */
int ftask_register_reread_self(void (*reread_func)(int))
{
    struct sigaction act,

    act sa_handler = reread_func,
    sigemptyset(&act sa_mask),
    act sa_flags = 0,

    return(sigaction(FTASK_REREAD_SIG, &act, NULL)),
}

int ftask_delete(ftask *ft)
{
    return (kill(ft->fid, FTASK_DELETE_SIG)),
}
```

```

int ftask_destroy(ftask *ft)
{
    return (kill(ft->fid, FTASK_DESTROY_SIG)),
}

void ftask_generic_cleanup_func(int sig)
{
    exit(EXIT_SUCCESS),
}
\
int ftask_create(ftask *ft)
{
    /* might need to do some setup */
    if (parent_prefork(ft) == -1)
        return -1,

    /* first off fork */
    switch(ft->fid=fork())
    {
        case 0      /* child */
            break,
        case -1     /* err */
            return -1,
        default     /* parent */
            return (parent_postfork(ft)),
    }

    /* set policy, priority */
    if (ftask_sched_adjust_self(ft->policy, ft->priority) == -1)
    {
        DTRACE(("ftask_sched_adjust_self failed for %d\n",
                getpid())),
        goto fail,
    }

    /* (re)set all signal handlers possibly
     * messed with to the default */
    if (ftask_register_cleanup_self(SIG_DFL) == -1)
    {
        DTRACE(("reseting cleanup failed for %d\n", getpid())),
        goto fail,
    }

    if (ftask_register_reread_self(SIG_DFL) == -1)
    {
        DTRACE(("reseting reread failed for %d\n", getpid())),
        goto fail,
    }

    /* even if a cleanup routine isn't requested for a
     * periodic timer task, add one that just calls exit(),
     * this lets the atexit() call later on work */
    if ((ft->cleanup_func == NULL) && ft->allow_periodic_timer)
        ft->cleanup_func = ftask_generic_cleanup_func,

    /* same for triggers */
    if ((ft->cleanup_func == NULL) && ft->allow_trigger)
        ft->cleanup_func = ftask_generic_cleanup_func,
}

```

```
/* register the cleanup function, if applicable */
if (ft->cleanup_func)
{
    if (ftask_register_cleanup_self(ft->cleanup_func) == -1)
    {
        DTRACE(("ftask_register_cleanup failed for %d\n",
                getpid()),
                goto fail,
            }
    }

/* if triggering ability is requested */
if (ft->allow_trigger)
{
    if (trigger_setup(ft) == -1)
    {
        DTRACE(("trigger setup failed for %d\n", getpid()),
                goto fail,
            }

    trigger_register_cleanup(),
}

if (ft->allow_periodic_timer)
{
    int current_hz = timer_get_timer_hz(),
        int div = current_hz / ft->periodic_timer_hz,

    if (div <= 0)
    {
        /* req hz > current hz, or current_hz fail */
        /* increasing the timer freq not supported (yet) */
        DTRACE(("bad div for %d\n", getpid()),
                goto fail,
            }
    }

    if (getuid() != 0) /* don't even attempt if not root */
    {
        DTRACE(("you must be root to attach to the timer\n"),
                goto fail,
            }
    }

    if (timer_hint_attach(div) == -1)
    {
        DTRACE(("timer_hint_attach failed for %d\n",
                getpid()),
                goto fail,
            }
    }

    timer_register_cleanup(),
}

/* call the start routine */
if (ft->start_func)
    ft->start_func(ft->start_func_arg),

/* shouldn't return, but call cleanup if available */
if (ft->cleanup_func)
{
    ft->cleanup_func(0),
}
```

```

    }

    exit(EXIT_SUCCESS), /* task complete successful */

fail
    /* should print some message */
    DTRACE(("ftask_create reached \"fail\" mark\n")),
    exit(EXIT_FAILURE),

    return -1, /* keep compiler happy */
}

int ftask_wait_on_tasks(ftask *ret)
{
    int status,
    ret->fid = wait(&status),
    return status,
}

/* more like similar */
int ftask_same(ftask *first, ftask *second)
{
    if (first->fid == second->fid)
        return 1,
    return 0,
}

/*****/

```

E.2.7 ftask_time.c

```

#ifdef __QNX__
#include <sys/time h> /* nanosleep */
#else
#include <time h>
#endif

#include <time h> /* clock_gettime */
#include "ftask h"

double ftask_gettime(void)
{
    struct timespec ts,

    clock_gettime(CLOCK_REALTIME, &ts),
    return ((double)ts tv_sec + (double)(ts tv_nsec)/1000000000 0),
}

int ftask_delay(double sec)
{
    struct timespec ts,

    ts tv_sec = (time_t)sec, /* trunc */
    ts tv_nsec =(time_t)((sec - (double)ts tv_sec)*1000000000 0),

    return(nanosleep(&ts, NULL)),
}

```

E.2.8 timer_qnx.c

```

/* timer_qnx.c */

#include <stdlib.h>           /* atexit */
#include <sys/irqinfo.h>     /* qnx_hint_attach */
#include <sys/proxy.h>       /* qnx_proxy_attach */
#include <sys/osinfo.h>      /* qnx_osinfo */
#include <sys/kernel.h>      /* FP_SEG, Receive */
#include <time.h>            /* qnx_ticksize */

#include "ftask.h"
#include "ftask_private.h"
#include "timer_qnx_private.h"

#define FTASK_TIMER_IRQ      0      /* 386 pc timer interrupt */

/* a hardware interrupt handler, must be compiled
 * with "-zu -Wc,-s" in the CFLAGS */

static volatile unsigned irqcounter = 0,
static volatile unsigned timer_ticks,
static pid_t timer_proxy,
static int qnx_interrupt_id,

#pragma off(check_stack),

pid_t far timer_inthandler(void)
{
    ++irqcounter,
    if (irqcounter == timer_ticks)
    {
        irqcounter = 0,
        return timer_proxy,
    }
    return 0,
}

#pragma on(check_stack),

/* APIentry */
int ftask_periodic_timer_block(void)
{
    return(Receive(timer_proxy, 0, 0)),
}

int timer_hint_attach(int divisor)
{
    /* get a proxy */
    if ( (timer_proxy=qnx_proxy_attach(0,0,0,-1)) == -1)
        return -1,

    timer_ticks = divisor,

    /* attach handler to hardware interrupt */
    if ( (qnx_interrupt_id=qnx_hint_attach(FTASK_TIMER_IRQ,
        &timer_inthandler, FP_SEG(&irqcounter))) == -1)
        return -1,

    return 0,
}

```

```

}

void timer_hint_detach(void)
{
    qnx_hint_detach(qnx_interrupt_id),
    qnx_proxy_detach(timer_proxy),
}

int timer_get_timer_hz(void)
{
    struct _osinfo osi,
    unsigned usec,

    if (qnx_osinfo(0, (struct _osinfo *)&osi) == -1)
        return -1,

    usec = (unsigned)osi tick_size,

    return (int)(1000000U / usec),
}

#if 0
/* this is a bad function, processes shouldn't
 * arbitrarily mess with the timer tick */
int ftask_set_timer_hz(int hz)
{
    long nsec,

    nsec = 1000000000L / (long)hz,

    /* standard ticksizes 5, 1, 2, 5, 10, 25, 50, 55 ms
     *                      2000,1000,500,200,100, 40, 20, 18 2 Hz */

    if (qnx_ticksizes(nsec, _TICKSIZE_STANDARD) == -1)
        return -1,
    return 0,
}
#endif

void timer_register_cleanup(void)
{
    /* there should be a better way to do this make
     * sure that the interrupt handler is released
     * before the task exits */
    atexit(timer_hint_detach),
}

```

E.2.9 trigger_pipe.c

```

#include <unistd.h>          /* pipe, read, write */
#include <stdlib.h>         /* atexit */
#include <errno.h>
#include "ftask.h"
#include "ftask_private.h"

static int readfd,

```

```
int parent_prefork(ftask *ft)
{
    /* initialize pipe */
    return(pipe(ft->pipefds)),
}

int parent_postfork(ftask *ft)
{
    /* give child a chance to run */
    sched_yield(),

    /* parent doesn't use reader side */
    return (close(ft->pipefds[0])),
}

int trigger_setup(ftask *ft)
{
    readfd = ft->pipefds[0],

    /* child doesn't use writer side */
    return (close(ft->pipefds[1])),
}

/* APIentry */
int ftask_trigger(ftask *ft)
{
    int msg = 0,

    if (ft->allow_trigger)
    {
        if (write(ft->pipefds[1], &msg, sizeof(msg)) == -1)
            return -1,

        return 0,
    }

    errno = EINVAL,
    return -1,
}

/* APIentry */
int ftask_trigger_block(void)
{
    int msg,

    if (read(readfd, &msg, sizeof(msg)) == -1)
        return -1,

    return 0,
}

void trigger_detach(void)
{
    close(readfd),
}

void trigger_register_cleanup(void)
{
    atexit(trigger_detach),
}
```


E.2.10 trigger_qnx.c

```

#include <sys/proxy.h>          /* qnx_proxy_attach */
#include <sys/kernel.h>        /* FP_SEG, Receive, Send */
#include <unistd.h>            /* getpid */
#include <stdlib.h>            /* atexit */
#include <errno.h>
#include "ftask.h"
#include "ftask_private.h"

static pid_t trig_proxy,

int parent_prefork(ftask *ft)
{
    /* nothing */
    return 0,
}

int parent_postfork(ftask *ft)
{
    if (ft->allow_trigger)
    {
        /* wait for the proxy number from child */
        if (Receive( ft->fid, (void *)&trig_proxy,
                    sizeof(trig_proxy)) == -1)
            return -1,

        ft->trig_proxy = trig_proxy,

        /* send back a null reply (OK) */
        if (Reply(ft->fid, NULL, 0) == -1)
            return -1,
    }
    return 0,
}

int trigger_setup(ftask *ft)
{
    /* get a proxy */
    if ((trig_proxy = qnx_proxy_attach(0,0,0,-1)) == -1)
        return -1,

    /* let the parent know the proxy number, so that it
     * can trigger later */
    Send(getpid(), (void *)&trig_proxy, NULL,
         sizeof(trig_proxy), NULL),

    return 0,
}

/* APIentry */
int ftask_trigger(ftask *ft)
{
    if (ft->allow_trigger)
    {
        return (Trigger(ft->trig_proxy)),
    }

    errno = EINVAL,
    return -1,
}

```

```
}

/* APIentry */
int ftask_trigger_block(void)
{
    return(Receive(trig_proxy, 0, 0)),
}

void trigger_detach(void)
{
    qnx_proxy_detach(trig_proxy),
}

void trigger_register_cleanup(void)
{
    atexit(trigger_detach),
}
```

E.2.11 trigger_sig.c

```
#include <signal h>
#include <errno h>
#include "ftask h"
#include "ftask_private h"

int parent_prefork(ftask *unused)
{
    /* nothing */
    return 0,
}

int parent_postfork(ftask *unused)
{
    /* give child a chance to run */
    sched_yield(),
    sched_yield(),
    sched_yield(),
    sched_yield(),
    return 0,
}

void ftask_null_handler(int ignored)
{
    /* null handler, just return */
}

int trigger_setup(ftask *unused)
{
    struct sigaction act,
    sigset_t block_these,

    sigemptyset(&block_these),
    sigaddset(&block_these, FTASK_TRIGGER_SIG),
    sigprocmask(SIG_BLOCK, &block_these, NULL),

    act.sa_handler = ftask_null_handler,
    sigemptyset(&act.sa_mask),
    act.sa_flags = 0,

    return(sigaction(FTASK_TRIGGER_SIG, &act, NULL)),
}
```

```
}

/* APIentry */
int ftask_trigger(ftask *ft)
{
    if (ft->allow_trigger)
        return (kill(ft->fid, FTASK_TRIGGER_SIG)),
        errno = EINVAL,
        return -1,
}

/* APIentry */
int ftask_trigger_block(void)
{
    /* this complication allows the trigger signal to
     * be blocked, that is it won't interrupt the
     * task until it calls ftask_trigger_block() */
    sigset_t s,
    sigemptyset(&s),
    return(sigsuspend(&s)),
}

void trigger_register_cleanup(void)
{
    /* nothing here */
}
```

Appendix F

User Interface Code Listing

F.1 vfd: Vacuum Flourescent Display program

F.1.1 Makefile

```
CC          =      cc
DEFINES     =
INCLUDES    =      -I /libfclient -I /cgi/libcgl
QUIET       =      -Q -wx
OPTS        =      -Ora1nextm -4r -fp3 -fp187
DEBUG       =      #-g
CFLAGS      =      $(QUIET) $(DEBUG) $(OPTS) $(DEFINES) $(INCLUDES)
LIBS        =      -L /libfclient
LIBS        +=     -lfclient
LDFLAGS     =      $(QUIET) $(LIBS)
LIBS_CONFIG =      -L /cgi/libcgl -L /libfclient
LIBS_CONFIG +=    -lcgl -lfclient
LDFLAGS_CONFIG = $(QUIET) $(LIBS_CONFIG)
FILES       =      vfd.c
FILES_CONFIG =     vfdconfig c
OBJS        =      $(FILES c= o)
OBJS_CONFIG =     $(FILES_CONFIG c= o)
OUT         =      vfd
OUT_CONFIG  =      vfdconfig cgi
DEPEND      =      makedepend
BINDIR      =      /opt/fcar/bin/
CGI_BINDIR  =      /usr/local/apache/cgi-bin/

all          $(OUT) $(OUT_CONFIG)

$(OUT)       $(OBJS)
             $(CC) $(LDFLAGS) $^ -o $@

$(OUT_CONFIG) $(OBJS_CONFIG)
              $(CC) $(LDFLAGS_CONFIG) $^ -o $@

install     $(OUT) $(OUT_CONFIG)
             cp -f $(OUT) $(BINDIR)
             cp -f $(OUT_CONFIG) $(CGI_BINDIR)

dep         $(DEPEND) -- $(CFLAGS) -D_QNX__ -- $(FILES) $(FILES_CONFIG)
```

```

clean
    $(RM) $(OUT) $(OUT_CONFIG) $(OBJS) $(OBJS_CONFIG) * err

F.1.2 vfd.h
/* vfd h */

#ifndef VFD_INCLUDED
#define VFD_INCLUDED

#define VFD_DEVICE        "//1/dev/ser1" /* qnx node 1 == octagon hardware */
#define VFD_SHM_NAME      "vfd_shared_mem"

#define MAX_DESC_LEN      32          /* maximum description label length */

struct vfd_config
{
    char bignum[MAX_DESC_LEN],
    char row1[MAX_DESC_LEN],
    char row2[MAX_DESC_LEN],
    char row3[MAX_DESC_LEN],
    char row4[MAX_DESC_LEN],
},

#define DEFAULT_BIG        "vehicle_speed"
#define DEFAULT_ROW1       "SOC"
#define DEFAULT_ROW2       "iFuelPressure"
// #define DEFAULT_ROW2     "vehicle_distance"
#define DEFAULT_ROW3       "mode"
#define DEFAULT_ROW4       "iBattPackVoltage"

struct vfd_screen
{
    int    bignum_value,
    int    row1_value,
    int    row2_value,
    int    row3_value,
    int    row4_value,
    char   *bignum_label,
    char   *row1_label,
    char   *row2_label,
    char   *row3_label,
    char   *row4_label,
},

#define SOC_VALUE          (int)(100 * s->cv[s->active] SOC)
#define SOC_LABEL          "% SOC"

#define vehicle_speed_VALUE (int)s->cv[s->active] vehicle_speed
#define vehicle_speed_LABEL " mph "

#define vehicle_distance_VALUE (int)s->cv[s->active] vehicle_distance
#define vehicle_distance_LABEL " dist"

#define energy_level_VALUE (int)(100 * s->cv[s->active] energy_level)
#define energy_level_LABEL "% En "

#define mode_VALUE         (int)(s->cv[s->active] mode)
#define mode_LABEL         " mode"

#define iEmMotorSpeed_VALUE (int)(s->analog[s->active] in iEmMotorSpeed)

```

```

#define iEmMotorSpeed_LABEL      "TmRPM"

#define iGenMotorSpeed_VALUE      (int)(s->analog[s->active] in iGenMotorSpeed)
#define iGenMotorSpeed_LABEL     "GnRPM"

#define iIceEngineSpeed_VALUE    (int)(s->analog[s->active] in iIceEngineSpeed)
#define iIceEngineSpeed_LABEL    "EnRPM"

#define iAccelPedalLevel_VALUE   (int)(s->analog[s->active] in iAccelPedalLevel)
#define iAccelPedalLevel_LABEL   "% APS"

#define iBrakePedalLevel_VALUE   (int)(s->analog[s->active] in iBrakePedalLevel)
#define iBrakePedalLevel_LABEL   "% BPS"

#define iActualEmTorque_VALUE    (int)(s->analog[s->active] in iActualEmTorque)
#define iActualEmTorque_LABEL    "TmN-m"

#define iFuelPressure_VALUE      (int)((s->analog[s->active] in iFuelPressure)\
                                     *100 OF/3600 OF)
#define iFuelPressure_LABEL      "% CNG"

#define iEmCurrent_VALUE         (int)(s->analog[s->active] in iEmCurrent)
#define iEmCurrent_LABEL         "A Em"

#define iGenCurrent_VALUE        (int)(s->analog[s->active] in iGenCurrent)
#define iGenCurrent_LABEL        "A Gen"

#define iBattPackTemp_VALUE      (int)(s->analog[s->active] in iBattPackTemp)
/* ascii es-sett translates to a degree symbol in the vfd */
#define iBattPackTemp_LABEL      "C Bt"

#define iBattPackVoltage_VALUE   (int)(s->analog[s->active] in iBattPackVoltage)
#define iBattPackVoltage_LABEL   "V bus"

#define iBattPackCurrent_VALUE   (int)(s->analog[s->active] in iBattPackCurrent)
#define iBattPackCurrent_LABEL   "A bus"

#define iEmRotorTemp_VALUE       (int)(s->analog[s->active] in iEmRotorTemp)
#define iEmRotorTemp_LABEL       "C Rt"

#define iEmInvTemp_VALUE         (int)(s->analog[s->active] in iEmInvTemp)
#define iEmInvTemp_LABEL         "C It"

#define iTpsFeedback_VALUE       (int)(100 * s->analog[s->active] in iTpsFeedback)
#define iTpsFeedback_LABEL       "% Tps"

#define oGenSpeedReq_VALUE       (int)(s->analog[s->active] out oGenSpeedReq)
#define oGenSpeedReq_LABEL       "rpm G"

#define oGenRegenLimit_VALUE     (int)(100 * s->analog[s->active] out oGenSpeedReq)
#define oGenRegenLimit_LABEL     "% rgn"

#define oEmAccelReq_VALUE        (int)(100 * s->analog[s->active] out oEmAccelReq)
#define oEmAccelReq_LABEL        "% aTm"

#define oEmBrakeReq_VALUE        (int)(100 * s->analog[s->active] out oEmBrakeReq)
#define oEmBrakeReq_LABEL        "% bTm"

#define oIceThrottlePos_VALUE    (int)(100 * s->analog[s->active] out oIceThrottlePos)
#define oIceThrottlePos_LABEL    "% thr"

```

```

#define oDisSpeedo0_VALUE      (int)(s->analog[s->active] out oDisSpeedo0)
#define oDisSpeedo0_LABEL     " Spd0"

#define oDisSpeedo1_VALUE      (int)(s->analog[s->active] out oDisSpeedo1)
#define oDisSpeedo1_LABEL     " Spd1"

#define LOOKFOR(what, where)
    if (strcmp(v->where, #what) == 0)
    {
        vs where##_value = what##_VALUE,
        vs where##_label = what##_LABEL,
    }

#define LOOKFORBIG()
    LOOKFOR(SOC, bignum)           else \
    LOOKFOR(vehicle_speed, bignum) else \
    LOOKFOR(energy_level, bignum)

#define LOOKFORROW(row)
    LOOKFOR(SOC,row)             else \
    LOOKFOR(mode,row)           else \
    LOOKFOR(vehicle_speed,row)  else \
    LOOKFOR(vehicle_distance,row) else \
    LOOKFOR(energy_level,row)   else \
    LOOKFOR(iEmMotorSpeed,row)  else \
    LOOKFOR(iGenMotorSpeed,row) else \
    LOOKFOR(iIceEngineSpeed,row) else \
    LOOKFOR(iAccelPedalLevel,row) else \
    LOOKFOR(iBrakePedalLevel,row) else \
    LOOKFOR(iActualEmTorque,row) else \
    LOOKFOR(iFuelPressure,row)  else \
    LOOKFOR(iEmCurrent,row)     else \
    LOOKFOR(iGenCurrent,row)    else \
    LOOKFOR(iBattPackTemp,row)  else \
    LOOKFOR(iBattPackVoltage,row) else \
    LOOKFOR(iBattPackCurrent,row) else \
    LOOKFOR(iEmRotorTemp,row)   else \
    LOOKFOR(iEmInvTemp,row)     else \
    LOOKFOR(iTpsFeedback,row)   else \
    LOOKFOR(oGenSpeedReq,row)   else \
    LOOKFOR(oGenRegenLimit,row) else \
    LOOKFOR(oEmAccelReq,row)    else \
    LOOKFOR(oEmBrakeReq,row)    else \
    LOOKFOR(oIceThrottlePos,row) else \
    LOOKFOR(oDisSpeedo0,row)    else \
    LOOKFOR(oDisSpeedo1,row)

#endif

```

F.1.3 vfd.c

```

/* vfd c */
/* control Matrix Orbital Vacuum Fluorescent Display on /dev/ser1 */

#include <stdio h>
#include <stdlib h> /* exit, atoi */
#include <fcntl h> /* shm_open */
#include <sys/mman h> /* shm_unlink, shm_open, mmap */

```

```

#include <sys/types.h> /* ltrunc */
#include <sys/stat.h> /* umask */
#include <string.h> /* memset */
#include <termios.h> /* terminal control functions */
#include <unistd.h> /* terminal control functions, close */
#include <time.h> /* nanosleep */
#include <fcntl.h>
#include "vfd.h"

/* function prototypes */
void setup_vfd_shm(void);

/* global variables */
static volatile struct vfd_config *v;
static volatile struct shared_hw_data *s;

void setup_vfd_shm()
{
    int fd, /* temp file descriptor */

    /* first get rid of any old segment (if it exists) */
    (void)shm_unlink(VFD_SHM_NAME),

    umask(0), /* no assumptions on file creat */

    /* create shared memory object */
    /* note mode rw-rw-rw (0666) so that the cgi config
     * program can change variable even as a non-privileged user */
    fd = shm_open(VFD_SHM_NAME, O_RDWR | O_CREAT, 0666),

    if (fd == -1) die("shm_open"),

    /* set the shared memory object size */
    if (ltrunc(fd, sizeof(struct vfd_config), SEEK_SET) == -1)
        die("ltrunc"),

    /* map memory object to local space */
    v = mmap(0, sizeof(struct vfd_config),
            PROT_READ | PROT_WRITE | PROT_NOCACHE,
            MAP_SHARED, fd, 0),

    if (v == (void *)-1) die("mmap"),

    /* cleanup */
    if (close(fd) != 0) die("close"),

    /* this would be nice if it works
     * unlink the shared mem segment, but it doesn't really
     * go away until its reference count is zero, that is,
     * when this program exits */
    /* if (shm_unlink(VFD_SHM_NAME) != 0) die("shm_unlink"), */
    /* update doesn't work */

    /* initialize the segment to zero */
    memset((void *)v, 0, sizeof(struct vfd_config)),
}

int main(int argc, char *argv[])
{
    FILE *vfdout, /* serial port */

```



```

int vfd_fd,          /* file descriptor */
struct termios attr, /* serial port terminal attributes */
speed_t speed,      /* serial port baud rate */
struct vfd_screen vs,

/* setup our psuedo-private shared memory segment */
setup_vfd_shm(),

notice("vfd shm setup done"),

/* put default values in the vfd shared mem */
strncpy(v->bignum, DEFAULT_BIG, MAX_DESC_LEN),
strncpy(v->row1, DEFAULT_ROW1, MAX_DESC_LEN),
strncpy(v->row2, DEFAULT_ROW2, MAX_DESC_LEN),
strncpy(v->row3, DEFAULT_ROW3, MAX_DESC_LEN),
strncpy(v->row4, DEFAULT_ROW4, MAX_DESC_LEN),

/* open the public fcar shared memory segment */
if ((s = fclient_open_shm(O_RDONLY)) == NULL)
    die("fclient_open_shm(O_RDONLY)"),

notice("attached to fcar shm ok"),

/* open the serial port to which the vfd is attached */
if ((vfdout = fopen(VFD_DEVICE, "w")) == NULL) die("fopen"),
if ((vfd_fd = fileno(vfdout)) == -1) die("fileno"),

/**/ setup the serial port /**/
/* get current settings */
if (tcgetattr(vfd_fd, &attr) == -1) die("tcgetattr"),

/* change the speed */
speed = B19200,
if ((cfsetispeed(&attr, speed) || cfsetospeed(&attr, speed)) != 0)
    die("cfsetxspeed"),

/* terminal changes happen immediately */
if (tcsetattr(vfd_fd, TCSANOW, &attr) == -1) die("tcsetattr"),

/**/ done with serial port setup /**/
notice("serial setup done"),

/* clearscreen, large characters on */
fprintf(vfdout, "%c%c%c", '\f', 0xfe, '\n'),

while(s->cv[s->active] running) /* fcar control process is running */
{
    int tens, ones,          /* for big numbers */
    struct timespec t,      /* for nanosleep */

    t.tv_sec = 0,
    t.tv_nsec = 500000000L, /* half a second refresh rate */

    /* bignum */
    LOOKFORBIG(),

    /* fabs */
    if (vs.bignum_value < 0 OF)
        vs.bignum_value = -vs.bignum_value,

```

```

ones = vs bignum_value % 10,
tens = (vs bignum_value - ones) / 10,

if (tens > 9 ) tens = 0,

/* rows */
LOOKFORROW(row1),
LOOKFORROW(row2),
LOOKFORROW(row3),
LOOKFORROW(row4),

/* actually write out to the vfd */
/* big numbers */
fprintf(vfdout, "%c%c%c%c", 0xfe, '#', 1, tens), /* 3 */
fprintf(vfdout, "%c%c%c%c", 0xfe, '#', 4, ones), /* 6 */

/* go to row 4, column 7, bignum label */
fprintf(vfdout, "%c%c%c%c%s",
        0xfe, 'G', 7, 4, vs bignum_label), /* 5 wide */

/* go to row 1, column 12, row1 stuff */
fprintf(vfdout, "%c%c%c%c%0 2d%s",
        0xfe, 'G', 12, 1, vs row1_value, vs row1_label),

/* go to row 2, column 12, row2 stuff */
fprintf(vfdout, "%c%c%c%c%0 2d%s",
        0xfe, 'G', 12, 2, vs row2_value, vs row2_label),

/* go to row 3, column 12, row3 stuff */
if (strstr(vs row3_label, "mode") != NULL)
    fprintf(vfdout, "%c%c%c%c%s%s",
            0xfe, 'G', 12, 3,
            (vs row3_value == ZEV_MODE) ? "ZEV"
            (vs row3_value == ECON_MODE) ? "HEV"
            (vs row3_value == SPORT_MODE) ? "SPT"
            (vs row3_value == REVERSE_MODE) ? "REV"
            (vs row3_value == PARK_MODE) ? "PRK"
            (vs row3_value == NEUTRAL_MODE) ? "NEU"   "???",
            vs row3_label),
else
    fprintf(vfdout, "%c%c%c%c%0 2d%s",
            0xfe, 'G', 12, 3, vs row3_value, vs row3_label),

/* go to row 4, column 12, row4 stuff */
fprintf(vfdout, "%c%c%c%c%0 2d%s",
        0xfe, 'G', 12, 4, vs row4_value, vs row4_label),

/* done writing */
fflush(vfdout),
nanosleep(&t, NULL),
}

notice("exiting"),
if (shm_unlink(VFD_SHM_NAME) != 0)
    die("shm_unlink"),

return 0,
}

```

Appendix G

Diagnostic Program Code Listings

G.1 mon: Shared Memory Display Utility

G.1.1 Makefile

```
CC           =      cc
DEFINES      =
INCLUDES     =      -I /libfclient
QUIET        =      -Q -wx
OPTS         =      -Ora1nextm -4r -fp3 -fp187
DEBUG        =      #-g
CFLAGS       =      $(QUIET) $(DEBUG) $(OPTS) $(DEFINES) $(INCLUDES)
LIBS         =      -L /libfclient
LIBS         +=     -lfclient -lincurses
LDFLAGS      =      $(QUIET) $(LIBS)
FILES        =      mon c inputs c outputs c cal c mode c
OBJS         =      $(FILES c= o)
OUT          =      mon
DEPEND       =      makedepend
BINDIR       =      /opt/fcar/bin/

all          $(OUT)

$(OUT)       $(OBJS)
             $(CC) $(LDFLAGS) $^ -o $@

dep
             $(DEPEND) -- $(CFLAGS) -D__QNX__ -- $(FILES)

install      $(OUT)
             cp -f $(OUT) $(BINDIR)

clean
             $(RM) $(OUT) $(OBJS) * o * err
```

G.1.2 mon.h

```
/* mon h */

#ifdef MON_H
#define MON_H
```

```

#include <fcntl.h>

#define INPUTS_KEY    'i'
#define OUTPUTS_KEY  'o'
#define CALIBRATION_KEY 'c'
#define MODE_KEY     'm'
#define QUIT_KEY     'q'

#define STATS_LINE    LINES-1
#define COL1          0
#define COL2          (COLS/4+3)
#define COL3          (COLS/2)
#define COL4          (COLS*3/4+3)

/* types WA_NORMAL, WA_STANDOUT, WA_UNDERLINE, WA_REVERSE, WA_BLINK,
   WA_DIM, WA_BOLD, WA_ALTCHARSET
*/

#define STATS_LINE_ATTR WA_BOLD

/* globals */
extern volatile struct shared_hw_data *s,

/* function protoypes */
void finish(int),
void show_inputs(void),
void show_outputs(void),
void show_calibration(void),
void show_mode(void),
void calibration_label(void),
void inputs_label(void),
void outputs_label(void),
void mode_label(void),

#endif

```

G.1.3 cal.c

```

/* cal c */

#include < curses.h>
#include "mon.h"

#define FIRST_COL(what) \
    sprintf(buf, #what) , \
    mvaddstr(row,COL1,buf), \
    sprintf(buf, "% 0 3f", s->cal what), \
    mvaddstr(row,COL2,buf), \
    row++

#define SECOND_COL(what) \
    sprintf(buf, #what) , \
    mvaddstr(row,COL3,buf), \
    sprintf(buf, "% 0 3f", s->cal what), \
    mvaddstr(row,COL4,buf), \
    row++

void show_calibration()
{

```

```

int row,
char buf[64],

row = 0,
/* inputs */
FIRST_COL(1EmMotorSpeedSen),
FIRST_COL(1EmMotorSpeedOff),
FIRST_COL(1GenMotorSpeedSen),
FIRST_COL(1GenMotorSpeedOff),
FIRST_COL(1IceEngineSpeedSen),
FIRST_COL(1IceEngineSpeedOff),
FIRST_COL(1AccelPedalLevelSen),
FIRST_COL(1AccelPedalLevelOff),
FIRST_COL(1BrakePedalLevelSen),
FIRST_COL(1BrakePedalLevelOff),
FIRST_COL(1ActualEmTorqueSen),
FIRST_COL(1ActualEmTorqueOff),
FIRST_COL(1FuelPressureSen),
FIRST_COL(1FuelPressureOff),
FIRST_COL(1EmCurrentNegSen),
FIRST_COL(1EmCurrentNegOff),
FIRST_COL(1EmCurrentPosSen),
FIRST_COL(1EmCurrentPosOff),
FIRST_COL(1GenCurrentSen),
FIRST_COL(1GenCurrentOff),
FIRST_COL(1BattPackVoltageSen),
FIRST_COL(1BattPackVoltageOff),
FIRST_COL(1BattPackCurrentSen),
row = 0,
SECOND_COL(1BattPackCurrentOff),
SECOND_COL(1EmRotorTempSen),
SECOND_COL(1EmRotorTempOff),
SECOND_COL(1EmInvTempSen),
SECOND_COL(1EmInvTempOff),
SECOND_COL(1TpsFeedbackSen),
SECOND_COL(1TpsFeedbackOff),
/* outputs */
SECOND_COL(oGenSpeedReqSen),
SECOND_COL(oGenSpeedReqOff),
SECOND_COL(oGenRegenLimitSen),
SECOND_COL(oGenRegenLimitOff),
SECOND_COL(oEmAccelReqSen),
SECOND_COL(oEmAccelReqOff),
SECOND_COL(oEmBrakeReqSen),
SECOND_COL(oEmBrakeReqOff),
SECOND_COL(oIceThrottlePosSen),
SECOND_COL(oIceThrottlePosOff),
SECOND_COL(oDisSpeedo0Amplitude),
SECOND_COL(oDisSpeedo0PhaseAngle),
SECOND_COL(oDisSpeedo1Amplitude),
SECOND_COL(oDisSpeedo1PhaseAngle),
SECOND_COL(oPwrToDigRacksSen),
SECOND_COL(oPwrToDigRacksOff),

/* speedometer based on sine */

return,
}

```

G.1.4 inputs.c

```
/* inputs c */
```

```

#include <curses h>
#include "mon h"

#define FIRST_COL(what,where)          \
    sprintf(buf, #what) ,              \
    mvaddstr(row,COL1,buf),            \
    sprintf(buf, "%d", where what),    \
    mvaddstr(row,COL2,buf),            \
    row++

#define SECOND_COL(what,where)         \
    sprintf(buf, #what) ,              \
    mvaddstr(row,COL3,buf),            \
    sprintf(buf, "% 0 3f", where what), \
    mvaddstr(row,COL4,buf),            \
    row++

#define IDIG_FIRST_COL(what)           FIRST_COL(what,s->digital[s->active] in)
#define IDIG_SECOND_COL(what)          SECOND_COL(what,s->digital[s->active] in)

#define IANA_FIRST_COL(what)           FIRST_COL(what,s->analog[s->active] in)
#define IANA_SECOND_COL(what)          SECOND_COL(what,s->analog[s->active] in)

void show_inputs()
{
    int row,
    char buf[64],

    /* digital */
    row = 0,
    sprintf(buf, "Digital"), mvaddstr(row++,COL1,buf),
    IDIG_FIRST_COL(iPark),
    IDIG_FIRST_COL(iReverse),
    IDIG_FIRST_COL(iNeutral),
    IDIG_FIRST_COL(iDriveSport),
    IDIG_FIRST_COL(iDriveEcon),
    IDIG_FIRST_COL(iZEV),
    IDIG_FIRST_COL(iHvac),
    IDIG_FIRST_COL(iEmTempWarn),
    IDIG_FIRST_COL(iEmControllerReady),
    IDIG_FIRST_COL(iEmFaultIndicator),
    IDIG_FIRST_COL(iEmOvertempIndicator),
    IDIG_FIRST_COL(iGenTempWarn),
    IDIG_FIRST_COL(iGenControllerReady),
    IDIG_FIRST_COL(iGenFaultIndicator),
    IDIG_FIRST_COL(iGenDirectionIndicator),
    IDIG_FIRST_COL(iIceFaultIndicator), /* 16 */

    /* analog */
    row = 0,
    sprintf(buf, "Analog"), mvaddstr(row++,COL3,buf),
    IANA_SECOND_COL(iEmMotorSpeed),
    IANA_SECOND_COL(iGenMotorSpeed),
    IANA_SECOND_COL(iIceEngineSpeed),
    IANA_SECOND_COL(iAccelPedalLevel),
    IANA_SECOND_COL(iBrakePedalLevel),
    IANA_SECOND_COL(iActualEmTorque),
    IANA_SECOND_COL(iFuelPressure),
    IANA_SECOND_COL(iEmCurrent),
    IANA_SECOND_COL(iGenCurrent),

```

```

        IANA_SECOND_COL(1BattPackTemp),
        IANA_SECOND_COL(1BattPackVoltage),
        IANA_SECOND_COL(1BattPackCurrent),
        IANA_SECOND_COL(1EmRotorTemp),
        IANA_SECOND_COL(1EmInvTemp),
        IANA_SECOND_COL(1TpsFeedback),

        return,
    }

```

G.1.5 mode.c

```

/* mode c */

#include <curses h>
#include "mon h"

void show_mode()
{
    int row,

    row = 0,
    mvprintw(row, COL1, "SOC"),
    mvprintw(row, COL2, "%0 3f", s->cv[s->active] SOC), row++,

    mvprintw(row, COL1, "SOCcat"),
    mvprintw(row, COL2, "%s",
        (s->cv[s->active] SOCcat == VVH) ? "VVH"
        (s->cv[s->active] SOCcat == VH) ? "VH"
        (s->cv[s->active] SOCcat == H) ? "H"
        (s->cv[s->active] SOCcat == L) ? "L"
        (s->cv[s->active] SOCcat == VL) ? "VL"
        (s->cv[s->active] SOCcat == VVL) ? "VVL" " " ), row++,

    mvprintw(row, COL1, "mode"),
    mvprintw(row, COL2, "%s",
        (s->cv[s->active] mode == ZEV_MODE) ? "ZEV"
        (s->cv[s->active] mode == ECON_MODE) ? "ECON"
        (s->cv[s->active] mode == SPORT_MODE) ? "SPORT"
        (s->cv[s->active] mode == REVERSE_MODE) ? "REVERSE"
        (s->cv[s->active] mode == PARK_MODE) ? "PARK"
        (s->cv[s->active] mode == NEUTRAL_MODE) ? "NEUTRAL"
        " " ), row++,

    mvprintw(row, COL1, "vehicle_speed"),
    mvprintw(row, COL2, "% 0 3f", s->cv[s->active] vehicle_speed), row++,

    mvprintw(row, COL1, "vehicle_distance"),
    mvprintw(row, COL2, "% 0 3f", s->cv[s->active] vehicle_distance); row++,

    mvprintw(row, COL1, "energy_level"),
    mvprintw(row, COL2, "% 0 3f", s->cv[s->active] energy_level), row++,

    mvprintw(row, COL1, "kwh_volts"),
    mvprintw(row, COL2, "% 0 3f", s->cv[s->active] kwh_volts), row++,

    mvprintw(row, COL1, "kwh_amps"),
    mvprintw(row, COL2, "% 0 3f", s->cv[s->active] kwh_amps), row++,

```

```

mvprintw(row, COL1, "IceReqState"),
mvprintw(row, COL2, "%s",
(s->cv[s->active] IceReqState == ON) ?      "ON"
(s->cv[s->active] IceReqState == OFF) ?     "OFF"
"      "), row++,

mvprintw(row, COL1, "IceState"),
mvprintw(row, COL2, "%s",
(s->cv[s->active] IceState == ON) ?          "ON"
(s->cv[s->active] IceState == OFF) ?        "OFF"
(s->cv[s->active] IceState == WORKING) ?    "WORKING"
(s->cv[s->active] IceState == DEAD) ?      "DEAD"
"      "), row++,

mvprintw(row, COL1, "hybrid_regime"),
mvprintw(row, COL2, "%s",
(s->cv[s->active] hybrid_regime == LO_REGIME) ? "LO"
(s->cv[s->active] hybrid_regime == HI_REGIME) ? "HI"
"      "), row++,

row = 0,
mvprintw(row, COL3, "timer_hertz"),
mvprintw(row, COL4, "%u",
s->fcard_rv fcard_hardware_timer_hz), row++,

mvprintw(row, COL3, "manager_prio"),
mvprintw(row, COL4, "%d",
s->fcard_rv fcard_manager_prio), row++,

mvprintw(row, COL3, "hardware_task_prio"),
mvprintw(row, COL4, "%d",
s->fcard_rv fcard_hardware_task_prio), row++,

mvprintw(row, COL3, "kwh_task_prio"),
mvprintw(row, COL4, "%d",
s->fcard_rv fcard_kwh_task_prio), row++,

mvprintw(row, COL3, "config_task_prio"),
mvprintw(row, COL4, "%d",
s->fcard_rv fcard_config_task_prio), row++,

mvprintw(row, COL3, "active pointer"),
mvprintw(row, COL4, "%d", s->active), row++,

mvprintw(row, COL3, "total seconds"),
mvprintw(row, COL4, "%u", s->seconds), row++,

return,
}

```

G.1.6 mon.c

```

/* mon c */

#include <stdio h>
#include <stdlib h>
#include <string h>

```



```

#include <curses h>
#include <signal h>
#include "mon h"

volatile struct shared_hw_data *s,

void finish(int sig)
{
    curs_set(1),
    endwin(),
    exit(0),
}

void reinit(int sig)
{
    erase(),
    refresh(),
    endwin(),
    initscr(),           /* initialize the curses library */
    keypad(stdscr, TRUE), /* enable keyboard mapping */
    nonl(),              /* tell curses not to do NL->CR/NL on output */
    cbreak(),           /* take input chars one at a time, no wait for \n */
    noecho(),           /* don't echo input */
    /* leaveok(stdscr, TRUE), /* turn off cursor if possible */
    curs_set(0),        /* magic ??? */
    timeout(500),       /* getch times out in 500 milliseconds */
    return,
}

int main(int argc, char *argv[])
{
    void (*last)(void), /* pointer to function that displays stuff */
    void (*last_label)(void),

    (void)signal(SIGINT, finish), /* goto finish on ctrl-c */
    (void)signal(SIGTERM, finish), /* goto finish on terminate */
    (void)signal(SIGWINCH, reinit), /* reinit on window size change */

    if (argc == 2)
    {
        /* catch the unique part */
        if (strstr(argv[1], "c") != NULL)
        {
            last = show_calibration,
            last_label = calibration_label,
        }
        else if (strstr(argv[1], "in") != NULL)
        {
            last = show_inputs,
            last_label = inputs_label,
        }
        else if (strstr(argv[1], "out") != NULL)
        {
            last = show_outputs,
            last_label = outputs_label,
        }
        else if (strstr(argv[1], "m") != NULL)
        {
            last = show_mode,
            last_label = mode_label,
        }
    }
}

```

```

        else
        {
            printf("Usage %s [calibration|inputs|outputs|mode]\n",
                argv[0],
                exit(1),
            }
    }
else
{
    last = show_mode,          /* default */
    last_label = mode_label,
}

/* open fcar shared mem */
if ((s = fclient_open_shm(0_RDONLY)) == NULL)
{
    perror("fclient_open_shm_ro"),
    exit(1),
}

initscr(),                  /* initialize the curses library */
keypad(stdscr, TRUE),      /* enable keyboard mapping */
nonl(),                    /* tell curses not to do NL->CR/NL on output */
cbreak(),                  /* take input chars one at a time, no wait for \n */
noecho(),                  /* don't echo input */
/* leaveok(stdscr, TRUE), /* turn off cursor if possible */
curs_set(0),               /* magic ??? */
timeout(500),              /* getch times out in 500 milliseconds */

while(s)
{
    erase(),
    last(),
    last_label(),
    refresh(),
    switch(getch())
    {
        case INPUTS_KEY
            last = show_inputs,
            last_label = inputs_label,
            break,
        case OUTPUTS_KEY
            last = show_outputs,
            last_label = outputs_label,
            break,
        case CALIBRATION_KEY
            last = show_calibration,
            last_label = calibration_label;
            break;
        case MODE_KEY
            last = show_mode,
            last_label = mode_label,
            break,
        case KEY_LEFT
            if (last == show_calibration)
            {
                last = show_mode,
                last_label = mode_label,
            }
            else if (last == show_mode)
            {

```

```

        last = show_outputs,
        last_label = outputs_label,
    }
    else if (last == show_outputs)
    {
        last = show_inputs,
        last_label = inputs_label,
    }
    else if (last == show_inputs)
    {
        last = show_calibration,
        last_label = calibration_label,
    }
    break,
case KEY_RIGHT
    if (last == show_calibration)
    {
        last = show_inputs,
        last_label = inputs_label,
    }
    else if (last == show_mode)
    {
        last = show_calibration,
        last_label = calibration_label,
    }
    else if (last == show_outputs)
    {
        last = show_mode,
        last_label = mode_label,
    }
    else if (last == show_inputs)
    {
        last = show_outputs,
        last_label = outputs_label,
    }
    break,
case QUIT_KEY
    finish(0),
default
    break,
}
}
/* doesn't get here */
finish(0),
return 0,
}

void calibration_label()
{
    int len,
    char buf[80],

    attron(STATS_LINE_ATTR),
    len = sprintf(buf,"%c - calibration", CALIBRATION_KEY),
    mvaddstr(STATS_LINE,0,buf),
    attroff(STATS_LINE_ATTR),

    sprintf(buf, " %c - inputs %c - outputs %c - mode %c - quit",
        INPUTS_KEY, OUTPUTS_KEY, MODE_KEY, QUIT_KEY),
    mvaddstr(STATS_LINE,len,buf),
}

```

```
void inputs_label()
{
    int len, l,
    char buf[80],

    len = sprintf(buf,"%c - calibration  ", CALIBRATION_KEY),
    mvaddstr(STATS_LINE,0,buf),

    attron(STATS_LINE_ATTR),
    l = sprintf(buf,"%c - inputs", INPUTS_KEY),
    mvaddstr(STATS_LINE, len, buf),
    len += 1,
    attroff(STATS_LINE_ATTR),

    sprintf(buf," %c - outputs %c - mode %c - quit",
            OUTPUTS_KEY, MODE_KEY, QUIT_KEY),
    mvaddstr(STATS_LINE, len, buf),
}

void outputs_label()
{
    int len, l,
    char buf[80],

    len = sprintf(buf,"%c - calibration %c - inputs  ",
            CALIBRATION_KEY, INPUTS_KEY),
    mvaddstr(STATS_LINE,0,buf),

    attron(STATS_LINE_ATTR),
    l = sprintf(buf,"%c - outputs", OUTPUTS_KEY),
    mvaddstr(STATS_LINE, len, buf),
    len += 1,
    attroff(STATS_LINE_ATTR),

    sprintf(buf," %c - mode %c - quit",
            MODE_KEY, QUIT_KEY),
    mvaddstr(STATS_LINE, len, buf),
}

void mode_label()
{
    int len, l,
    char buf[80],

    len = sprintf(buf,"%c - calibration %c - inputs %c - outputs  ",
            CALIBRATION_KEY, INPUTS_KEY, OUTPUTS_KEY),
    mvaddstr(STATS_LINE,0,buf),

    attron(STATS_LINE_ATTR),
    l = sprintf(buf,"%c - mode", MODE_KEY),
    mvaddstr(STATS_LINE, len, buf),
    len += 1,
    attroff(STATS_LINE_ATTR),

    sprintf(buf," %c - quit", QUIT_KEY),
    mvaddstr(STATS_LINE, len, buf),
}
```

G.1.7 outputs.c

```

/* inputs c */

#include <curses h>
#include "mon h"

#define FIRST_COL(what,where)          \
    sprintf(buf, #what) ,              \
    mvaddstr(row,COL1,buf),            \
    sprintf(buf, "%d", where what),    \
    mvaddstr(row,COL2,buf),            \
    row++

#define SECOND_COL(what,where)         \
    sprintf(buf, #what) ,              \
    mvaddstr(row,COL3,buf),            \
    sprintf(buf, "% 0 3f", where what), \
    mvaddstr(row,COL4,buf),            \
    row++

#define ODIG_FIRST_COL(what)           FIRST_COL(what,s->digital[s->active] out)
#define ODIG_SECOND_COL(what)          SECOND_COL(what,s->digital[s->active] out)

#define OANA_FIRST_COL(what)           FIRST_COL(what,s->analog[s->active] out)
#define OANA_SECOND_COL(what)          SECOND_COL(what,s->analog[s->active] out)

void show_outputs()
{
    int row,
    char buf[64],

    /* digital */
    row = 0,
    sprintf(buf, "Digital"), mvaddstr(row++,COL1,buf),
    ODIG_FIRST_COL(oIceStarter),
    ODIG_FIRST_COL(oTecEnable),
    ODIG_FIRST_COL(oEmEnable),
    ODIG_FIRST_COL(oEmDirection),
    ODIG_FIRST_COL(oGenEnable),
    ODIG_FIRST_COL(oPwrSteeringEnable),
    ODIG_FIRST_COL(oThrottlePwrCycle),
    ODIG_FIRST_COL(oSmartChargerEnable),

    /* analog */
    row = 0,
    sprintf(buf, "Analog"), mvaddstr(row++,COL3,buf),
    OANA_SECOND_COL(oGenSpeedReq),
    OANA_SECOND_COL(oGenRegenLimit),
    OANA_SECOND_COL(oEmAccelReq),
    OANA_SECOND_COL(oEmBrakeReq),
    OANA_SECOND_COL(oIceThrottlePos),
    OANA_SECOND_COL(oDisSpeedo0),
    OANA_SECOND_COL(oDisSpeedo1),
    OANA_SECOND_COL(oPwrToDigRacks),

    return,
}

```

G.2 flogger: Data Logging Utility

G.2.1 Makefile

```

CC          =      cc
DEFINES     =
INCLUDES    =      -I /libfclient
QUIET       =      -Q -wx
OPTS        =      -Ora1nextm -4r -fp3 -fp187
DEBUG       =      #-g
CFLAGS      =      $(QUIET) $(DEBUG) $(OPTS) $(DEFINES) $(INCLUDES)
LIBS        =      -L /libfclient
LIBS        +=     -lfclient
LDFLAGS     =      $(QUIET) $(LIBS)
FILES       =      flogger c
OBJS        =      $(FILES c= o)
OUT         =      flogger
DEPEND      =      makedepend
BINDIR      =      /opt/fcar/bin/

all          $(OUT)

$(OUT)      $(OBJS)
            $(CC) $(LDFLAGS) $^ -o $@

dep          $(DEPEND) -- $(CFLAGS) -D__QNX__ -- $(FILES)

install     $(OUT)
            cp -f $(OUT) $(BINDIR)

clean       $(RM) $(OUT) $(OBJS) * o * err

```

G.2.2 flogger.c

```

#include <stdio h>          /* printf */
#include <stdlib h>         /* exit */
#include <fclient h>        /* shared mem */
#include <time h>           /* timestamp stuff */
#include <unistd h>         /* sleep, getopt */
#include <string h>         /* strdup, memcpy */
#include <time h>           /* nanosleep */

#define DEFAULT_LOGFILE    "flogger log"
#define DEFAULT_UPDATE     1 OF
#define DEFAULT_FLUSHCOUNT 10

struct logvalues
{
    char tbuf[64],
    struct calculated_values cv,
    struct digital_hardware digital,
    struct analog_hardware analog,
},

/* global variables */
volatile struct shared_hw_data *s,

```

```

/* function prototypes */
void usage(char *),

void usage(char *p)
{
    printf("Usage %s [-l logfile] [-f flushtime] [-u update]\n",p),
    printf("    defaults are \n"),
    printf("    logfile - %s\n", DEFAULT_LOGFILE),
    printf("    flushcount - %d updates\n", DEFAULT_FLUSHCOUNT),
    printf("    update - %f seconds\n", DEFAULT_UPDATE),
    exit(1),
}

int dump_log(char *filename, struct logvalues *l, int count, int headers)
{
    int i,
    FILE *f,
    f = fopen(filename, "a"),
    if (f == NULL) die("fopen"),

    if (headers)
    {
        /* print headers */
        fprintf(f,"time," "SOC," "mode,"
            "vehicle_speed," "vehicle_distance," "energy_level,"
            "kwh_volts," "kwh_amps," "hybrid_regime,"),

        fprintf(f,"shifter," "iHvac," "iEMTempWarn,"
            "iEMControllerReady," "iEMFaultIndicator,"
            "iEmOvertempIndicator," "iGenTempWarn,"
            "iGenControllerReady," "iGenFaultIndicator,"
            "iGenDirectionIndicator," "iIceFaultIndicator,"),

        fprintf(f,"oIceStarter," "oTecEnable," "oEmEnable,"
            "oEmDirection," "oPwrSteeringEnable,"
            "oThrottlePwrCycle," "oSmartChargerEnable,"),

        fprintf(f,"iEmMotorSpeed," "iGenMotorSpeed," "iIceEngineSpeed,"
            "iAccelPedalLevel," "iBrakePedalLevel,"
            "iActualEmTorque," "iFuelPressure," "iEmCurrent,"
            "iGenCurrent," "iBattPackTemp," "iBattPackVoltage,"
            "iBattPackCurrent," "iEmRotorTemp," "iEmInvTemp,"
            "iTpsFeedback,"),

        fprintf(f,"oGenSpeedReq," "oGenRegenLimit," "oEmAccelReq,"
            "oEmBrakeReq," "oIceThrottlePos," "oDisSpeedo0,"
            "oDisSpeedo1," "oPwrToDigRacks\n"),
    }

    for (i=0,i<count,i++)
    {
        /* print out the data */
        fprintf(f,"%s,%f,%s,%f,%f,%f,%f,%f,%s,",
            l[i] tbuf,
            l[i] cv SOC,

            (l[i] cv mode == ZEV_MODE) ? "ZEV"
            (l[i] cv mode == ECON_MODE) ? "ECON"
            (l[i] cv mode == SPORT_MODE) ? "SPORT"

```



```

        l[1] analog out oEmBrakeReq,
        l[1] analog out oIceThrottlePos,
        l[1] analog out oDisSpeedo0,
        l[1] analog out oDisSpeedo1,
        l[1] analog out oPwrToDigRacks),
    }

    return (fclose(f)),
}

int main(int argc, char *argv[])
{
    struct logvalues *l,
    int c,
    int errflag = 0,
    char *logfile = DEFAULT_LOGFILE,
    float update = DEFAULT_UPDATE,
    int flushcount = DEFAULT_FLUSHCOUNT,
    size_t size,
    struct timespec t,
    int first_time = 1,

    while ( (c=getopt(argc,argv, "l f u h")) != -1)
    {
        switch (c)
        {
            case 'l'
                logfile = strdup(optarg),
                if (logfile == NULL) die("strdup"),
                break,
            case 'f'
                flushcount = atoi(optarg),
                break,
            case 'u'
                update = atof(optarg),
                break,
            case 'h'
                usage(argv[0]),
                break,
            case '?'
                ++errflag,
                break,
        }

        if (errflag) usage(argv[0]),
    }

    size = (size_t)flushcount,
    l = (struct logvalues *)calloc(size, sizeof(struct logvalues)),
    if (l == NULL) die("calloc"),

    t.tv_sec = (int)update,
    t.tv_nsec = (long)(((double)update - (double)t.tv_sec)*1e9),

    /* open fcar shared mem */
    if ((s = fclient_open_shm(0_RDONLY)) == NULL)
        die("fclient_open_shm(0_RDONLY)"),

```

```
while(s)
{
    int i,
    time_t tm,
    for(i=0,i<size,i++)
    {
        tm = time(NULL),
        strftime(l[i] tbuf, sizeof(l[i] tbuf),
            "%m/%d/%y %H %M %S", localtime(&tm)),

        memcpy(&l[i] cv, &s->cv[s->active],
            sizeof(struct calculated_values)),

        memcpy(&l[i].digital, &s->digital[s->active],
            sizeof(struct digital_hardware)),

        memcpy(&l[i] analog, &s->analog[s->active],
            sizeof(struct analog_hardware)),

        nanosleep(&t, NULL),
    }

    if (first_time)
    {
        dump_log(logfile, i, size, 1),
        first_time = 0,
    }
    else
        dump_log(logfile, i, size, 0),
}

free(l),
return 0,
}
```

G.3 trends.cgi: CGI Interface to Shared Memory History

G.3.1 Makefile

```

CC          =      cc
DEFINES     =
INCLUDES    =      -I / /libfclient -I /libcgi
QUIET       =      -Q -wx
OPTS        =      -Ora1nextm -4r -fp3 -fp187
DEBUG       =      #-g
CFLAGS      =      $(QUIET) $(DEBUG) $(OPTS) $(DEFINES) $(INCLUDES)
LIBS        =      -L /libcgi
LIBS        +=     -lcgi
LDFLAGS     =      $(QUIET) $(LIBS)
LIBS_HELPER =      -L / /libfclient -L /libcgi
LIBS_HELPER +=     -lfclient -lcgi
LDFLAGS_HELPER =    $(QUIET) $(LIBS_HELPER)
FILES       =      trends c
FILES_HELPER =     trend_helper c
OBJS        =      $(FILES c= o)
OBJS_HELPER =      $(FILES_HELPER c= o)
OUT         =      trends cgi
OUT_HELPER  =      trend_helper cgi
DEPEND      =      makedepend
CGI_BINDIR  =      /usr/local/apache/cgi-bin/

all         $(OUT) $(OUT_HELPER)

$(OUT)      $(OBJS)
            $(CC) $(LDFLAGS) $^ -o $@

$(OUT_HELPER) $(OBJS_HELPER)
            $(CC) $(LDFLAGS_HELPER) $^ -o $@

install     $(OUT) $(OUT_HELPER)
            cp -f $(OUT) $(OUT_HELPER) $(CGI_BINDIR)

dep         $(DEPEND) -- $(CFLAGS) -D__QNX__ -- $(FILES) $(FILES_HELPER)

clean       $(RM) $(OUT) $(OUT_HELPER) $(OBJS) $(OBJS_HELPER) * err

```

G.3.2 trends.h

```

/* trends h */

#ifndef TRENDS_INCLUDED
#define TRENDS_INCLUDED

#define WIDTH 500
#define HEIGHT 200

#ifdef DEBUG
#define NOTFOUNDPATH \
"Location http //mechaero9 engr utk edu/~matt/images/notfound gif\n\n"
#else
#define NOTFOUNDPATH \
"Location http //128 169 100 192/images/notfound gif\n\n"
#endif

```

```
#define BGCOLOR      "ffffff"
#define TEXTCOLOR   "000000"

#endif
```

G.3.3 trends.c

```
/* trends c */

#include <stdio h>
#include <stdlib h>
#include <string h>
#include <cgi h>
#include "trends h"

#define SHOW_OPTION(what) \
    printf("<option value=\"%s\"> %s \n", #what, #what)

#define ALL_OPTIONS() \
    SHOW_OPTION(iEmMotorSpeed), \
    SHOW_OPTION(iGenMotorSpeed), \
    SHOW_OPTION(iIceEngineSpeed), \
    SHOW_OPTION(iAccelPedalLevel), \
    SHOW_OPTION(iBrakePedalLevel), \
    SHOW_OPTION(iActualEmTorque), \
    SHOW_OPTION(iFuelPressure), \
    SHOW_OPTION(iEmCurrent), \
    SHOW_OPTION(iGenCurrent), \
    SHOW_OPTION(iBattPackTemp), \
    SHOW_OPTION(iBattPackVoltage), \
    SHOW_OPTION(iBattPackCurrent), \
    SHOW_OPTION(iEmRotorTemp), \
    SHOW_OPTION(iEmInvTemp), \
    SHOW_OPTION(iTpsFeedback), \
    \
    SHOW_OPTION(oGenSpeedReq), \
    SHOW_OPTION(oGenRegenLimit), \
    SHOW_OPTION(oEmAccelReq), \
    SHOW_OPTION(oEmBrakeReq), \
    SHOW_OPTION(oIceThrottlePos), \
    SHOW_OPTION(oDisSpeedo0), \
    SHOW_OPTION(oDisSpeedo1), \
    SHOW_OPTION(oPwrToDigRacks), \
    \
    SHOW_OPTION(SOC), \
    SHOW_OPTION(vehicle_speed), \
    SHOW_OPTION(vehicle_distance), \
    SHOW_OPTION(kwh_volts), \
    SHOW_OPTION(kwh_amps),

void show_choices(void),
void display_graphs(void),

int main(int argc, char *argv[])
{
    printf("Content-type text/html\n\n"),
```

```

    if (strcmp(getenv("REQUEST_METHOD"), "POST") == 0)
        display_graphs(),
    else
        show_choices(),

    return 0,
}

void display_graphs()
{
    char **cgivars,

    cgivars = getcgivars(), /* cgivars [name value] pairs */

    printf("                                \n\
    <html>                                \n\
    <head>                                  \n\
    <meta http-content-refresh=\"30\">      \n\
        <title> Trend graphs </title>      \n\
    </head>                                 \n\
    <body>                                  \n\
    You may have to hold down the shift key when reloading \n\
    to get updated info                    \n\
    <form action=\"trends.cgi\" method=post> \n\
    <h2> Trend results </h2><br>           \n\
    <input type=\"submit\" value=\"Submit\"> \n\
    <input type=\"reset\" value=\"Reset\">   \n\
    <hr>                                    \n\"),

    /* first row */
    printf("                                \n\
    <select name=\"graph1\">               \n\
    <option value=\"%s\" selected>default - %s\n\", cgivars[1], cgivars[1]),
    ALL_OPTIONS(),
    printf("                                \n\
    </select><br>                          \n\
    <img src=\"trend_helper.cgi?%s\" alt=\"%s\" width=%d height=%d\n\
    <hr width=\"50%\">\",
    cgivars[1], cgivars[1], WIDTH, HEIGHT),

    /* second row */
    printf("                                \n\
    <select name=\"graph2\">               \n\
    <option value=\"%s\" selected>default - %s\n\", cgivars[3], cgivars[3]),
    ALL_OPTIONS(),
    printf("                                \n\
    </select><br>                          \n\
    <img src=\"trend_helper.cgi?%s\" alt=\"%s\" width=%d, height=%d\n\
    <hr width=\"50%\">\",
    cgivars[3], cgivars[3], WIDTH, HEIGHT),

    /* third row */
    printf("                                \n\
    <select name=\"graph3\">               \n\
    <option value=\"%s\" selected>default - %s\n\", cgivars[5], cgivars[5]),
    ALL_OPTIONS(),
    printf("                                \n\
    </select><br>                          \n\
    <img src=\"trend_helper.cgi?%s\" alt=\"%s\" width=%d, height=%d\n\
    cgivars[5], cgivars[5], WIDTH, HEIGHT),

```

```

        /* ending */
        printf("                                \n\
        <hr>                                \n\
        <input type=\"submit\" value=\"Submit\"> \n\
        <input type=\"reset\" value=\"Reset \"> \n\
        </body>                                \n\"),
    }

void show_choices()
{
    printf("                                \n\
    <html>                                \n\
    <head>                                \n\
        <title> Choose trends to display </title> \n\
    </head>                                \n\
    <body>                                \n\
    <form action=\"trends cgi\" method=post> \n\
    <h2> Choose the trends you want to display </h2><br> \n\
    <input type=\"submit\" value=\"Submit\"> \n\
    <input type=\"reset\" value=\"Reset \"> \n\
    <hr>                                \n\
    <select name=\"graph1\"> \n\
    <option value=\"none\" selected>none \n\"),
    ALL_OPTIONS(),
    printf("                                \n\
    </select>                                \n\
    <br>                                \n\
    <select name=\"graph2\"> \n\
    <option value=\"none\" selected>none \n\"),
    ALL_OPTIONS(),
    printf("                                \n\
    </select>                                \n\
    <br>                                \n\
    <select name=\"graph3\"> \n\
    <option value=\"none\" selected>none \n\"),
    ALL_OPTIONS(),
    printf("                                \n\
    </select>                                \n\
    <br>                                \n\
    <hr>                                \n\
    <input type=\"submit\" value=\"Submit\"> \n\
    <input type=\"reset\" value=\"Reset \"> \n\
    </body>                                \n\
    </html>                                \n\");
}

```

G.3.4 trend_helper.c

```

/* trend_helper c */

#include <string h>
#include <stdio h>      /* printf, popen */
#include <stdlib h>     /* exit */
#include <fcntl h>
#include <cgi h>
#include "trends h"

#define GNUPLOT_BIN    "/usr/local/bin/gnuplot"

#define DO_ANALOG_IN(what) \

```

```

    if (strcmp(cgivars[0], #what) == 0)
    {
        for(i=0,i<now,i++)
        {
            fprintf(p, "%f\n", s->analog[i] in what),
        }
        fprintf(p, "e\n"), /* end of data */
        fflush(p),
    }

#define DO_ANALOG_OUT(what)
    if (strcmp(cgivars[0], #what) == 0)
    {
        for(i=0,i<now,i++)
        {
            fprintf(p, "%f\n", s->analog[i] out what),
        }
        fprintf(p, "e\n"), /* end of data */
        fflush(p),
    }

#define DO_CALC(what)
    if (strcmp(cgivars[0], #what) == 0)
    {
        for(i=0,i<now,i++)
        {
            fprintf(p, "%f\n", s->cv[i] what),
        }
        fprintf(p, "e\n"), /* end of data */
        fflush(p),
    }

volatile struct shared_hw_data *s,

/* this opens a pipe with gnuplot and spits out a gif file */
int main(int argc, char *argv[])
{
    char **cgivars,
    FILE *p,
    int i,
    int now,

    /* first open shared mem */
    if ((s = fcllent_open_shm(0_RDONLY)) == NULL)
        die("fcllent_open_shm(0_RDONLY)"),

    now = s->active,

    /* get requested plot name */
    cgivars = getcgivars(), /* returns name value pairs */

    if ((now == 0) || (strcmp(cgivars[0], "none") == 0))
    {
        printf(NOTFOUNDPATH),
        fflush(stdout),
        exit (0),
    }
}

```

```

printf("Content-type image/gif\n\n"),
fflush(stdout),

/* then open a pipe to gnuplot */
p = popen(GNUPLOT_BIN, "w"),

if (p == NULL) die("popen"),

/* start talking to gnuplot */
fprintf(p, "set term gif interlace transparent small size %d,%d xb0b0b0\n",
        WIDTH, HEIGHT),
fprintf(p, "plot '-' title '%s' with lines\n", cgivars[0]),

DO_ANALOG_IN(1EmMotorSpeed)      else
DO_ANALOG_IN(1GenMotorSpeed)     else
DO_ANALOG_IN(1IceEngineSpeed)    else
DO_ANALOG_IN(1AccelPedalLevel)   else
DO_ANALOG_IN(1BrakePedalLevel)   else
DO_ANALOG_IN(1ActualEmTorque)    else
DO_ANALOG_IN(1FuelPressure)      else
DO_ANALOG_IN(1EmCurrent)         else
DO_ANALOG_IN(1GenCurrent)        else
DO_ANALOG_IN(1BattPackTemp)      else
DO_ANALOG_IN(1BattPackVoltage)   else
DO_ANALOG_IN(1BattPackCurrent)   else
DO_ANALOG_IN(1EmRotorTemp)       else
DO_ANALOG_IN(1EmInvTemp)         else
DO_ANALOG_IN(1TpsFeedback)       else

DO_ANALOG_OUT(oGenSpeedReq)      else
DO_ANALOG_OUT(oGenRegenLimit)   else
DO_ANALOG_OUT(oEmAccelReq)       else
DO_ANALOG_OUT(oEmBrakeReq)      else
DO_ANALOG_OUT(oIceThrottlePos)  else
DO_ANALOG_OUT(oDisSpeedo0)      else
DO_ANALOG_OUT(oDisSpeedo1)      else
DO_ANALOG_OUT(oPwrToDigRacks)   else

DO_CALC(SOC)                     else
DO_CALC(vehicle_speed)           else
DO_CALC(vehicle_distance)        else
DO_CALC(kwh_volts)               else
DO_CALC(kwh_amps)                else

printf(NOTFOUNDPATH),

pclose(p),

return 0,
}

```


G.4 libcgi: C Language CGI Library

G.4.1 Makefile

```
# requires gnu make

CC          =      cc
AR          =      ar
ARFLAGS    =      qcr
RANLIB     =      true
DEFINES    =
INCLUDES   =
QUIET      =      -Q -wx
OPTS       =      -Ora1nextm -4r -fp3 -fp187
DEBUG      =      #-g
CFLAGS     =      $(QUIET) $(DEBUG) $(OPTS) $(DEFINES) $(INCLUDES)
FILES      =      get_cgi.c
OBJS       =      $(FILES) c= o)
LIBOUT     =      libcgi.a
DEPEND     =      makedepend
LIBDIR     =      /opt/fcar/lib/

all        $(LIBOUT)

$(LIBOUT)  $(OBJS)
           $(AR) $(ARFLAGS) $@ $^
           $(RANLIB) $@

dep
           $(DEPEND) -- $(CFLAGS) -D__QNX__ -- $(FILES)

install    $(LIBOUT)
           cp -f $(LIBOUT) $(LIBDIR)

clean
           rm -f $(OBJS) $(LIBOUT) core * err
```

G.4.2 cgi.h

```
/* cgi.h */

#ifndef CGI_INCLUDED
#define CGI_INCLUDED

#ifndef __cplusplus
extern "C" {
#endif

extern char **getcgivars(void),

#ifndef __cplusplus
}
#endif

#endif
```

G.4.3 get_cgi.c

```
/* get_cgi.c */
```

```

/*
 *   getcgivars c-- routine to read CGI input variables into an
 *   array of strings
 *
 *   Written in 1996 by James Marshall, james@jmarshall.com, except
 *   that the x2c() and unescape_url() routines were lifted directly
 *   from NCSA's sample program util.c, packaged with their HTTPD
 *
 *   For the latest, see http://www.jmarshall.com/easy/cgi/
 *   "CGI Made Really Easy"
 */

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include "cgi.h"

/* Convert a two-char hex string into the char it represents */
char x2c(char *what)
{
    register char digit,
        digit=(what[0] >= 'A' ? ((what[0] & 0xdf) - 'A')+10 : (what[0] - '0')),
        digit*=16,
        digit+=(what[1]>= 'A' ? ((what[1] & 0xdf) - 'A')+10 : (what[1] - '0')),
        return(digit),
}

/* Reduce any %xx escape sequences to the characters they represent */
void unescape_url(char *url)
{
    register int i,j,

    for(i=0,j=0, url[j], ++i,++j)
    {
        if((url[i] = url[j]) == '%')
        {
            url[i] = x2c(&url[j+1]),
            j+= 2,
        }
        url[i] = '\0',
    }
}

/* Read the CGI input and place all name/val pairs into list
 * Returns list containing name1, value1, name2, value2, ..., NULL */
char **getcgivars()
{
    register int i,
        char *request_method,
        int content_length,
        char *cginput,
        char **cgivars,
        char **pairlist,
        int paircount,
        char *nvpair,
        char *eqpos,

    /* Depending on the request method, read all CGI input into cginput

```

```

    * (really should produce HTML error messages, instead of exit()ing) */
    request_method= getenv("REQUEST_METHOD"),
    if ('strcmp(request_method, "GET") || 'strcmp(request_method, "HEAD") )
    {
        cginput= strdup(getenv("QUERY_STRING")),
    }
    else if ('strcmp(request_method, "POST"))
    {
        if ( strcmp(getenv("CONTENT_TYPE"),
                    "application/x-www-form-urlencoded"))
        {
            printf("getcgivars  Unsupported Content-Type \n"),
            exit(1),
        }
        if ( !(content_length = atoi(getenv("CONTENT_LENGTH"))) )
        {
            printf(
                "getcgivars  No Content-Length was sent with the "
                "POST request \n"),
            exit(1),
        }
        if ( !(cginput= (char *) malloc(content_length+1)) )
        {
            printf("getcgivars  Could not malloc for cginput \n"),
            exit(1),
        }
        if ('fread(cginput, content_length, 1, stdin))
        {
            printf("getcgivars  Couldn't read CGI "
                    "input from stdin \n"),
            exit(1) ,
        }
        cginput[content_length]='\0',
    }
    else
    {
        printf("getcgivars  unsupported REQUEST_METHOD\n"),
        exit(1),
    }

    /* Change all plusses back to spaces */
    for(i=0, cginput[i], i++) if (cginput[i] == '+') cginput[i] = ' ',

    /* First, split on "&" to extract the name-value pairs into pairlist */
    pairlist= (char **) malloc(256*sizeof(char **)),
    paircount= 0,
    nvpair= strtok(cginput, "&"),

    while (nvpair)
    {
        pairlist[paircount++]= strdup(nvpair),
        if (!(paircount%256))
            pairlist= (char **) realloc(pairlist,
                                        (paircount+256)*sizeof(char **)),
            nvpair= strtok(NULL, "&"),
    }
    pairlist[paircount]= 0 ,    /* terminate the list with NULL */

    /* Then, from the list of pairs, extract the names and values */
    cgivars= (char **) malloc((paircount*2+1)*sizeof(char **)),
    for (i= 0, i<paircount, i++)

```

```
{
    if (eqpos=strchr(pairlist[i], '='))
    {
        *eqpos= '\0',
        unescape_url(cgivars[i*2+1]= strdup(eqpos+1)),
    }
    else
    {
        unescape_url(cgivars[i*2+1]= strdup("")),
    }
    unescape_url(cgivars[i*2]= strdup(pairlist[i])),
}
cgivars[paircount*2]= 0, /* terminate the list with NULL */

/* Free anything that needs to be freed */
free(cgiinput),
for (i=0, pairlist[i], i++) free(pairlist[i]),
    free(pairlist),

/* Return the list of name-value strings */
return cgivars,
}
```

Vita

Matthew D Smith was born in Dallas, Texas, on April 11, 1975. His family moved early in his childhood to Northeast Tennessee where he attended public schools. In 1993 he graduated from University High School and entered the University of Tennessee, Knoxville. After four years of undergraduate work and one year of co-op work, he received a Bachelor of Science degree in Mechanical Engineering in 1998. Continuing on in the Mechanical and Aerospace Engineering and Engineering Science department at UTK culminated in a Master of Science degree in Mechanical Engineering in 2000.