Safety and Reliability of Distributed Embedded Systems

Technical Report *ESL 04-03*

Development of a Hardware-in-the-Loop Test Facility for Distributed Embedded Systems

Michael Short
Michael J. Pont
and
Qiang Huang

Embedded Systems Laboratory
University of Leicester

[ESL04-03A - 11 OCTOBER 2004]

Project summary

This technical report is one of a series (listed in full below). Together these reports describe a complete hardware-in-the-loop (HIL) simulation that reproduces the behaviour of a passenger car travelling down a motorway. In the simulation, the speed and position of the car are determined by an adaptive cruise control system implemented using one or more embedded microcontrollers.

The test bed is intended to be used to assess and compare different software architectures for use in distributed embedded systems, particularly those for which high reliability is a key design consideration.

Full list of reports in this series

Available now:

ESL04/01 "Simulation of Vehicle Longitudinal Dynamics"

ESL04/02 "Simulation of Motorway Traffic Flows"

ESL04/03 "Development of a Hardware-in-the-Loop Test Facility for Automotive ACC Implementations"

Forthcoming:

ESL04/04 "Control Technologies For Automotive Drive-By-Wire Applications"

ESL04/05 "10-Node Distributed ACC System: Co-Operative Implementation"

ESL04/06 "10-Node Distributed ACC System: Pre-Emptive Implementation"

Acknowledgements

The work described in this report was supported by the Leverhulme Trust (F / 00212 / D)

Contents

1. Introduction 1
  1.1 Test-bed Principle 1
  1.2 Scenario Files 2
  1.3 Data Gathering 4
2. Hardware Interfacing 5
  2.1 Digital I/O Assignment 6
  2.2 DAC Interface 9
3. Timing Resources 12
  3.1 DOS Scheduler 12
4. Virtual Sensors/Actuators 14
  4.1 Sensor Physical Description 14
  4.2 Dynamic Model 15
  4.3 Sensor Noise Model 17
5. Software Design 18
  5.1 Software Specification 18
  5.2 Achieving a stable real-time simulation 20
  5.3 Software Architecture 20
6. Conclusions 24
7. References / Bibliography 25

1. Introduction

This report describes the development of a PC-based, Hardware-In-the-Loop (HIL) test facility for the evaluation of embedded automotive control system designs.

The report begins by explaining the principles of Adaptive Cruise Control (ACC) and exploring how the work reported in previous technical reports in this series is integrated into the overall simulation. Following this, the hardware interface to the embedded system is described, along with the techniques that were used to generate a suitable timing resource for implementation in a DOS-based environment. The next section details how the various system sensor and actuators may be simulated in a realistic manner. The report then concludes with a description of a suitable software architecture, bringing together each of the required elements. The simulation is coded entirely in C, with reference to certain functions made in the text.

Further information regarding the structure of the control systems may be found in technical report *ESL04/04*, and details of specific implementation architectures will follow in subsequent reports.

1.1 Test-bed Principle

This section describes the overall principle of the test-bed, and how it is intended to be used. The embedded system is to be used to control the throttle and brakes of a simulated vehicle, travelling down a motorway. A host PC is to implement all the required equations of motion, and full motorway simulation, in real-time. The host PC will generate sensor signals for, and read in actuator signals from, the embedded control system. In order to be able to assess the efficiency of the embedded system design, the host PC must both be able to log data at arbitrary time intervals regarding the state of the simulation, and control the simulation scenario. Figure 2 shows an overall schematic of the test facility:

In order to ensure that the test facility is realistic, a complex non-linear model is used to represent the host vehicle as it travels down the motorway. This model includes concepts such as wheel slip, thus requiring the embedded system to implement ABS and traction control algorithms in addition to the main system, an Adaptive Cruise Control system (ACC). This host vehicle will be integrated with an intelligent driver, who will enable/disable the ACC system and decide when to change lanes etc. The simulation will be controlled by a scenario file, which may be loaded or created before the simulation starts. The following section describes the structure of these files.

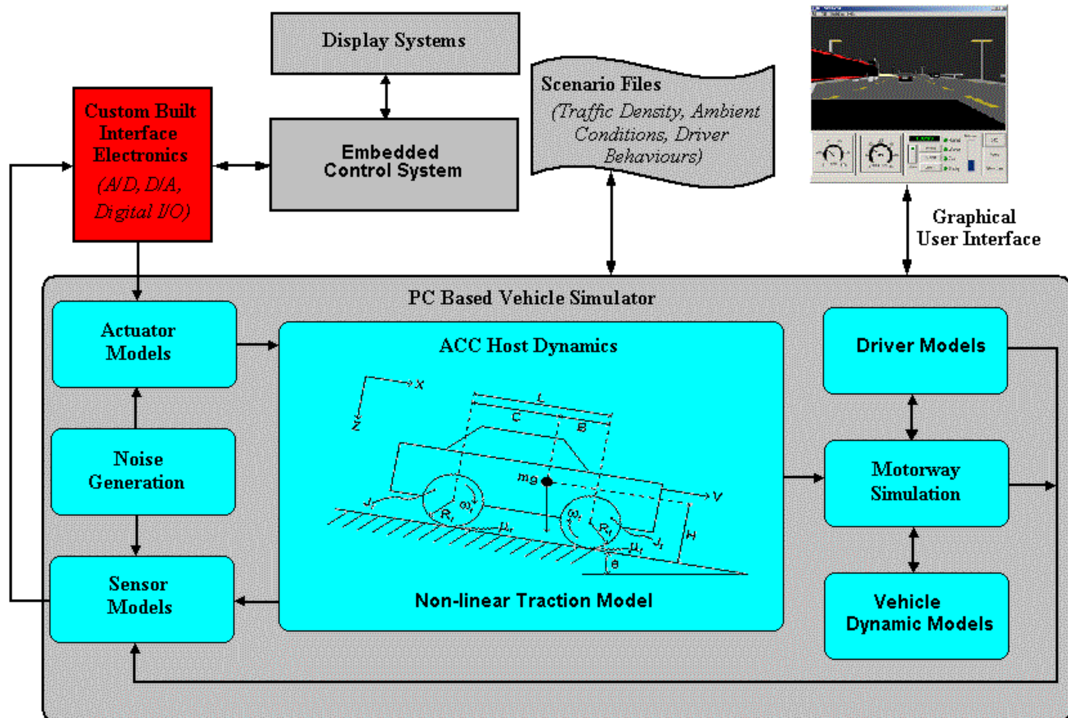

Figure 1: Test Facility Elements

1.2 Scenario Files

This scenario file contains both initialisation and run-time information. The following information is required at initialisation:

- Traffic density
- Passenger vehicle speed limit and standard deviation.
- Percentage of HGV traffic.
- HGV speed limit and standard deviation.
- Host target speed.
- Road conditions/wind speed/gradient.

This information is used to generate the initial simulation conditions and initialise the driver models. Following this, the scenario file holds an arbitrary number of entries that dictate how the simulation is to behave as it progresses. The following elements may be changed after a certain time, or distance, has elapsed:

- Gradient (distance)
- Speed limits (distance)
- Percentage of HGV traffic (time OR distance)
- Traffic density (time OR distance)
- Host target speed (time OR distance)
- Road conditions/wind speed (time)

This scenario structure is used to reflect that changing weather conditions are time dependent, yet speed limits and road gradient are distance dependent. Traffic density varies due to time of day, and also what area of a given motorway is currently occupied [May 1990], so either may alter this. The host target speed may be modified at any time or distance to explore the effect of changing the desired speed to suit any environmental condition, speed limit or traffic flow. The scenario files are simple text files, possessing the structure of figure 3.

::INIT::
TESTNAME: **S** DATE: **S**
DATA: **S**
PATH: **S**
DEN: **I** PSP: **F F** HGV: **F** HSP: **F F** HTS: **F** RCN: **I** WSP: **F** GRD: **F**
::TIME::
TIM: **F** DEN: **I** HGV: **F** HTS: **F** RCN: **I** WSP: **F**
TIM: **F** DEN: **I** HGV: **F** HTS: **F** RCN: **I** WSP: **F**
::END::
::DIST::
DIS: **F** DEN: **I** PSP: **F** HGV: **F** HSP: **F** HTS: **F**
DIS: **F** DEN: **I** PSP: **F** HGV: **F** HSP: **F** HTS: **F**
DIS: **F** DEN: **I** PSP: **F** HGV: **F** HSP: **F** HTS: **F**
::LOOP::
::NOTES::

Figure 3: Scenario file structure

The initial section of the script file allows the scenario to be given a name, and a test date recorded. The bold **S** indicates a string, and the fields are delimited with a space. The next two entries specify a path for data saving, and a string of flags indicating which data is to be saved – see section 1.4. The initialisation data then follows this – the meaning of the fields is explained below:

- DEN (traffic DENsity): **I**nteger (vehicles/km)
- PSP (Passenger vehicle SPeeds): **F**loat (mean Km/hr) **F**loat (std deviation Km/hr)
- HGV (number of HGV's): **F**loat (percentage of traffic)
- HSP (Hgv SPeeds) **F**loat (mean Km/hr) **F**loat (standard deviation Km/hr)
- HTS (Host Target Speed) **F**loat (Km/hr)
- RCN (Road CoNditions) **I**nteger (0=Dry, 1=Wet, 2=Snow, 3=Ice)
- WSP (Wind SPeed) **F**loat (Km/hr)[1]
- GRD (GRaDient) **F**loat (percent incline)

Entering the command NC in a particular entry will result in no change being made to its value. Other values are changed instantaneously, with the exception of the traffic density, which will be altered over a short space of time as cars exit/enter the simulation. The following section

[1] The mean wind speed is specified; during simulation, turbulence of 12% is added via a digitally filtered Gaussian process.

then stores the information regarding the time events for the simulation. The definition of the variables has the same structure as given above, with the addition of a TIMe field, which defines the simulation time duration that the entries are active. The entries are arranged in ascending order, and in order to limit memory use the file is parsed into the simulation line-by-line. If the simulation encounters the ::LOOP:: command, the file pointer relocates to the top of the time sections and begins to loop through once again. If the ::END:: command is encountered, the simulation will terminate.

The section following this then defines the DIStance information entries, which operate in a similar manner to the time entries except that distance duration must be specified instead of time. The simulation will again terminate or loop upon encountering the relevant command, allowing the simulation to loop either for a specified time or distance duration, or until a user stops the simulation manually. The simulation will also terminate if a fault occurs in the embedded system, but only when the vehicle has been brought to rest either successfully or otherwise. The final section of the scenario file allows any user comments to be added. A simple text file editor may be used to create the entries, or a specific tool created for the purpose. The c functions fopen() and fgets() may be used to first open the scenario file, then parse through on a line-by-line basis. By declaring pointers to two individual FILE data types, then opening them with the same scenario file, it is possible to parse through both the time and distance entries simultaneously. Section 1.4 describes how the simulation data may be recorded to hard disk.

1.3 Data Gathering

A very similar methodology to that which was outlined in the previous section was utilised to write data to the hard disk. The data that may be recorded include simulation inputs, simulation outputs, and simulation variables such as distance to lead car/traffic flow etc. Before the simulation begins execution, a pointer to a FILE data type is declared and opened with the required name and path (which is read form the scenario file) using the fopen() function. At the required intervals, the output data is first written into a text buffer and then appended to the text file using the fputs() command. In this way, large amounts of data may be logged directly to disk without using large amounts of memory for temporary storage. Upon simulation termination, the fclose() command is used on the output file (and the scenario file) to close the files properly. The following figure shows a list of entries for the DATA string that controls what data to record, and the format that is saved into the data file:

VEL: Host vehicle linear VELocity (float)
FLV: Front Left wheel Velocity (any wheel may be specified; eg Rear Right would be RRV) (float)
FLS: Front Left wheel Slip (any wheel may be specified; eg Rear Right would be RRS) (float)
FLB: Front Left Brake Setting (any wheel may be specified; eg Rear Right would be RRB) (float)
THR: THRottle setting (float)
DTL: Distance To Lead vehicle (float)
LRV: Lead Relative Velocity (float)
STA: STAtus (byte)
FLW: traffic FloW (float)
DEN: traffic DENsity (int)
DBS: Driver Brake Setting (float)
DTS: Driver Throttle Setting (float)

Figure 4: Data String Format

2. Hardware Interfacing

This section describes the methodology that was utilised to connect the PC based simulation to the embedded system itself. To keep the cost to a minimum, it was decided to implement this interface entirely through the use of several parallel ports. Due to the nature of the signals that have to be generated by the PC, and read from the embedded system, a total of four ports were required, utilising a combination of analog and digital signals.

Since 1981, the majority of PC's have had a 25-pin D-type connector on the rear panel. This provides a connection to the parallel port, sometimes known as the printer port, and has been used mainly to connect a printer using the Centronics interface protocol. A cable with a Centronics plug is normally used, the connections of which are shown in figure 5:

Pin No.	Pin Name	Register/Bit	Input/Output
1	!STR	Control 0	Input/Output
2	D0	Data 0	Input/Output
3	D1	Data 1	Input/Output
4	D2	Data 2	Input/Output
5	D3	Data 3	Input/Output
6	D4	Data 4	Input/Output
7	D5	Data 5	Input/Output
8	D6	Data 6	Input/Output
9	D7	Data 7	Input/Output
10	!ACK	Status 6	Input
11	BSY	Status 7	Input
12	PAP	Status 5	Input
13	ONOF	Status 4	Input
14	!ALF	Control 1	Input/Output
15,16	GND	N/A	N/A
17	Case	N/A	N/A
18	5v	N/A	N/A
19-30	GND	N/A	N/A
31	INI	Control 2	Input/Output
32	!ERR	Status 3	Input
33	Gnd	N/A	N/A
34,35	Unused	N/A	N/A
36	!DSL	Control 3	Input/Output

Figure 5: Centronics Pin-Out

Of the 36 pins in total, 17 may be used for I/O. These pins are either written to or read from using interface registers on the PC, named DATA, STATUS and CONTROL. The address of these byte registers is generated as an offset from the main port address, with DATA having offset 0x0, STATUS 0x01 and CONTROL 0x02. The register and corresponding bit for each pin is shown in the third column in figure 5. The remaining register bits serve no purpose, except for bit 4 of the control register (the IRQ bit), which is not used in this implementation, and bit 5 of the same register whose use is described in the next section. The relationship between the pin

value and register value is inverted in five cases; these are indicated by the use of logical not (!) before the pin name in figure 5.

Most standard PC's may possess up to 4 parallel ports (LPT's) without the need for special software drivers. In practice, however they commonly only equipped with the hardware for a single port with address 0x378. In this implementation, four LPT's are used with addresses given below:

LPT1: 0x378
LPT2: 0x278
LPT3: 0x288
LPT4: 0x280

The additional ports were obtained at a low cost and purchased from Microcomputer Research Ltd. [Microcomputer Research 2004]. The DOS operating system allows calls from an application program to either directly write to, or read from the PC bus (typically inp() and outp() and their variants). Thus an application program, such as the simulation described in this report, may communicate with external devices via the output pins. The following section describes the digital I/O assignment of the test bed.

2.1 Digital I/O Assignment

The first three parallel ports are used solely for digital inputs/outputs to the embedded control system. The fourth port is mainly used to generate analog outputs, with several pins reserved for digital I/O. The LPT data port is commonly used as an output port: however it may be switched into open-collector mode by setting bit 5 of the port's control register to logic 1 [Messmer 2002]. The main input signals to the PC are the throttle setting, and the four brake settings. The throttle setting is implemented as a 10-bit signal, whilst each brake setting is achieved by an 8-bit signal. In addition to these, there are number of control and status signals to be interfaced. The full digital I/O assignment for each LPT is given in Figure 6, Figure 7, Figure 8 and Figure 9.

Pin	Purpose	I/O
1	Cruise Enable	Output
2	Throttle Setting LSB	Input
3	Throttle Setting	Input
4	Throttle Setting	Input
5	Throttle Setting	Input
6	Throttle Setting	Input
7	Throttle Setting	Input
8	Throttle Setting	Input
9	Throttle Setting	Input
10	Throttle Setting	Input
11	Throttle Setting MSB	Input
12	CCS Running	Input
13	CCS Error	Input
14	Speed -5 MPH	Output
31	Speed +5 MPH	Output
32	CCS Warning	Input
36	Cruise Disable	Output

Figure 6: LPT1 Digital I/O Assignment

Pin	Purpose	I/O
1	FR Brake Setting LSB	Input
2	FL Brake Setting LSB	Input
3	FL Brake Setting	Input
4	FL Brake Setting	Input
5	FL Brake Setting	Input
6	FL Brake Setting	Input
7	FL Brake Setting	Input
8	FL Brake Setting	Input
9	FL Brake Setting MSB	Input
10	FR Brake Setting	Input
11	FR Brake Setting MSB	Input
12	FR Brake Setting	Input
13	FR Brake Setting	Input
14	FR Brake Setting	Input
31	FR Brake Setting	Input
32	FR Brake Setting	Input
36	N/C	N/A

Figure 7: LPT2 Digital Pin Assignment

Pin	Purpose	I/O
1	RR Brake Setting LSB	Input
2	RL Brake Setting LSB	Input
3	RL Brake Setting	Input
4	RL Brake Setting	Input
5	RL Brake Setting	Input
6	RL Brake Setting	Input
7	RL Brake Setting	Input
8	RL Brake Setting	Input
9	RL Brake Setting MSB	Input
10	RR Brake Setting	Input
11	RR Brake Setting MSB	Input
12	RR Brake Setting	Input
13	RR Brake Setting	Input
14	RR Brake Setting	Input
31	RR Brake Setting	Input
32	RR Brake Setting	Input
36	N/C	N/A

Figure 8: LPT3 Digital Pin Assignment

Pin	Purpose	I/O
10	N/C	Input
11	N/C	Input
12	N/C	Input
13	N/C	Input
31	Cruise Resume	Output
32	N/C	Input
36	ACC Target Detected	Output

Figure 9: LPT4 Digital Pin Assignment

A description of each of the ACC status and control signals is now given. The Cruise Enable signal is used by the PC simulation to indicate if the virtual driver has enabled the ACC system or not. Similarly, Cruise Disable enables the virtual driver to disable the ACC system. The ACC Resume signal is used to enable ACC at the previous set speed (enable sets the current speed as the set speed). The speed +5 MPH and –5 MPH signals enable the driver to adjust the ACC set speed, and are only active when the system is enabled. The ACC Target Detected signal from the PC indicates that a valid lead vehicle is being tracked. CCS Running, CCS Warning and CCS Error are inputs to the simulation from the embedded system. They indicate the ACC status, the CCS Running signal indicates that the CCS is enabled. CCS Warning indicates that a non-fatal error has been detected (‘limp-home’ mode) and the driver should seek assistance ASAP. CCS Error indicates that a fatal error has occurred, the system cannot operate and the driver should take emergency action. Many of the required PC outputs, for example ACC distance, require an

analog signal. The following section describes an interface that enables these signals to be generated via the parallel port LPT4.

2.2 DAC Interface

In order to allow the simulator to generate analogue voltage signals for the embedded system, a series of DAC devices were interfaced to the fourth parallel port. The devices themselves were high speed, 12-bit, low power dual channel converters obtained from Analog Devices. Figure 10 shows the analog signals that the simulator is required to generate:

Signal	Purpose
1	Linear Vehicle Speed
2	ACC Target Distance
3	ACC Target Relative Velocity
4	Driver Throttle Setting
5	Driver Brake Setting
6	Front Left Wheel Speed
7	Front Right Wheel Speed
8	Rear Left Wheel Speed
9	Rear Right Wheel Speed

Figure 10: Analog Signal Requirements

The signals are described fully in section 4.1. Since each DAC device is dual channel, five devices were required to synthesise the signals. The recommended wiring of each individual chip is as shown in figure 11 [Analog Devices 1998]:

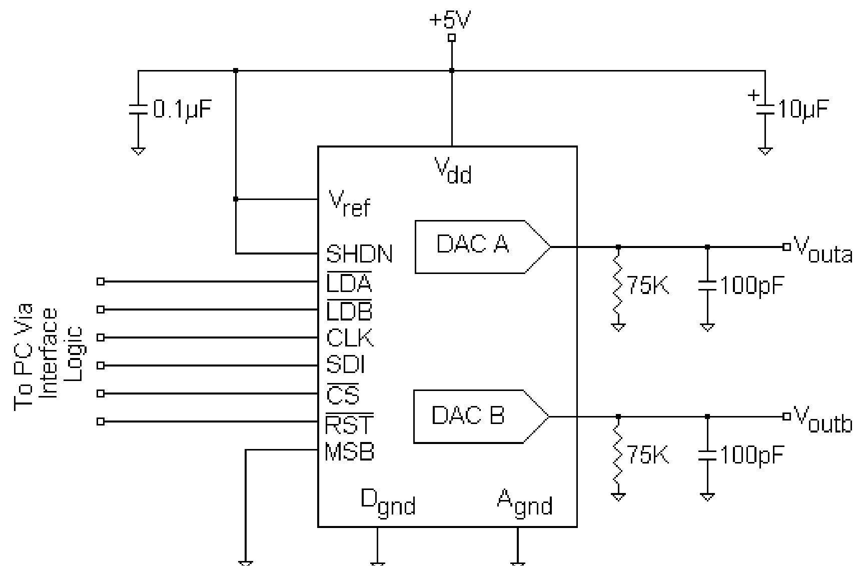

Figure 11: DAC Wiring

The SDI (Serial Data In) and CLK (ClocK) pin enable a 12-bit word to be clocked into the device shift register MSB first, whilst strobing the LDA or LDB (LoaD A/B) pin performs the D/A conversion on the corresponding channel. The CS (Chip Select) pin must be pulled to logic zero in order to enable the device shift register. Pulling the RST (ReSeT) pin to a logic zero causes both channels to output zero volts. The SHDN and MSB pins are for configuration purposes and weree hardwired as shown. Each device was hardwired according to this diagram. In order to reduce the number of pins required for the PC interface, use was made of the CS lines of each chip. However, the CS line only affects the shift register and does not affect the LDA and LDB pins, so a small amount of interface logic was required. This was achieved with three quad, low cost, high-speed CMOS OR gates manufactured by Fairchild Semiconductor [Fairchild 1999]. The interface logic was connected as follows:

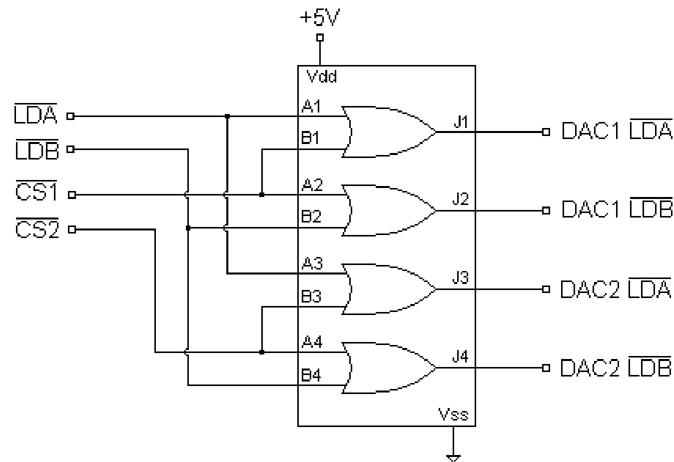

Figure 12: Interface Logic

From the figure it can be seen that the LDA and LDB lines are now only active for each individual chip when the corresponding CS line is pulled low. The CS line is also connected to the corresponding DAC. A similar methodology was applied to the remaining DAC chips, which are not shown in figure 12. The CLK, SDI and RST outputs from the PC were connected to each chip in parallel. Thus, to perform a 12-bit conversion on a particular output, the following algorithm may be used:

1. Pull the required CS line low.
2. Set the SDI line to the required value.
3. Strobe the CLK line (low-high-low).
4. Repeat steps 2 and 3 until the required 12 bits have been clocked in.
5. Strobe the required load line (high-low-high) (LDA or LDB).
6. Pull the CS line high.

All outputs may be reset by simply strobing the RST pin (high-low-high). The PC outputs required to interface to this circuit were assigned as shown in figure 13, using parallel port 4 (0x280).

Pin	Name	Function
1	!STR	RST
2	D0	CLK
3	D1	SDI
4	D2	LDA
5	D3	LDB
6	D4	CS1
7	D5	CS2
8	D6	CS3
9	D7	CS4
14	!ALF	CS5

Figure 13: Analog Interface Pins

For both the analog and digital interfaces, the actual connections to the LPT interfaces were achieved using 25-way D connector to 36-way Centronics cables, with push-connect terminals soldered to prototype board containing the required circuitry inside a small enclosure. This enables rapid-reconfiguration should the I/O requirements of future system implementations change. The following section describes the creation of a suitable timing resource to implement the simulation.

3. Timing Resources

This section describes the development of a suitable timing resource that may be used in a DOS environment. This timing resource is needed to implement a method to solve the vehicle equations of motion and motorway simulation in real-time. A simple method of doing this is by creating a DOS scheduler. A scheduler may be viewed as a simple operating system that allows tasks to be called periodically, using a single timer-driven interrupt service routine. The scheduler used in this implementation is co-operative: it provides a single-tasking architecture. For a comparison of other types of scheduler, refer to, for example [Pont 2001]. Further information regarding this type of scheduler is available in [Pont et al. 2003], and the following section describes its key operation.

3.1 DOS Scheduler

In this implementation the scheduler lies on top of the DOS operating system, and its creation is quite straightforward since DOS programs are allowed access to the underlying hardware. DOS itself lays above the system BIOS (Basic Input Output System), firmware that enables the system to run an initial power-on self-test and initialization, load the boot program from the boot disk, and handle low-level I/O to peripheral controllers such as keyboard and display. Figure 14 shows the overall architecture.

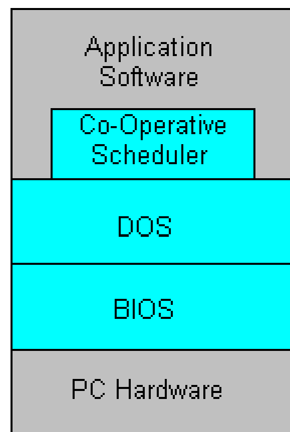

Figure 14: Scheduler Architecture

When developing a scheduler for a PC platform, the main issue that must be considered is a source of stable timer 'ticks'. In the PC architecture, the system timer runs at 1.193 MHz. From this, DOS uses a system tick of approximately 55 ms to update the system clock and perform certain BIOS functions. This system tick arises as follows:

- An external (crystal) oscillator provides a square wave of frequency 1.19318 MHz.
- This square wave is fed into the Counter 0 channel of an 8253 programmable interval timer (PIT).

- The PIT (Channel 0) is loaded with a "divider" value of 0x00 (zero defaults to 65536), which means that it generates an output square wave of 18.206 Hz (= 1193180 / 65536).
- The 18.206 Hz square wave is fed, via an 8259A programmable interrupt controller (PIC), into the IR0 interrupt pin on the 80x86 (or equivalent) processor at the heart of the PC.
- The IR0 input gives rise to an interrupt number 0x08. This in turn (explicitly) calls the 0x1C "user" interrupt.

It is possible to change the PIT divider settings to achieve a different tick rate, and install a user-written interrupt service routine to the 0x1C source. The original DOS ISR may then be chained, to call it at a rate of 55ms, to ensure normal operation of the DOS behavior of the system. A suitable tick interval for the scheduler in this system is 1 ms, giving a suitable divider value for the PIT chip as:

$$DIV = \frac{1193180}{1000} = 1193.18$$

Equation 04/03/B

The nearest integer value for this divider is 0x04A9, actually giving a tick interval of 0.0009998 – this is more than accurate for our purposes. To load the new PIT divider, data can be written to the timer via the PC bus (address 0x43 for the control register, 0x40 for the timer 0 divider value). To write a new divider value, the control register must first be loaded with a value of 0x34 (counter 0, rate generator mode, count in binary, write least/most significant bytes). Following this, the data bytes can then be written to 0x40 in the correct order (low-high). Again, the DOS functions outp() may be used for this purpose.

The user-written interrupt handler that is to be installed in this implementation acts as the scheduler update function, and also calls the original handler ever 55 ticks of the timer to maintain normal function. The _dos_setvect() and _dos_getvect() functions (or their equivalent) may be used to retrieve the old interrupt handler and install the new one respectively. When the scheduler terminates, these functions are also used to re-install the original handler. Additionally, the PIT value must be reset to zero if the original interrupt handler is to be reused. Section 5 further discusses the architecture of the various software elements of the system. The following section describes how the various sensors in the system are to be modelled, and how their values are to be transferred, via the hardware interface, to the embedded system.

4. Virtual Sensors/Actuators

This section describes the methodology that was utilised to develop models of the virtual sensors and actuators in the system. Each sensor and actuator is represented by a simple dynamic model, which is detailed below. The development of suitable actuator models has been previously discussed[2], however they may be implemented using the discrete dynamic model discussed in section 4.2. The simulation sensors have additional measurement noise added to their outputs by a pseudo-random number generator before being output. The overall methodology for implementing each 'virtual' sensor can be represented by figure 15 below:

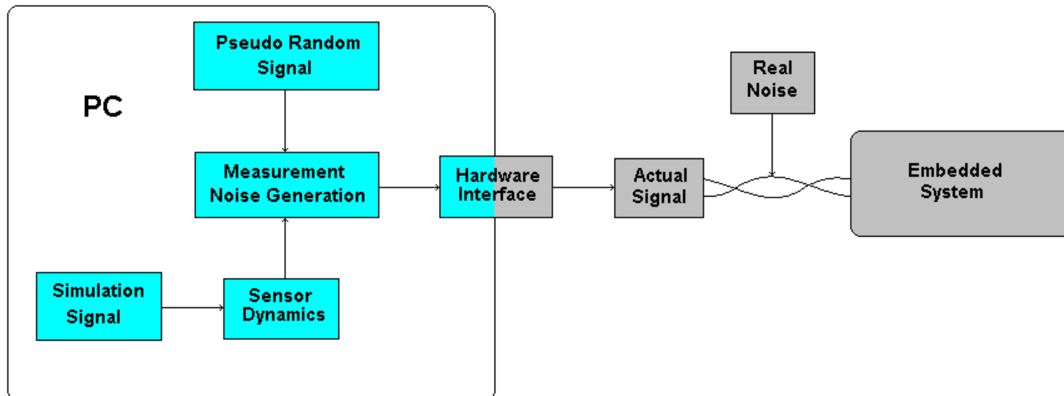

Figure 15: 'Virtual' Sensor Model

Before describing the dynamic model of the sensors, their physical construction and measurement technology will be described.

4.1 Sensor Physical Description

From the table of figure 10, it can be seen that there are effectively nine sensor signals to be generated, from four different types of sensor. The first sensor to be considered is the linear vehicle speed sensor, or speed-over ground sensor. Traditionally, sensors such as these have not featured in vehicle instrumentation; initially because the technology was not feasible in earlier years, and then as prototypes emerged the cost prohibited their use. Measuring the rotational velocity of each wheel is sufficient for speedometer readouts, but the accurate measurement slip is vital for an ABS or traction control algorithm. In the past, automotive manufacturers have favoured the use of accelerometers and complex sensor fusion algorithms such as Kalman filtering to estimate wheel slippage, however problems can arise due to drift and noise immunity issues associated with the accelerometers. Research has lead to the availability of (comparatively) low-cost, compact ground speed sensors using Doppler radar, ultrasound or optical technology [Datron 2004; Imou et al. 2001; Corrsys-Datron 2003], and these are increasingly being utilised in modern vehicles running complex stability and control algorithms. These relatively small sensors are designed to be mounted either under the front or rear of the vehicle, and use the Doppler effect to measure velocity. On-board signal processors then process this frequency shift and generate a filtered analog (or serial data) signal representing the

[2] Please see report *ESL 04/01* for details.

measured velocity. Figure 16 shows the physical location of this, and other, sensors in the vehicle.

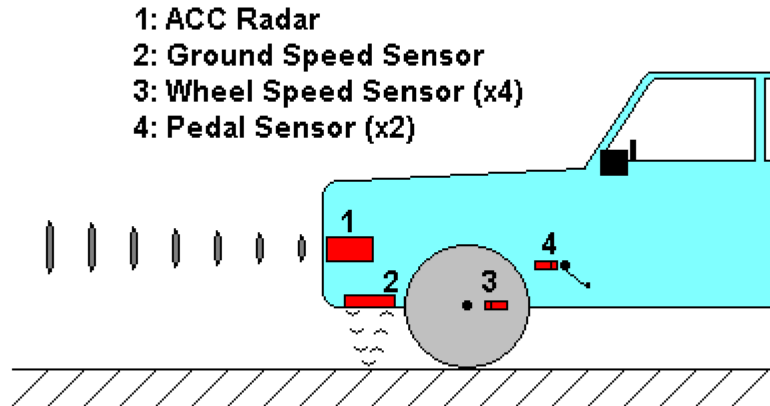

Figure 16: Sensor Physical Locations

The ACC sensor itself is typically realised by a 76-77 GHz radar, implemented with GaAs based MMIC (Monolithic Microwave Integrated Circuits) technology [Langheim et al. 1999]. The pulse travel time is used to measure the distance to the target, and the Doppler shift between sent and received pulses is used to measure the target relative velocity. The unit is mounted behind the front grille of the vehicle, and has a range of approx. 150 m. Typically, they have a long-range beam width of ≈12 degrees, with a 'short range' beam width of ≈70 degrees to detect cut-in situations more effectively. Signal processing is carried out by an on-board signal processor, giving analog/digital outputs or a serial data link (or a combination of both).

The wheel speed sensors act as high precision, non-contact digital tachometers, commonly using either optical or Hall effect type technologies to measure the rate at which a ferromagnetic or reflective toothed-wheel (which is attached to the drive axle) is rotating [Jaquet 2003]. Again it is common for the on-board signal processors to filter the pulse train signal and generate a proportional, high precision analog signal as output. Finally, the accelerator and brake pedal position sensors are implemented as encapsulated potentiometers attached to the axis of the pedal shaft. They typically exhibit a resistance range of 0-5 KΩ from minimum to maximum pedal travel range [Turner & Austin 2000]. A small excitation/amplification circuit allows these potentiometers to generate a 0-5 V signal suitable for a microcontroller interface. The next section discusses a suitable dynamic model of a sensor in general, and how it may be implemented.

4.2 Dynamic Model

In order to represent both the sensors and actuators in the system, a simplified (yet valid) dynamic representation that may be taken is to assume that each sensor may be modelled by a first-order lag, with gain K_s and time constant τ_s [Turner & Austin 2000]. If the sensor input (simulation variable) is X and the sensor output is Y, this may be represented by the continuous equation as shown below:

$$Y = K_s X - \tau_s \dot{Y}$$

Equation 04/03/C

A simple discrete-time representation of this relationship is given in equation D below, where i is the sample number:

$$Y_i = \alpha X_{i-1} + \beta Y_{i-1}$$

Equation 04/03/D

For each sensor the co-efficients α and β are related to the time constant τ_s and the gain K_s via the following relationships:

$$\alpha = K_s \left(\frac{1}{1 + \left(\frac{\tau_s}{T} \right)} \right)$$

Equation 04/03/E

$$\beta = \frac{\left(\frac{\tau_s}{T} \right)}{\left(\frac{\tau_s}{T} \right) + 1}$$

Equation 04/03/F

Where, in both cases, T is the sampling time. Equation D may be implemented in software, and called at a specified rate using the scheduler to achieve the correct timing. Since the co-efficients α and β for each sensor require knowledge of the sensor characteristics K_s and τ_s, these have been determined from manufacturers data and survey sources and are shown in the figure below [Datron 2003; Turner & Austin 2000; Langheim et al. 1999; Jaquet 2003]:

Sensor	Units	Range	Ks	Ts
Linear Vehicle Speed	Km/h	0-250	0.02	0.025
ACC Target Distance	m	0-150	0.0333	0.065
ACC Target Relative Velocity	Km/h	-55/+55	0.0455	0.065
Drivers Throttle Setting (Potentiometer)	Percent	0-100	0.05	0
Driver Brake Setting (Potentiometer)	Percent	0-100	0.05	0
Front Left Wheel Speed	Rads/s	0-300	0.0167	0.05
Front Right Wheel Speed	Rads/s	0-300	0.0167	0.05
Rear Left Wheel Speed	Rads/s	0-300	0.0167	0.05
Rear Right Wheel Speed	Rads/s	0-300	0.0167	0.05

Figure 17: Sensor Characteristics

The values for the sensor gains K_s in the above figure assume that each sensor outputs an analog signal between 0-5V. Before a digital to analog conversion may take place, however, a small amount of measurement noise must be added to each sensor. The following section describes how this is achieved.

4.3 Sensor Noise Model

The measurement noise that is added to each sensors output is injected after the evaluation of the sensor dynamics; otherwise the sensor time constant would act as a filter. The amount of noise that is added is dependent on the typical accuracy that is quoted for each sensor by manufacturers and found in survey data. Figure 18 shows some typical quoted accuracy data for the sensors [Datron 2003; Turner & Austin 2000; Langheim et al. 1999; Jaquet 2003]:

Signal	Accuracy	Units
Linear Vehicle Speed	+/- 0.85 + (0.0036*KPH)	Km/h
ACC Target Distance	+/- 0.1	m
ACC Target Relative Velocity	+/- 0.2	Km/h
Drivers Throttle Setting (Potentiometer)	+/- 0.1	Percent
Driver Brake Setting (Potentiometer)	+/- 0.1	Percent
Front Left Wheel Speed	+/- 0.03	Rads/s
Front Right Wheel Speed	+/- 0.03	Rads/s
Rear Left Wheel Speed	+/- 0.03	Rads/s
Rear Right Wheel Speed	+/- 0.03	Rads/s

Figure 18: Sensor Accuracy

It can be seen that most of the sensors are linear in their accuracy range, with the exception of the Linear Vehicle speed, whose accuracy decreases as the measured variable increases. Using these figures, a simple methodology may be implemented to add the noise – a randomly generated integer may cast as a floating-point number with a range between -1 and 1. This number is then multiplied by the accuracy level of the given sensor, and added to the output of the sensor dynamic model. This value may then be output to the embedded system via the DAC interface, as described in section 2.4. The following section of the report describes the overall software architecture of the simulator.

5. Software Design

This section describes how each element of the test-bed, and its associated software, is incorporated into the application software. First, a software specification is presented for both the motorway simulation and the host vehicle dynamics. Following this, the overall software architecture and the modular design approach are then described.

5.1 Software Specification

The main structure of the simulation model consists of a loop over a set of software modules that calculate the required equations, at a specified time advance interval. Figure 19 shows a suitable flow chart to implement the dynamic equations and driver models as described in report *ESL 04/02*. Depending on the accuracy required and the available computing power, a simulation time step of between 1~1000 ms may be suitable. Due to the complexity of the simulation, a time step of 10ms was chosen for this implementation. The dynamic states governing the vehicle dynamics may be integrated using the simple Euler method, where T is the sample time, i.e. 0.010 seconds:

$$y(t) = y(t-1) + Tx(t)$$

Equation 04/03/G

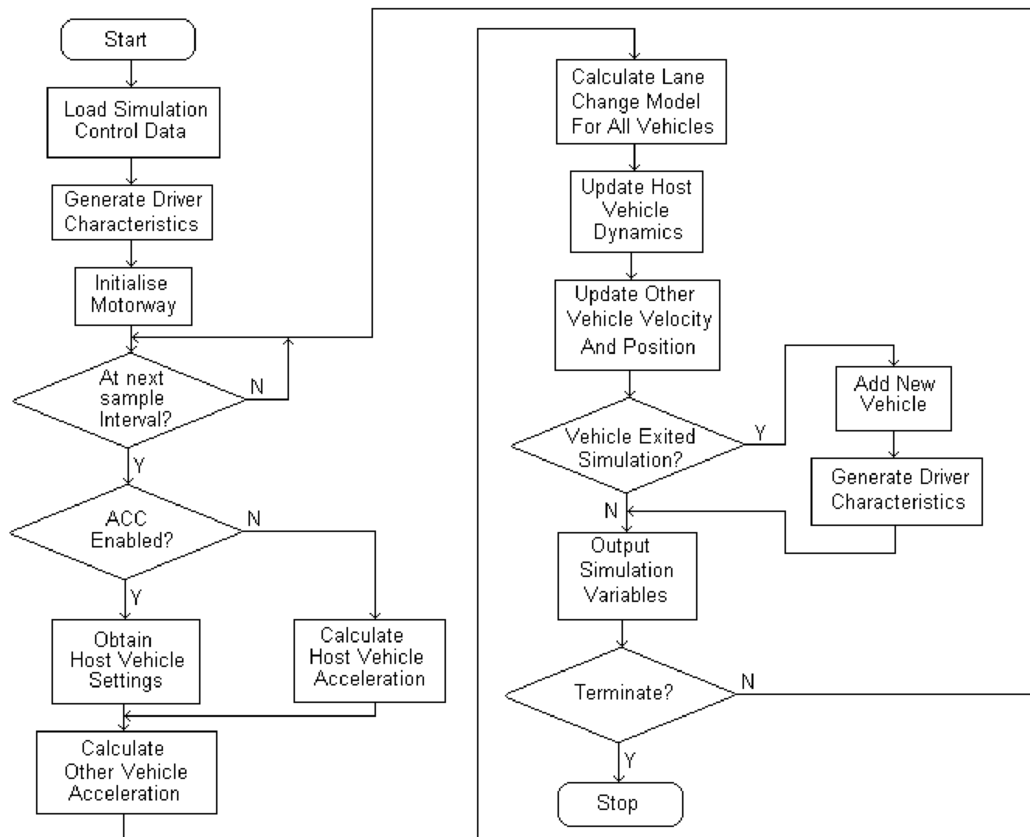

Figure 19: Simulation Flow Chart

Due to the relative complexity of the host vehicle dynamics, as described in report *ESL 0402*, when compared with the additional road user dynamics, a separate flow chart is required for the evaluation of the models. The main structure of the simulation model again consists of a loop over a set of software modules that calculate the required equations, at a specified time advance interval. Figure 20 shows a suitable flow chart. Depending on the accuracy required and the available computing power, a simulation time step of between 1~1000 ms may be suitable. However, a time step of at least 10ms is essential for this implementation to achieve the required operation of the interface to the embedded system. Using this time step of 10ms, the dynamic states may be integrated using the trapezoidal method:

$$y(t) = y(t-1) + \frac{T}{2}\left(x(t) + x(t-1)\right)$$

Equation 04/03/H

Where T is again the sample time, i.e. 0.01 seconds. Applying this equation to each dynamic state, the output y at simulation time t is the state velocity, and the input x is the state acceleration. For larger time steps, a different integration method such as Runge-Cutta should be considered [Press et al. 1992].

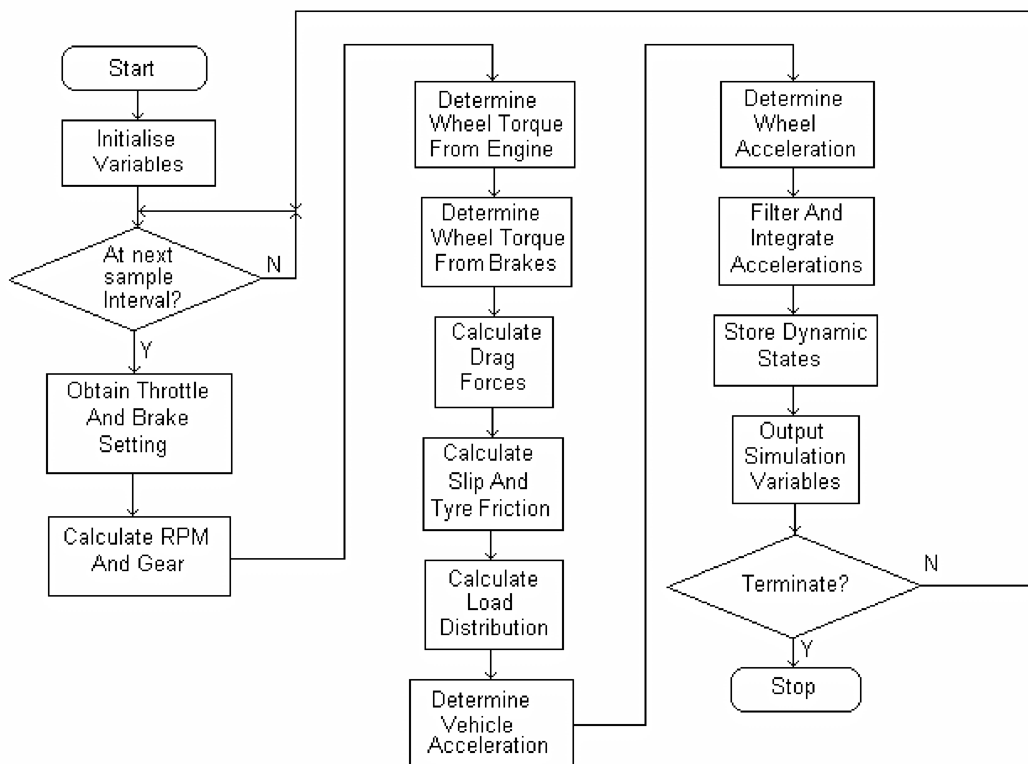

Figure 20: Simulation Flow Chart

Great care must be taken when evaluating the dynamic models that are incorporated into this flow chart, as some of the equations are unstable; the following section described how to achieve a stable simulation.

5.2 Achieving a stable real-time simulation

Due to the tightly coupled and highly non-linear nature of the simulation model equations presented in document *ESL 04/01*, it is imperative that for an effective real time simulation they are solved in the correct order and integrated using a suitable integration method, with an appropriate time step. In addition, observing the equation that defines slip, equation 04/01/C, it can be seen that when both vehicle velocity V and wheel velocity ω are both zero, a singularity occurs. This may be overcome by recognising this situation in software and setting the slip equal to zero. However, when the vehicle speed and wheel speed are both of low value and we approach the singularity, the equations of motion become highly unstable as minute changes to either variable may cause large swings in the sign and magnitude of the calculated slip. In reality, no such oscillatory behaviour occurs due to a variety of effects including static friction, which is dominant at low velocities. A simple solution to this problem (which does not require a stiction model to be incorporated) is suggested in equation I.

$$if\,(V < 1)\,\dot{\omega}_i = \frac{\dot{\omega}_i}{10}, \quad i = r, f$$

Equation 04/03/I

This equation simply acts to attenuate the wheel acceleration by a given factor when the vehicle velocity is below an arbitrary value. These values have been chosen as an attenuation factor of 10 below a cut off value of 1 m/sec^2, which is suitable to stabilise the equations based on the parameters given in report *ESL 04/01*.

5.3 Software Architecture

A modular, file-based approach was taken in the design of the software [Pont 2003]; although this approach is primarily aimed at the embedded systems designs, it will provide a standard environment for the software documentation. All the software was written in ‘C’ and compiled/linked using the open-source, shareware Watcom Compiler[3]. The software was created as a logical series of modules, each implementing a key element of the overall system. Each of these modules, in turn, is comprised of a header file and a main ‘C’ file. The overall architecture is shown in figure 21:

[3] Please see http://www.openwatcom.com for further details.

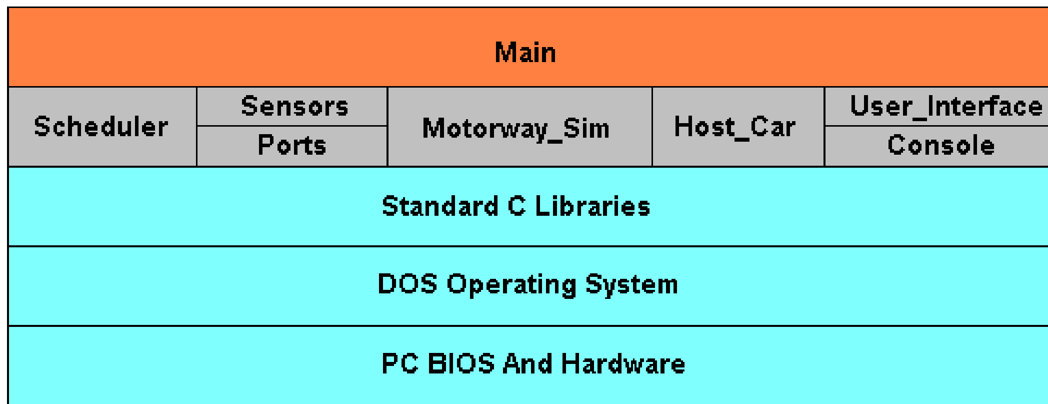

Figure 21: Layered Software Architecture

The figure represents a layered architecture; each module in each layer exports functions and data to the layer immediately above it. The individual modules in a single layer do not export functions and data between themselves, and the size of the modules in each layer do not necessarily represent the code size of the corresponding source files. The lowest three layers, coloured in blue, are existing elements of the system and have not been created, only utilised, in this application. The function of the other elements in the architecture are explained below:

- Main – the main application module. This module contains the main application entry code, and controls the flow of the program and the interaction with the user.
- Scheduler – this module exports functions and data to implement the DOS scheduler.
- Ports – this module contains definitions and functions to provide access to the simulation LPT ports.
- Sensors – this module sits atop the ports module and contains higher-level functionality to implement the sensor/actuator interfaces.
- Motorway_Sim – this module contains all the required functionality and data to implement the simulation of the 3-lane motorway.
- Host_Car – this module implements the non-linear dynamic model of the host vehicle, travelling along the 3-lane motorway.
- Console – this module implements some simple console functions that are not standard with the Watcom ‘C’ libraries, such as clrscr() and gotoxy().
- User_Interface – this module implements a simple user start-up menu, run-time information menu and the data gathering/scenario file interfaces.

Each source module is arranged in a similar manner, exhibiting the following features:

- ‘Public’ functions and constants have their prototypes in the header (H) file.
- ‘Private’ functions and variables have their prototypes statically declared in the C file with the static keyword.
- ‘Private’ constants are declared in the C file.
- ‘Public’ variables are declared in the C file, and are accessed by other modules using the extern keyword.

In addition, each source file is commented such as to explicitly reference individual equations contained in these reports that are implemented in source code. The start-up user interface allows a user three choices – to load a scenario file, to view/edit the current scenario file, or to start the simulation. When viewing/editing a scenario file, the DOS file editor is invoked with the required file address using the system() function. If the simulation is started with no scenario loaded, a test mode is entered with no other cars present on the motorway. This may be used for test and configuration purposes, and the following keys are active during this time:

- ‘q’ – quit and return to menu.
- ‘1’ – set road conditions to dry/normal.
- ‘2’ – set road conditions to wet.
- ‘3’ – set road conditions to snow.
- ‘4’ – set road conditions to ice.
- ‘a’ – increase road gradient.
- ‘z’ – decrease road gradient.
- ‘s’ – increase wind level.
- ‘x’ – decrease wind level.
- ‘e’ – enable ACC.
- ‘d’ – disable ACC.
- ‘r’ – ACC resume.
- ‘t’ – ACC Speed +5 MPH
- ‘g’ – ACC Speed –5 MPH
- ‘l’ – enable/disable lead car.
- ‘k’ – increase lead velocity.
- ‘m’ – decrease lead velocity.

The run-time user interface displays information such as host vehicle position, velocity and wheel speeds; front gap, lead and lag driver and passenger side gaps; traffic flow information; ACC status information and elapsed time. Sample screenshots are shown in figure 22. The user interface is updated every 100ms, while the simulation is updated and the I/O refreshed every 10ms. The full code listings for each module are available in the appendices of this report, and will subsequently be available online (along with other associated code) as the project develops[4].

[4] Please see http://www.le.ac.uk/eg/embedded for details.

```
d:\software\misc\motorway.exe

            ACC Simulation Testbed: Time Elapsed 02 H 43 M 08 S

  Lead Gaps: 1.8 N/A                    Lag Gaps: 3.2 N/A
  Current Flow: 3984 V/HR               Current Density: 70 V/KM
  Lane Change Events: 2452              Occupancy: 25% 35% 40%
  ACC Status: Enabled  No Fault         Current Lane: 3 Y Position: 2.0
  Velocity: 58.2 MPH                    Distance Travelled: 145.23 M
  Front Wheel Velocity: 88.0 RPS        Engine State: 1966 RPM Gear: 5
  Rear Wheel Velocity: 86.5 RPS         Slip: -0.0168 0.0004
  Throttle Setting: 37%                 Friction: -34.9% 0.08%
  Front Brake Settings: 0% 0%           Rear Brake Settings 0% 0%
  Current Target: 1.2 S                 Rel Velocity: 0.1 MPH_
```

```
d:\software\icc_sim2\motorway.exe

                 Motorway Simulator For Embedded HIL Testing
                                Version: 1.4

                            Please Select Option:

                             1: Play Scenario
                             2: View/Edit Scenario
                             3: Load Scenario
                             4: Exit
```

Figure 22: Sample Screenshots

6. Conclusions

This report has discussed the development of a hardware-in-the-loop test facility for automotive ACC designs, implemented by one or more embedded processors. The simulation developed in this report shall be used for the majority of the system testing to be carried out, however for display/presentation purposes a Windows-based, graphical simulation environment has been created. This is detailed in a separate report, *ESL 04/07*. The next report in the series, *ESL 04/04*, is concerned with the operational design of suitable control systems to be implemented by the embedded hardware.

7. References / Bibliography

Analog Devices [1998] *"AD7394 Data Sheet"*, Analog Devices, Norwood, MA, USA. WWW http://www.analog.com

Corrsys-Datron (2003) *"Correvit L200 Optical Sensor Data Sheet"*, Article no. 1.021.00, Corrsys-Datron Technology Ltd., Wetzlar, Germany. WWW http://www.corrsys-datron.com

Datron (2004) *"Delta Speed Sensor DRS1000 Data Sheet"*, Datron Technology Ltd., Milton Keynes, UK. WWW http://www.datrontechnology.co.uk

Fairchild Semiconductor [1999] *"CD4071BC Data Sheet"*, Fairchild Semiconductor. WWW http://www.fairchildsemi.com

Imou, K., Ishida, M., Okamoto, T., Kaizu, Y., Sawamura, A. and Sumida, N. (2001) *"Ultrasonic Doppler Sensor for Measuring Vehicle Speed in Forward and Reverse Motions Including Low Speed Motions"*, CIGR Journal of Scientific Research and Development, Vol. 3, Manuscript PM 01 007.

Jaquet (2003) *"IQ Speed Sensor Data Sheet"*, Jaquet Technology Group, Basel, Switzerland. WWW http://www.jaquet.com

Langheim, J., Henrio, J.-F., Liabeuf, B. (1999) *"ACC Radar System "Autocruise" with 77 GHz MMIC Radar"*, ATA, Florence, 1999.

May, A.D. (1990) *"Traffic Flow Fundamentals"*, Prentice Hall, ISBN 0139260722.

Messmer, H.P. (2002) *"The Indispensable PC Hardware Book"*, Pearson Education Ltd., ISBN 0201596164.

Microcomputer Research (UK) Ltd.(2004) *"MRI-PCI-PP2 Dual Parallel Port Data Sheet"*, Microcomputer Research, London, UK. WWW http://www.mri.co.uk

Pont, M. J., Norman, A.J., Mwelwa, C. and Edwards, T. (2003) *"Prototyping time-triggered embedded systems using PC hardware"*, Proceedings of EuroPLoP 2003, Germany, June 2003.

Pont, M.J. (2001) *"Patterns for time-triggered embedded systems: Building reliable applications with the 8051 family of microcontrollers"*, Addison-Wesley, ISBN 0201331381.

Pont, M.J. (2003) *"An object-oriented approach to software development for embedded systems implemented using C"*, Transactions of the Institute of Measurement and Control, Vol. 25, No. 3, pp. 217-238.

Press, W., Teukolsky, S., Vetterling, W. and Flannery, B. (1992) *"Numerical recipes in C: The art of scientific computing (2nd Edition)"*, Cambridge University Press, ISBN: 0521431085.

Turner, J.D. and Austin, L. (2000) *"A review of current sensor technologies and applications within automotive and traffic control systems"*, Proceedings of the Institution of Mechanical Engineers, Vol. 214, Part D, No. D6, pp. 589-615.