The development of reliable X-by-Wire systems: Assessing the effectiveness of a "simulation first" approach

Thesis submitted for the degree of
Doctor of Philosophy
at the University of Leicester

by

Devaraj Ayavoo
B.Eng. (1st Hons.)

Embedded Systems Laboratory
Department of Engineering
University of Leicester
Leicester, United Kingdom

September 2006

ProQuest LLC
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106-1346

The development of reliable X-by-Wire systems: Assessing the effectiveness of a "simulation first" approach

Devaraj Ayavoo

Abstract

Networks of embedded processors play an increasingly important role in the control of automotive, aerospace, industrial, defence and medical systems. The requirements for such "X-by-Wire" applications are highly demanding and complex in nature, and there are numerous possible design and technology options available. As a consequence, in all but the most trivial systems, engineering teams who wish to identify the "best" solution can only hope to prototype a small percentage of the possible designs.

Several researchers have argued that an effective solution to these problems is to use computer simulations in the early stages of the design process. The aim of this thesis is to explore the effectiveness of such a "simulation first" approach when developing X-by-Wire systems. The main focus is on the automotive sector, but it is suggested that the techniques developed during the course of this project can be more widely applied.

This document makes three main contributions, as follows.

First, it provides clear empirical evidence of the extent to which a "simulation first" approach can be used to support the development of non-trivial X-by-Wire systems.

Second, it introduces a novel, cost-effective empirical "small group methodology" (SGM) to compare between different development techniques for embedded systems. The SGM is described, and its effectiveness demonstrated in four non-trivial case studies.

Third, evidence is presented which suggests that the SGM may be more widely applicable.

Acknowledgement

Firstly, I would like to express my deepest gratitude to my supervisor Dr. Michael Pont for his constant guidance and encouragement throughout this research project. I am very appreciative of all the help that you have provided and have thoroughly enjoyed working with you. You have made this "journey" of mine a very memorable one.

Next, I would like to thank the UK Government (Department of Education and Skills) for awarding me the ORS Scholarship. In addition, I would also like to thank Pi Technology, for partially funding my research project. In particular, I am grateful to Stephen Parker and Mike Ellims for their various useful contributions to the project.

This research project would not have been possible without the peer support of my colleagues in the Embedded Systems Laboratory. I thank you all for your encouragement, support and the occasional distraction. Also, a special thanks to all my colleagues from University of Leicester's Welfare Service. And to my good friends here and abroad, I thank you for your unwavering belief in me.

A special thanks to my uncle, Dr. Nadaraja Kannan, for encouraging me to pursue a doctorate degree.

Last but not least, I cannot thank my parents, sister and all my family members enough for their constant support and prayers over the last few years. Thank you for always being there for me.

I dedicate this thesis to my parents and sister

Ayavoo Kannan, Gunabushany Munusamy & Nalini Ayavoo

Table of Contents

List of Case Studies

List of Figures

List of Tables

LIST OF PUBLICATIONS

A number of papers were published during the course of the work described in this thesis. These are listed below (in reverse chronological order). Please note that the contents of some of these papers have been adapted for presentation in this thesis: where applicable, a footnote at the beginning of a chapter indicates that material from one or more papers has been included. For those papers with contents that have not been included in this thesis, a copy of the paper itself is included in the appendices.

Directly-related publications

Ayavoo, D., Pont, M. J. and Parker, S. (in preparation) "Developing reliable embedded systems: Does a 'simulation first' approach improve software quality?".

Ayavoo, D., Pont, M. J. and Parker, S. (submitted 29 August 2006) "Comparing different techniques for developing software for mechatronic systems: A low-cost empirical methodology". IEEE/ASME Transactions on Mechatronics.

Pont, M. J., Mwelwa, C., Bonthonneau, L., Ayavoo, D., Kurian, S. and Ward, D. (submitted 10 August 2006) "Pattern-based development of time-triggered embedded systems using software tools: Challenges and solutions". Journal of Systems and Software.

Ayavoo, D., Pont, M.J. and Parker, S. (2006) "Does a 'simulation first' approach reduce the effort involved in the development of distributed embedded control systems?". Proceedings of the 6th UKACC International Control Conference, Glasgow, Scotland, 2006.

Ayavoo, D., Pont, M.J. and Parker, S. (2005a) "Observing the development of a reliable embedded system". In Vardanega, T. and Wellings, A. (Eds.) "Proceedings of the 10th Ada-Europe International Conference on Reliable Software Technologies, York, UK, June 20-24 2005", p. 167-179. Lecture Notes in Computer Science, Vol. 3555. Published by Springer-Verlag [ISBN: 3-540-26286-5]. *(A copy of this paper is included in Appendix-A)*

Ayavoo, D., Pont, M.J. and Parker, S. (2004) "Using simulation to support the design of distributed embedded control systems: A case study". In: Koelmans, A., Bystrov, A. and Pont, M.J. (Eds.) "Proceedings of the 1st UK Embedded Forum 2004 (Birmingham, UK, October 2004)", p. 54-65. Published by University of Newcastle upon Tyne [ISBN: 0-7017-0180-3].

Associated publications

Ayavoo, D., Pont, M.J., Short, M. and Parker, S. (accepted) "Two novel shared-clock scheduling algorithms for use with CAN-based distributed systems". Microprocessors and Microsystems. *(Accepted for publication subject to minor revisions. A copy of this paper is included in Appendix-B)*

Ayavoo, D., Pont, M.J., Short, M. and Parker, S. (2005b) "Two novel shared-clock scheduling algorithms for use with CAN-based distributed systems". In: Koelmans, A., Bystrov, A., Pont, M.J., Ong, R. and Brown, A. (Eds.), Proceedings of the Second UK Embedded Forum (Birmingham, UK, October 2005), p. 246-261. Published by University of Newcastle upon Tyne [ISBN: 0-7017-0191-9].

Ayavoo, D., Pont, M.J., Fang, J., Short, M. and Parker, S. (2005c) "A 'hardware-in-the-loop' testbed representing the operation of a cruise-control system in a passenger car". In: Koelmans, A., Bystrov, A., Pont, M.J., Ong, R. and Brown, A. (Eds.), Proceedings of the Second UK Embedded Forum (Birmingham, UK, October 2005), p. 60-90. Published by University of Newcastle upon Tyne [ISBN: 0-7017-0191-9]. *(A copy of this paper is included in Appendix-C)*

List of Abbreviations, Symbols and Units

Abbreviations

ACCS	Adaptive Cruise-Control System
ADC	Analogue to Digital Converter
ASIC	Application Specific Integrated Circuit
CA	Controller/Actuator
CAN	Controller Area Network
CASE	Computer Aided Software Engineering
CCS	Cruise-Control System
CSMA/CA	Carrier Sense Multiple Access / Collision Avoidance
DAC	Digital to Analogue Converter
DSP	Digital Signal Processor
DVD	Digital Versatile Disc
ESL	Embedded Systems Laboratory
FPGA	Field Programmable Gate Array
GQM	Goal Question Metrics
HIL	Hardware-in-the-Loop
HW	Hardware
I/O	Input/Output
IMP	Implementation
kbps	kilo bits per second
LCD	Liquid Crystal Display
LED	Light Emitting Diode
LIN	Local Interconnect Network
LOC	Lines of (Source) Code
LQR	Linear Quadratic Regulator
Mbps	Mega bits per second
O-O	Object-Oriented
PDA	Personal Digital Assistant
PID	Proportional, Integral and Derivative
PIE	Pattern Implementation Example
PTTES	Patterns for Time-Triggered Embedded Systems
PWM	Pulse Width Modulation
SCC	Shared-Clock CAN
SGM	Small Group Methodology
SIM	Simulation
SW	Software
TDMA	Time Division Multiple Access
TTCAN	Time-Triggered Controller Area Network
TTP	Time-Triggered Protocol
TTP/C	Time-Triggered Protocol, Class C
TT-Plus	TrueTime-Plus
UML	Unified Modelling Language
VAN	Vehical Area Network

Symbols

a	Acceleration
Fr	Frictional coefficient
m	Mass
v_f	Final speed
v_i	Initial speed
Δx	Displacement
θ	Throttle setting
τ	Engine torque

Units

m/s	Metres / Second
ms	Millisecond
s	Second
μs	Microsecond

1. Introduction

In this introductory chapter, an overview of the work undertaken in this thesis is presented and the importance of this area is discussed.[1]

1.1 Introduction

Most branches of engineering have a long history associated with them. For instance, the study of electrical engineering can be dated back to the invention of the electric motor by Michael Faraday in 1821 (Faraday, 2004), while James Watt is largely recognised for his seminal contributions to control engineering with the development of the flyball governor in the 1760s (Marsden, 2004). It is possible that the practice of civil engineering may have the longest history of all, dating back to the building of the Egyptian pyramids. Certainly, the Institution of Civil Engineers was founded in the UK in 1818 and is the oldest professional engineering institution in the world (ICE, 2006).

For the software engineer, a different situation applies. Although Charles Babbage began work on his 'Analytical Engine' around 1833 (Babbage, 1888), it was not until 1965 that the first mass-produced mini computer, the PDP-8, was launched (Jones, 2004). Intel followed suit with the release of the first microprocessor, the Intel 4004, only in 1971 (Wilson, 2001). As a result of the late introduction of small programmable computer systems, the field of software engineering lacks the rigorous theoretical foundation which marks out most branches of the engineering profession. This situation might not matter if software development was a niche occupation: however, the development of software-based systems lies at the heart of many business and technical activities (Storey, 1996; Lewis, 2001).

The focus of this thesis is on the development of reliable software for embedded systems. Embedded systems are special-purpose computers that combine hardware and software and are mounted on compact electronic circuit boards that are used inside a particular device (Sachitanand, 2002). Very often, such systems interact with the real world through input/output devices such as push-button switches, potentiometers, LCD panels and LEDs.

[1] Parts of this chapter have previously been published in Ayavoo *et al.* (2004) and Ayavoo *et al.* (2006).

Embedded systems may include microcontrollers, microprocessors, DSPs, FPGAs and/or ASICs to perform the core computation.

Embedded processors can be found in many day-to-day applications such as digital cameras, Personal Digital Assistants (PDAs), palm top computers, DVD players and mobile phones (Chouliaras *et al.*, 2005). In addition to these "non-critical" systems, embedded processors are also used in safety-critical applications such as automotive, aerospace, defence and medical systems (Storey, 1996). The use of embedded processors in such applications is very attractive as it can greatly improve the functionality of a particular product at very low cost (Pont, 2001; Sachitanand, 2002). Over the years, many industrial firms (such as Intel, Infineon, Philips, Atmel, Altera, Xilinx, Texas Instruments) have been actively producing a range of different embedded processors to match the growing needs and demand of the modern technology. Overall, embedded processors are poised to revolutionise many industrial sectors and consumer products by making various systems more "intelligent".

1.2 Description of the problem

With the rapid growth of technology, the development of huge and complex embedded systems is becoming increasingly difficult; especially where system safety and reliability is crucial. This section describes the difficulties involved in detail.

1.2.1 Distributed embedded control systems in the automotive sector

The focus of the work presented in this thesis is on the development of reliable embedded systems, particularly for automotive applications. Here, embedded processors are becoming ubiquitous (Maier *et al.*, 2002) and are used to perform a variety of safety-related functions in the vehicle such as automatic transmission, anti-lock braking systems and engine control. The number of embedded processors used in a vehicle has also been steadily increasing over the past few years (see Figure 1-1), and it is expected that this trend will continue over the next few years, as the complexity and functionality of the system increases (Lanfear *et al.*, 2006).

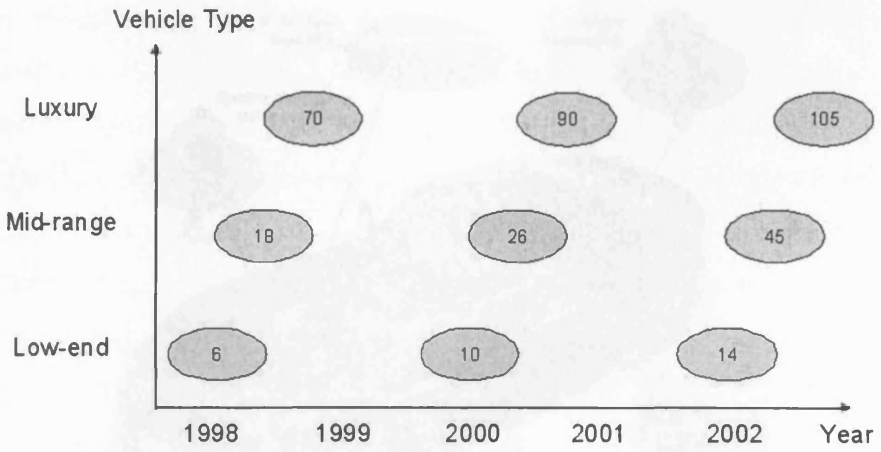

Figure 1-1 Microcontroller implementation growth in automobiles. Redrawn from data in Bannatyne (2003), Figure 1.

It is also expected that an increasing number of road vehicles will soon contain sophisticated distributed embedded control systems, consisting of a number of microcontrollers linked by one or more computer networks (Leen *et al.*, 1999): crucially, these systems will have no mechanical backup. As with similar distributed embedded control systems in other sectors (including aerospace, medicine, manufacturing and defence), these embedded automotive designs will be highly complex, and will have a central role in safety.

This cutting edge technology, which is also known as X-by-Wire[2] systems (Ellims *et al.*, 2002; Fredriksson, 2002; Koopman, 2002; Nossal and Lang, 2002), can be argued to have been inspired by the success of Fly-by-Wire systems adopted in the aerospace industry (Brière *et al.*, 1995; Schmitt *et al.*, 1998). The 'X' in X-by-Wire represents the basis of any safety-related application, such as steering, braking, power train, throttle or suspension control systems (see Figure 1-2). It is expected that X-by-Wire systems will assist the driver in different situations and make it safer for all road-users (Bretz, 2001; Koopman, 2002).

[2] Please note that the term X-by-Wire is used in the remainder of this thesis to represent any distributed embedded control systems.

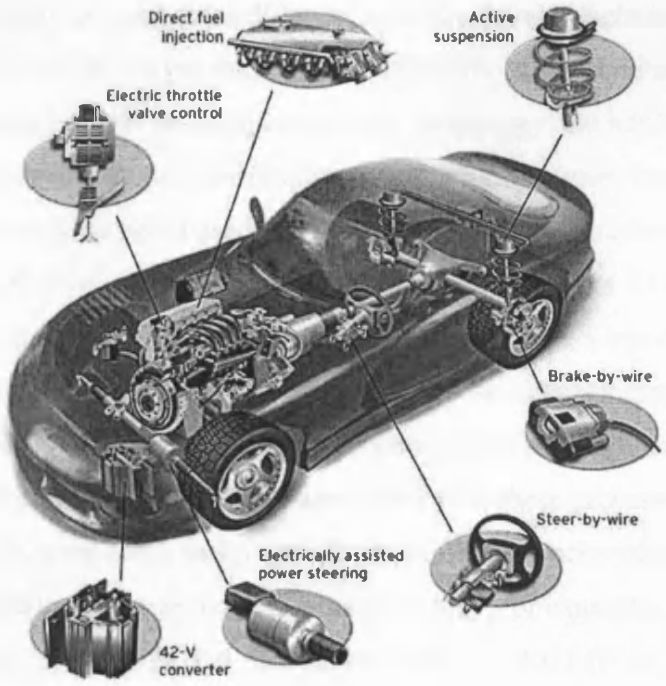

Figure 1-2 An illustration of an X-by-Wire system in a car from Bretz (2001), Figure 1. Copyright IEEE, reproduced with permission.

1.2.2 Why is the development of X-by-Wire systems difficult?

The designers of X-by-Wire systems, particularly for those in the automotive industry, face the challenge that the resulting systems must operate very reliably, and - at the same time - have minimal maintenance requirements and a low purchase price (Kopetz, 1995; Schoitsch, 2003). As a consequence of the market's demand for lower cost, the X-by-Wire systems often have to operate within severe resource constraints such as limited CPU speed, memory constrictions, minimal hardware peripherals and restricted network bandwidth.

Over the last few years, software engineers and programmers have been influenced by a stream of new methodologies and techniques, such as object-oriented (O-O) development (Dahl and Ngyaard, 1966), design patterns (Cunningham and Becks, 1987; Gamma *et al.*, 1995), aspect-oriented design (Kiczales *et al.*, 1997) and "extreme" programming (Becks, 2000). In any company, it is very difficult for team managers to determine whether the use of one or more of these techniques is likely to prove beneficial (Fenton *et al.*, 1994; Basili *et al.*, 1999; Dyba *et al.*, 2005).

In addition to choosing between these different software development methodologies, the implementation of X-by-Wire systems must also constantly adapt to changes in the software environment brought about by changes in technology and hardware design. For example, the designer of a modern passenger car may need to choose between the use of one (or more) network protocols based on CAN (Rajnak and Ramnerfors, 2002), TTCAN (Hartwich *et al.*, 2002), LIN (Specks and Rajnak, 2002), FlexRay or TTP/C (Kopetz, 2001). The resulting network may be connected in, for example, a bus or star topology (Tanenbaum, 1995). The individual processor nodes in the network may use event-triggered (Nissanke, 1997) or time-triggered (Kopetz, 1997) software architectures, or some combination of the two. The clocks associated with these processors may be linked using, for example, shared-clock techniques (Pont, 2001) or synchronisation messages (Hartwich *et al.*, 2000). These individual processors may, for example, be C167 (Siemens, 1996), ARM (ARM, 2001), MPC555 (Bannatyne, 2003) or 8051 (Pont, 2001).

To further complicate matters, the different design options are far from independent: for example, the CAN bus is supported on 8-bit processors for which – in many cases – memory restrictions means that O-O programming languages (like C++) are not widely available (Pont, 2003). However, use of a FlexRay bus as an alternative to CAN is likely to require a more powerful processor with increased memory: use of such a processor may, in turn, make the use of an O-O language feasible.

Moreover, most embedded systems that are related to safety-critical applications have "hard" timing constraints (Liu, 2000). This means there is no allowance for errors or "lateness" in the way the system executes with regards to its timing. For example, in a Brake-by-Wire system, the brake actuators may be required to respond within a fixed amount of time after the brake pedal has been pressed. If the system is unable to respond within this time frame, there could be a danger that the vehicle may not stop in time before crashing into another vehicle. In such cases, it is better to predict the timing behaviour of an embedded system to ensure that all timing constraints are met.

However, predicting the behaviour of X-by-Wire system is not straightforward. For future automotive X-by-Wire systems, it is unlikely that (say) a Brake-by-Wire system will work independently; by contrast, this system may be partially dependent on other control systems of the vehicle such as throttle and cruise control. For example, suppose the brakes

fail, the control system will wish to cut the throttle. Furthermore, it is also expected that these systems will share certain resources such as the network bandwidth, and have multiple levels of redundancy (Isermann *et al.*, 2002) such as back-up nodes (Short *et al.*, 2006) and secondary network protocols (Hilmer *et al.*, 1998; Pimentel and Fonseca, 2004). For such complex systems and taking into account the different design options and development methodologies available, the process of ensuring that all timing constraints are satisfied can be difficult.

Overall, the number of possible system designs is enormous, and prototyping even a small subset of these different systems is impractical. If the developers of embedded systems intend to rapidly understand the behaviour of a particular system, or identify the "best" system architecture from a range of possible options, an alternative approach is required (Castelpietra *et al.*, 2001).

1.2.3 A "simulation first" approach

The work described in this thesis was initiated with preliminary discussions between representatives from Pi Technology and Dr. Michael Pont from the Embedded Systems Laboratory (ESL), University of Leicester. The initial discussions resulted in the decision to develop a tool that supports the simulation of embedded systems, particularly for those in the automotive industry. The tool was expected to be able to support the development of reliable, cost-effective, X-by-Wire networks by allowing different network designs to be rapidly assessed, and their features compared, early in the product lifecycle.

Various researchers have suggested that a simulation approach could be used to design and validate various implementation options in this way (see El-khoury and Törngren, 2001; Palopoli *et al.*, 2001; Törngren *et al.*, 2001; Castelpietra *et al.*, 2002; Cervin *et al.*, 2003; Karatza, 2004; Redell *et al.*, 2004). For example, according to Karatza, "*The most straightforward way to evaluate the performance without a full-scale implementation is through a modelling and simulation approach. Detailed simulation models help determine performance bottlenecks inherent in the architecture and provide the basis for refining the system configuration.*"(Karatza, 2004, p. 183).

A versatile simulator is often expected to provide various benefits to the development process of embedded systems. For instance, El-khoury and Törngren (2001) suggest that a simulator could be used to address the development challenges by evaluating various error-detection and error-handling mechanisms for X-by-Wire systems. Redell and colleagues imply that a simulation approach could help lower development times and minimise implementation-related problems at the later stages of the design process (Redell *et al.*, 2004). Palopoli and colleagues conclude that a simulation tool can help provide guidelines to the choice of design by allowing the assessment of different system design options (Palopoli *et al.*, 2002). Henzinger and colleagues suggest that this approach can potentially increase the reliability and reusability of control software (Henzinger *et al.*, 2003).

Overall, the literature suggests that simulation has the potential to assist in the development of reliable X-by-Wire systems. Hence, the initial aim of the research project described in this thesis was to develop a simulation tool that could assist in the design and implementation of such systems.

1.3 Key contributions

This thesis makes three contributions to this research area:

First, empirical evidence of the contributions that simulation can make in the development of reliable X-by-Wire systems is provided. Specifically, the impact of a "simulation first" approach on the system development effort and software quality is reported.

Second, the thesis introduces a novel, cost-effective empirical methodology - the "Small Group Methodology" (SGM) - that can be used to compare development techniques for embedded systems. It is demonstrated that this approach allows a detailed evaluation and analyses of the various stages of the software development process.

Third, evidence is provided that the SGM may be suitable for use in a wider range of application areas. In support of this claim, a case study is presented in which the SGM was used to assess the effectiveness of a CASE tool in the implementation of reliable software patterns for embedded systems.

1.4 Thesis layout

Following this introductory chapter, Chapter 2 provides an overview of some of the existing simulation tools that are available. Chapter 3 then uses one particular simulator, and investigates the extent to which the tool can be used to predict the behaviour of various X-by-Wire systems.

In Chapter 4, discussions are carried out on techniques that can be used to evaluate the efficacy of novel development methodologies when used in practice. A description of the SGM is then presented.

Chapters 5, 6 and 7 are devoted towards using the SGM to assess the efficacy of the "simulation first" approach in the development of reliable embedded control systems. Chapter 5 describes a case study to investigate if a "simulation first" approach can reduce the overall effort involved in the development process. A different case study is used in Chapter 6 to explore ways in which the effort involved can be further reduced. In Chapter 7, the SGM is used to evaluate contributions of a "simulation first" approach on software quality.

In Chapter 8, a different (and otherwise unrelated) case study was carried out to investigate the extent to which the SGM can be applied more widely in other studies.

The discussion and conclusions are presented in Chapters 9.

1.5 Conclusions

This introductory chapter has presented an overview on the complexity involved in the development of reliable X-by-Wire systems. To assist in the development of such complex systems, some researchers have proposed a "simulation first" approach. Based on the discussion presented, the initial aim of the research project was outlined, and the contributions of this thesis were summarised. The remainder of this thesis will describe the work undertaken throughout this research project.

2. What simulators are available?

It was argued in Chapter 1 that the development of reliable X-by-Wire systems is becoming increasingly challenging. To address this issue, the research presented in this thesis was centred on the use of a "simulation first" approach in the development of X-by-Wire systems. Previous work in this area is reviewed in this chapter.[3]

2.1 Introduction

The idea of using computer simulations to better understand the characteristics of a particular phenomenon can be dated back to World War II when researchers were interested in studying the behaviour of neutrons. It was then that John von Neumann and Stanislaw Ulam came up with the notion of using Monte Carlo simulations to model nuclear detonation (Ulam *et al.*, 1947). Computer simulations in those days were not very useful, mainly because it cost too much and took a very long time to generate results. However, since then, computer simulations have come to play an important part in many sectors, such as medical (Popp *et al.*, 2004), defence (Ören *et al.*, 2000) and aerospace (Graziani, 2002).

Please note that such an approach is not without its critics (see Kurkowski *et al.*, 2005; Andel and Yasinsac, 2006). For example, Andel and Yasinsac (2006), p. 48, have argued that: "*Simulation is a powerful tool, but it's fraught with potential pitfalls. We question this approach's validity and show how it can systematically produce misleading results.*". Such views are not universally held and, as system development becomes ever more challenging, various researchers have attempted to simulate their application.

With regards to the simulation of embedded control systems, the traditional approach has been to use a comprehensive mathematical library, such as MATLAB and Simulink[4]. These tools are used to fine tune the mathematical models and control algorithms until satisfactory performance of the control system is achieved. The embedded control system is then either implemented manually or automatically (using code generation tools such as Real-Time Workshop Embedded Coder[4]). The code is then tested and optimised until it demonstrates the required timing behaviour (Henzinger *et al.*, 2003).

---

[3] Parts of this chapter have previously been published in Ayavoo *et al.* (2004) and Ayavoo *et al.* (2005c).

[4] http://www.mathworks.com/ (last accessed: 16/08/2004)

Developers working in this way tend to make the assumption (implicitly or explicitly) that the delays between the various control events are constant (Aström and Wittenmark, 1990). However, such assumptions are rarely valid when embedded control systems are implemented on the hardware (see Törngren, 1998; Sandfridson, 2000).

Another drawback with this approach is that – once the code has been generated – the link to the initial library is broken: as a consequence, adding new system functionality, porting the code to a different processor or simply "refining" the code may invalidate assumptions made during the early stages of the design process (Henzinger *et al.*, 2003).

To address these issues, various researchers (for example, see El-khoury and Törngren, 2001; Palopoli *et al.*, 2001; Törngren *et al.*, 2001; Castelpietra *et al.*, 2002; Cervin *et al.*, 2003; Karatza, 2004; Redell *et al.*, 2004) have argued for the use of simulation tools that allow developers of control systems to take into account the real-time implementation aspects of X-by-Wire systems. Several institutes have since begun to investigate the use of a similar tool support for the development of embedded systems, including Lund Institute of Technology (Eker and Cervin, 1999; Cervin *et al.*, 2003), Royal Institute of Technology Sweden (Törngren *et al.*, 2001), Uppsala University (Amnell *et al.*, 2002) and RETIS Laboratory Pisa (Palopoli *et al.*, 2001). This has resulted in the development of several simulation tools.

To avoid reinventing the wheel, a sensible starting point for this research was to review the range of available simulators. More specifically, this chapter attempts to establish if any of the available tools can currently meet the requirements of developers of X-by-Wire systems. To do this, a study on some of the basic features required in a simulator is carried out in Section 2.2. Section 2.3 then reviews some of the previous work carried out in the area of simulation for real-time embedded control systems. The discussions and conclusions are presented in Section 2.4 and Section 2.5, respectively.

2.2 What do we need to simulate?

The initial goal of this research was to build a simulator that can support the development of X-by-Wire systems, particularly in the automotive sector (see Section 1.2.3). To do this, it was necessary to identify the features of an appropriate simulator.

In Section 2.1, it was argued that that such a simulator must incorporate the real-time implementation aspects of control systems. Törngren *et al.* (2001) have previously described an "ultimate simulator" that includes various implementation attributes such as distributed real-time control applications, network, nodes, schedulers/real-time kernels and the modelling of system and environment. It is important to take some of these implementation options into consideration because they could potentially introduce uncertainties in the overall performance for X-by-Wire systems (see Törngren, 1998; Chen and Sandfridson, 2000).

Based on this discussion, the initial focus for the requirements of a simulator was outlined, as illustrated in Figure 2-1. A description of these features is provided in the following sections.

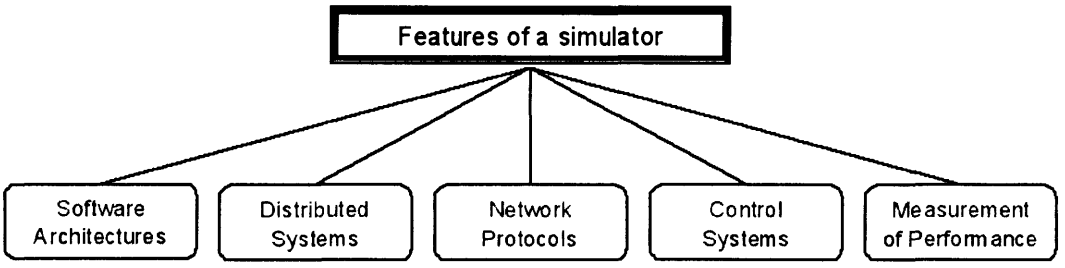

Figure 2-1 Required features of a tool to simulate X-by-Wire systems.

2.2.1 Software architectures

Although most embedded processors are required to run only one program, it is often the case that the program is executed through a collection of tasks (e.g. Nissanke, 1997; Shaw, 2001). These tasks can, for example, be implemented as 'C' functions (Pont, 2001).

The various possible software architectures in embedded design are often characterised in terms of the tasks that are to be performed (Marwedel, 2003). For example, if the tasks are invoked by events (typically hardware interrupts) the system may be described as 'event triggered' (Nissanke, 1997). Alternatively, if all the tasks are invoked periodically

(say every 10 ms) under the control of a timer, then the system may be described as time-triggered (Kopetz, 1997). The nature of the tasks themselves is also significant. If the tasks, once invoked, can pre-empt (or interrupt) other tasks, then the system is said to be 'pre-emptive'; if tasks cannot be interrupted, the system is said to be co-operative (or non-pre-emptive). Figure 2-2 illustrates how three tasks on an embedded processor may execute when using co-operative and pre-emptive schedulers.

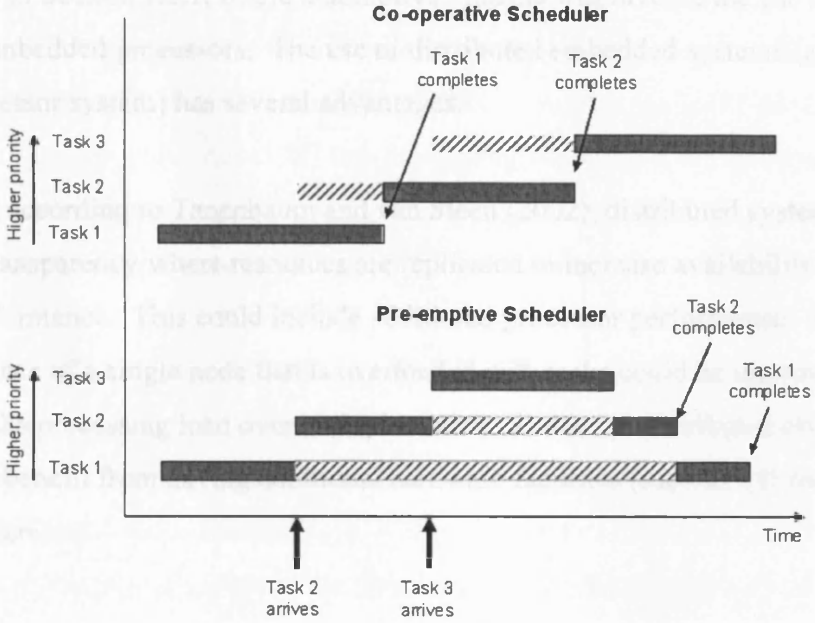

Figure 2-2 Task execution for co-operative and pre-emptive schedulers.

Note that a system may support both time-triggered and event-triggered tasks; some of these may be co-operative in nature while others are pre-emptive.

Another possibility to consider is the design of hybrid schedulers for embedded systems. Such systems could include - for example - tasks that are time-triggered and co-operative in nature with one high-priority event task that can pre-empt other tasks (Maaita and Pont, 2005). Other task scheduling algorithms include fixed-priority techniques like deadline-monotonic and rate-monotonic algorithms; and dynamic-scheduling approaches like the earliest-deadline-first algorithm (see Liu, 2000).

Overall, there is more than one way to schedule a set of tasks on an embedded processor. Each of these techniques could potentially affect the general performance of an embedded

control system in a different way (Fang and Pont, 2006). In this case, it has been argued that it would be useful to have a simulator that can simulate different software architectures to compare the system performance (Eker and Cervin, 1999; Törngren *et al.*, 2001).

2.2.2 Distributed systems

As described in Section 1.2.1, future automotive systems will involve the use of distributed embedded processors. The use of distributed embedded systems (as opposed to a single processor system) has several advantages.

For instance, according to Tanenbaum and van Steen (2002), distributed systems offer replication transparency where resources are replicated to increase availability and improve performance. This could include additional processor performance: for example, the performance of a single node that is overloaded with tasks could be improved by distributing the processing load over multiple nodes. Similarly, distributed embedded systems also benefit from having additional hardware facilities (such as I/O ports and hardware timers).

Furthermore, the use of distributed systems could also lead to modular embedded design architecture (Lonn, 1999; Chen and Sandfridson, 2000). Chen and Sandfridson (2000) described this as exploiting the natural parallelism inherent in distributed architectures. This could – for example – be the use of smart sensors and smart actuators where sensor related functions are kept separate from the actuator. In some instances, it could also be the case that distributed systems are preferred due to the physical location of the sensors and actuators (see Ball, 1996; Chen and Sandfridson, 2000).

Another advantage includes the capability of incorporating failure handling mechanisms, like redundancy. Lonn (1999) described an example where three out of four wheels that incorporates smart sensors and actuators can still safely stop a car, although the distance may increase and the braking conditions be poor under certain conditions.

With the various advantages of using distributed architectures, it is becoming necessary to predict the behaviour for future X-by-wire systems (see Section 1.2.2). To do this, it has

been argued that the capability of modelling distributed embedded systems is a useful feature that a simulation tool should have (Törngren *et al.*, 2001; Redell *et al.*, 2004).

2.2.3 Network protocols

To connect the various distributed embedded processors such that information can be exchanged among the nodes, a form of communication protocol is required (Tanenbaum and van Steen, 2002). Initially, the generic UART (Universal Asynchronous Receiver/Transmitter) device was used to transmit and receive messages using serial protocols like RS232 and RS485 (Leen *et al.*, 1999). However, as use of distributed electronics begun to expand into safety critical application (such as automotive control systems), network protocols that are more deterministic (Lonn, 1999) and have sophisticated fault-tolerant features (Kopetz, 1995) were needed.

In the mid 1980s, Robert Bosch GmbH introduced the Controller Area Network (CAN) protocol, which was first implemented in the Mercedes Benz S-class car in 1991 (Bosch, 1991; Leen *et al.*, 1999). The CAN protocol – which employs a CSMA/CA message transmission scheme – has since been widely used in many sectors, such as automotive and automation (Fredriksson, 1994; Zuberi and Shin, 1995; Sevillano *et al.*, 1998; Thomesse, 1998; Pazul, 1999; Farsi and Barbosa, 2000; Misbahuddin and Al-Holou, 2003). Besides the CAN protocol, there are also other new network protocols that are competing for a place in the automotive industry. These include TTCAN (Hartwich *et al.*, 2002), TTP (Kopetz, 2001) and FlexRay (Litterick and Brenner, 2005) that uses the Time Division Multiple Access (TDMA) technique to allocate data transmissions on the network bandwidth (Maier *et al.*, 2002). Some of the differences in these protocols are outlined in Table 2-1.

Table 2-1 Some differences in the characteristics of the competing network protocols in the automotive sector for safety-critical applications[5] (original sources obtained from Kopetz, 2001; Leen and Heffernan, 2002; Maier *et al.*, 2002).

Feature	CAN	TTCAN	TTP/C	FlexRay
Transmission speed	1 Mbps	1 Mbps	25 Mbps	10 Mbps
Data size (bytes)	8	8	240	246
Fault tolerant clock synchronisation	No	Yes	Yes	Yes
Replicated communication channels	No	No	Yes	Yes
Bus guardian (avoid babbling idiot)	No	No	Yes	Yes

Currently, there is no one protocol that can be claimed as being the "defacto" standard in the automotive industry. This is because each protocol has its own advantages and drawbacks. For example, although TTP/C appears to be far superior to CAN, the cost is also much higher than CAN controllers. In general, the use of any one of these techniques described could potentially affect the performance of an X-by-Wire system.

In some cases, the X-by-Wire system may also include bridge architectures (Wolf and Koller, 1998), network gateways (Ekiz *et al.*, 1996) and bus guardians (Temple, 1998), which could further complicate the analysis of the system performance. Therefore, it has been argued that the use of a flexible simulation tool that can model some of these network protocols could be valuable in the development of X-by-Wire systems (Cervin *et al.*, 2003).

2.2.4 Control systems

As described in Chapter 1, the focus of this thesis is on the development of reliable embedded control systems.

The process of control can usually be divided into three main operations: sampling, control and actuation. In the first operation, data are sampled. In the second operation, these data are processed using an appropriate control algorithm. In the third operation, an output signal is produced that (generally) alters the system state. In a single-processor system, all three functions are carried out on the same node. In a distributed environment, these functions may be carried out on multiple nodes, linked by an appropriate network

[5] Please note that, LIN (Specks and Rajnak, 2002) is not included in this table because it is not generally considered to be suitable for safety-critical applications (Ahlmark, 2000).

protocol. For example, a two-node design might carry out the sampling operations on Node 1, and the control and actuation operations on Node 2 (see, for example Lonn and Axelsson, 1999; El-khoury and Törngren, 2001).

To simulate the complete control system, both a model of the plant and a model of the control process are required (Dorf and Bishop, 2000). The term plant is often used to indicate the environment to be controlled; such as the engine, the wheels and brakes, steering wheel or the overall physical car dynamics. In a real implementation, the plant and control process will usually be linked by various I/O mediums, such as port pins, serial bus and parallel cables. This is illustrated in Figure 2-3.

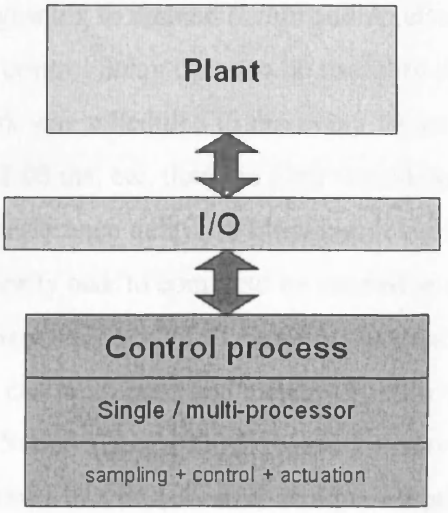

Figure 2-3 The plant, control process and I/Os for a control system.

Hence, if the aim is to model complete control systems, then the tool should support the simulation of both the control process as well as the plant dynamics (Törngren *et al.*, 2001). This also makes it possible to research the dynamic performance of a particular X-by-Wire system from a control perspective (Cervin *et al.*, 2003).

2.2.5 Measurement of system performance

As discussed in Section 2.1, the performance of a control system not only depends on the control algorithm, but also the way the overall system has been implemented in real-time (for example the number of nodes, choice of software architecture and network protocol). To assess the performance of a control system, timing measurements can be used as a crucial source of information (Wittenmark *et al.*, 1995; Sandfridson, 2000).

For instance, the measurement of an event response time can show how long a process would take to complete from the time a particular event has been detected on the system (Lonn and Axelsson, 1999). This can perhaps be used to measure the time between the detection of sensor failure (on one node) and the reaction to the failure (on the second node). More generally, event response times are an important concern in a distributed environment, where the signal might have to travel through a number of nodes and across several networks before it reaches its destination.

Control delay is another measure that is important to determine the time taken to perform a control process from the beginning to the end (Lonn and Axelsson, 1999; Sandfridson, 2000). The variation in the control delay can also be useful to determine the control jitter. For example, if a control task was scheduled to run every 10 ms, and executed at (say) t = 2.00 ms, t = 12.00 ms, t = 22.00 ms, etc, then the jitter would be 0. Jitter in this case can be caused by blocking or interference delay. A blocking delay is the longest time that task x has to wait for a lower-priority task to complete its execution (Tindell and Clark, 1994; Lonn and Axelsson, 1999; Sandfridson, 2000). An interference delay is the longest time that all higher priority tasks can be queued and executed before task x is finally executed (Lonn and Axelsson, 1999; Sandfridson, 2000). Figure 2-4 shows examples of blocking and interference delays for tasks in a co-operative and pre-emptive scheduler.

Ideally, it is preferable that embedded control systems have low event response time, short and constant control delay, and minimal control jitter (Lonn, 1999; Liu, 2000; Marti *et al.*, 2001).

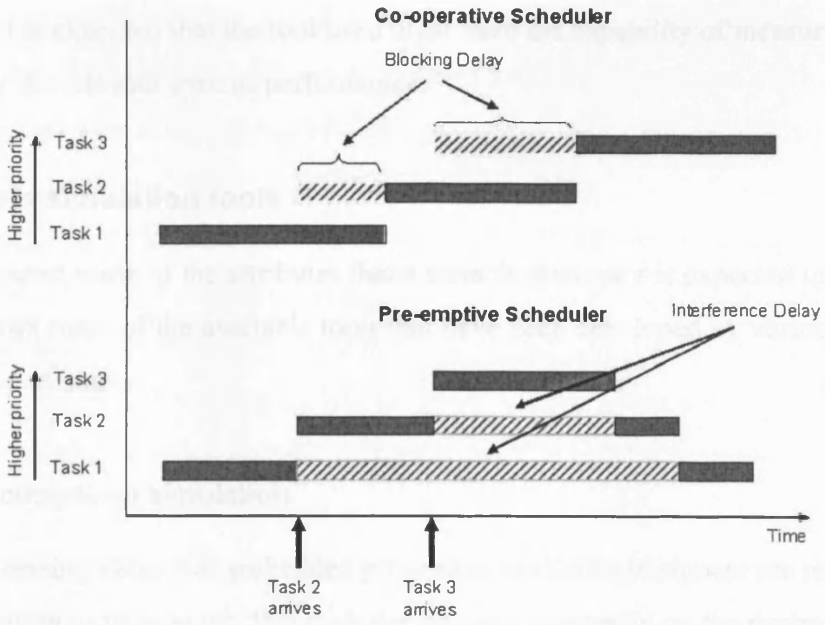

Figure 2-4 Examples of blocking and interference delay for co-operative and pre-emptive schedulers.

Since the choice of the system implementation can have an impact on these measurements and influence the overall system performance, it would be useful if the chosen simulator can collect and report the relevant measures to the system designer.

2.2.6 Summary of required features

The discussions presented in Section 2.2 covers various useful features for a suitable tool to simulate X-by-Wire systems. These features are summarised here.

- At a processor level, it has been argued that a simulator should be able to model the behaviour of a range of software architectures like time-triggered and event-triggered systems.
- To simulate X-by-Wire systems, it has been argued that the simulator should support a distributed architecture of the control system, as well as a range of network protocols used in the automotive sector like CAN.
- A case has also been made that in the event of modelling a complete control system, the tool should be capable of simulating the distributed embedded controller as well as the plant dynamics.

- Finally, it is expected that the tool used must have the capability of measuring and reporting the relevant system performance.

2.3 Available simulation tools

Having discussed some of the attributes that a suitable simulator is expected to have, this section reviews some of the available tools that have been developed by various research groups and companies.

2.3.1 Instruction-level simulation

With the increasing variety of embedded processors available, designers are relying on the use of simulation tools to verify that their design works correctly on the desired platform (e.g. Engblom *et al.*, 2006).

Some development tools come with its own built-in simulators to assist in the low-level simulation of the embedded processor. The simulation generally executes at the instruction level of the source code, and is tightly coupled with the embedded processor in use.

For example, the Keil IDE[6] is primarily used to write and compile C or assembly code for use in microcontrollers such as the 8051, C167 and ARM. In addition, the tool incorporates its own simulator to imitate the behaviour of various microcontrollers at the instruction level. Similarly, another tool specially dedicated for an 8051 processor has been developed to simulate faults that occur due to Electromagnetic Interference (Ong, 2002). Altera's Quartus-II (Altera, 2005) is another software development tool that has a built-in simulator that can perform behavioural and timing verifications of a design described using a hardware description language (such as VHDL and Verilog).

In addition, there are also third party vendors that specialise in simulating embedded systems at low level. For example, Modelsim[7] by Mentor Graphics is capable of simulating the detailed behaviour of various FPGAs. With the correct simulation library provided by the chip vendors (such as Altera and Xilinx), Modelsim can simulate all the

[6] http://www.keil.com/ (last accessed: 21/05/2005)
[7] http://www.model.com/ (last accessed: 07/11/2004)

signals on the processor for each electronic gate and wire. Such simulation is not only restricted to the functional behaviour, but includes timing details as well. Other third party FPGA simulators include Synopsys VCS (by Synopsys[8]) and Incisive (by Cadence[9]).

The tools described in this section perform the simulation at very low level and are processor specific. However, simulating X-by-Wire systems at such low level can be extremely difficult. For example, a system developer may need to go through the tedious process of configuring all the relevant device drivers at low level (such as hardware timers and I/O ports) just to simulate an LED flashing every one second. Indeed, it has been suggested that some researchers prefer not to simulate X-by-Wire systems at such low level (see Cervin *et al.*, 2003). In such cases, it might be better to just build the prototype.

It is concluded that simulating X-by-Wire systems at such low level can be extremely tedious, difficult and time consuming.

2.3.2 Analytical simulation models

In Section 2.3.1, it was concluded that using processor-level simulation tools may be unsuitable as it is extremely difficult to simulate complex X-by-Wire systems. Some researchers have therefore tried to model X-by-Wire systems at a higher level of abstraction. Such an approach requires the creation of analytical models to simulate control systems that are less dependent on the low-level specifications of the embedded processor.

For example, the study on holistic scheduling approach (Tindell and Clark, 1994) and the VOLCANO tool (Rajnak and Ramnerfors, 2002) have used various analytical models to calculate message latencies and task response times (see Audsley *et al.*, 1993; Tindell and Burns, 1994). Similarly, other researchers have also worked on the development of analytical models for a variety of software architectures and network protocols that include time-triggered and event-triggered systems (see Lonn and Axelsson, 1999; Pop, 2003). Appendix-H provides some examples of these analytical models.

---

8 http://www.synopsys.com/ (last accessed: 30/03/2006)

9 http://www.cadence.com/ (last accessed: 02/04/2006)

The simulation techniques discussed here depends heavily on the analytical models of the real-time system. Considering the complexity of X-by-Wire systems, such high-level models may prove to be rather inaccurate (Castelpietra *et al.*, 2002). For instance, the work by Tindell and Clark (1994) only calculates the worst-case end-to-end response time for distributed periodic tasks. The actual response time may however be lower than the worst-case scenario.

Moreover, such tools are usually targeted towards a specific implementation approach, making it inflexible to simulate other techniques. For instance, the work carried out by Lonn and Axelsson (1999) does not cater for the Shared-Clock CAN (SCC) communication approach (see Pont, 2001; Ayavoo *et al.*, 2005b). As such, a significant change to the proposed model is required in order to simulate a SCC system.

Overall, although analytical simulation models are useful, the extent to which these tools can be used to simulate a wide range of X-by-Wire systems is limited.

2.3.3 Recent general-purpose simulation tools

To overcome some of the complexities involved in simulating a wide range of embedded systems, general-purpose simulators have been developed recently (see Amnell *et al.*, 2002; Castelpietra *et al.*, 2002; Palopoli *et al.*, 2002; Cervin *et al.*, 2003; Henzinger *et al.*, 2003; Redell *et al.*, 2004). Some of these tools include a sufficient level of implementation details such as the flexibility of configuring hardware interrupts and timers. This in turn makes the tool more flexible by providing a wider range of options to design X-by-Wire systems. In addition, these tools are also equipped to perform the necessary timing analyses.

For instance, TimesTool – developed by Uppsala University, Sweden – allows the designer to perform schedulability analysis on real-time embedded designs (Amnell *et al.*, 2002). The tool is expected to assist in system modelling and analysis. The tool is capable of finding a suitable schedule and calculates the worst-case response time for a set of tasks. However, TimesTool currently only supports a single-processor system. Work on a multi-processor simulator is said to be in progress (Amnell *et al.*, 2003).

Unlike TimesTool, the Carosse-Perf tool (Castelpietra *et al.*, 2001) – developed by researchers at the LORIA French Research Laboratory in Computer Science – supports the simulation of distributed control systems. The tool was used effectively in a PSA Peugeot-Citroën application to evaluate the various response times for a distributed system which included the CAN and VAN network protocols (Castelpietra *et al.*, 2002). However, it remains unclear as to what extent this tool can be used to evaluate wider range of distributed control systems and network protocols. For instance, the tool does not mention the support of TDMA protocols like TTCAN, and TTP/C. Moreover, although the tool can calculate the relevant response times, it is still unable to address the performance of the complete control system. This is mainly because Carosse-Perf does not support the co-simulation of both the embedded processor and the plant dynamics.

Cheddar is another simulation tool which is similar to the Casosse-Perf approach. Cheddar was developed by researchers at University of Brest, France (Singhoff *et al.*, 2004). Their work was motivated by the lack of tools that could analyse and simulate a wide range of scheduling techniques.

Tools that support the co-simulation of the control process as well as the plant dynamics include Giotto (Henzinger *et al.*, 2003), AIDA (Redell *et al.*, 2004), RTSIM (Palopoli *et al.*, 2001; Palopoli *et al.*, 2002), and TrueTime (Cervin *et al.*, 2003). Most of these tools are general-purpose simulators that have the capability of simulating a wide range of X-by-Wire systems while taking into consideration the real-time implementation as well as the control aspects of the design.

For example, Giotto was created as a means to integrate the simulation of control system with the real-time concerns. To do this, the mathematical model developed on Simulink is separated to two components; the platform-independent functional and timing properties and the platform-dependent scheduling and communication issues. Finally, the Giotto compiler combines the two entities and generates the source code for the chosen platform.

In a similar vein, Redell and colleagues have developed the AIDA toolset to perform real-time analysis for control systems while taking into account the implementation aspect of the design. The AIDA toolset imports a control system design from the Simulink environment. It then models the real-time implementation by assigning various properties

to the control systems such as task delay, period, number of nodes, and communication strategy. The tool then analyses the response time for the system, and if satisfactory, exports the design with the relevant timing properties back to Simulink.

Another tool – RTSIM – has also been developed to assist with the design of real-time X-by-Wire systems. RTSIM uses a collection of C++ libraries to permit a separate specification of the functional behaviour of the controller and the hardware and software properties of the architecture. It then maps the two elements together and carries out the simulation and performance analysis.

Likewise, TrueTime was developed by the Department of Automatic Control, Lund Institute of Technology, Sweden, as a MATLAB toolbox capable of simulating real-time schedulers and network protocols (see Figure 2-5). TrueTime is intended to facilitate the co-simulation of controller task execution in real-time kernels, network transmissions, and continuous plant dynamics. The simulator supports various types of kernels, for example fixed-priority, deadline-monotonic, rate-monotonic and earliest-deadline-first. The kernels in turn support the use of interrupts and multiple A/D and D/A channels. Multiple network communication blocks can also be created, including CSMA/CA which is similar in behaviour to the CAN protocol (Kopetz, 1998; Hartwich *et al.*, 2000), and TDMA which is the basis of most time-triggered network protocols (e.g. see Kopetz, 2001).

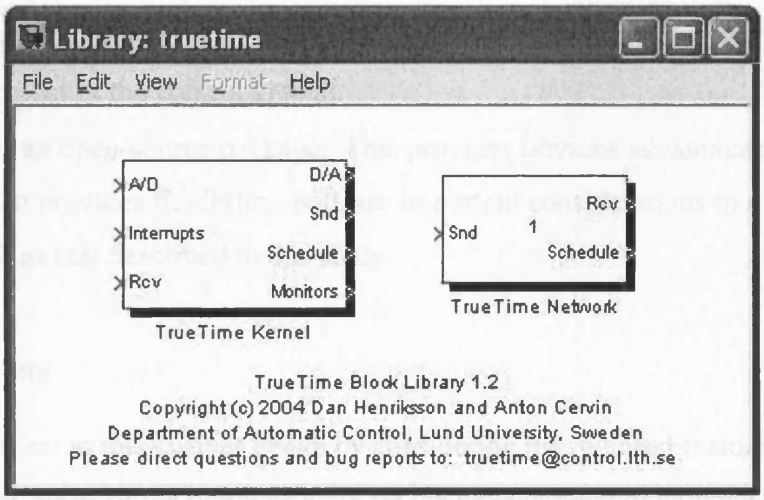

Figure 2-5 The TrueTime simulation library on Simulink.

2.4 Discussion

The requirements for a simulator and some existing simulation tools were described in Section 2.2 and Section 2.3 respectively. Clearly, it is preferable that the tool can perform the simulation at a sufficiently low level to closely emulate the behaviour of a real embedded processor. However, this – as discussed in Section 2.3.1 – poses other problems such as the difficulty to simulate complex X-by-Wire systems. By contrast, having high-level analytical models may lead to rather inaccurate solutions with stringent implementation options (see Section 2.3.2). It is concluded that the most suitable approach is the one where the tool is flexible enough to support the co-simulation of the real-time implementation aspects of embedded controller as well as the modelling of the plant dynamics.

To avoid reinventing the wheel, the TrueTime simulator was provisionally selected as a means of understanding the behaviour of X-by-Wire systems at the design stage.
The TrueTime simulator was chosen for the following reasons:

- TrueTime is capable of simulating the system to be controlled (that is, the plant dynamics) and a wide range of software architectures and network protocols for the distributed control system.
- Many control engineers are familiar with MATLAB and Simulink (e.g. see Dutton *et al.*, 1997; Dorf and Bishop, 2000). Since TrueTime is a MATLAB / Simulink package, this makes it easy to integrate the software and network design process with the development of the control system.
- TrueTime is an open-source package. This provides obvious advantages in terms of cost, and also provides flexibility: both are important considerations in a research project such as that described in this study.

2.5 Conclusions

The work described in this chapter began by considering the required features of a simulator. Then, some of the previous work on the development of suitable simulation tools for reliable X-by-Wire systems was described. TrueTime was provisionally chosen as a starting point to explore the use of simulation in the implementation of X-by-Wire systems. However, the choice of tool was made purely on a "desk" basis without the

backing of empirical evidence. It remains the case that not much experimentation has been conducted that can clearly demonstrate if TrueTime is capable of correctly predicting the behaviour a range of real embedded implementations. In order to verify this, the TrueTime simulator must be assessed beforehand to ensure that it works correctly. The evaluation process is described in Chapter 3.

3. Does the TrueTime simulator work?

In Chapter 2, TrueTime was provisionally selected as the simulator to be used in the remainder of the studies described in this thesis. The available literature suggests that TrueTime has the necessary potential to simulate a range of X-by-Wire systems. To confirm that TrueTime was appropriate for use in the present project, two small case studies were carried out. These studies, and the results obtained, are described in this chapter.[10]

3.1 Introduction

The initial aim described in this thesis was to develop a suitable simulation tool to support the development of X-by-Wire systems (see Section 1.2.3). A sensible starting point for this project was to first explore the available simulation tools (see Chapter 2). Based on the initial research, it was found that several tools were already available that support the simulation of X-by-Wire systems. Hence, in order to avoid reinventing the wheel, TrueTime was provisionally selected.

However, there is very little data available that can validate that TrueTime has been used to predict the behaviour of a range of X-by-Wire systems. For example, it remains unclear if TrueTime can support the simulation of (say) a Shared-Clock CAN (SCC) distributed system, or if the nature of event-triggered systems can be correctly modelled. There is also a lack of results for researchers to make a quantitative comparison between the measurements obtained from the TrueTime simulator with the actual hardware implementation. Overall, there is a lack of quantitative evidence to demonstrate that TrueTime can be efficient enough to support the simulation of X-by-Wire systems.

In light of the above considerations, the aim of the work described in this chapter was to investigate the extent to which a general purpose simulator (such as TrueTime) can be used to simulate X-by-Wire systems. To do this, two case studies were employed: (1) a cruise-control system for a passenger car and (2) a control system for an inverted pendulum. These systems were simulated on TrueTime first, and the results compared

[10] Parts of this chapter have previously been published in different forms in Ayavoo *et al.* (2004); Ayavoo *et al.* (2005c) and Ayavoo *et al.* (2006).

with the actual measurements obtained from a hardware testbed. These case studies and their results are discussed in detail in the remainder of this chapter.[11]

3.2 Case Study 3A: The Cruise-Control System

Case Study 3A was used to evaluate the extent to which TrueTime can predict the behaviour of a non-trivial embedded control system. This study involved the design of an automotive cruise-control system (CCS). A description of the case study is presented in this section.

3.2.1 The CCS testbed

An automotive CCS provides the driver with an option of maintaining his or her vehicle at a desired speed without further intervention, by controlling the throttle (accelerator) setting (Heintz, 1990). Such a driver assistance system can reduce the strain on the driver especially while travelling on long journeys. This type of CCS was chosen as the basis for the present case study as it represents a non-trivial embedded control system which is in widespread use in the automotive industry (Clarke, 1998).

An automotive CCS will typically have the following features (Kureemun, 1999; Sanz and Zalewski, 2003):

1) An ON / OFF button to enable/disable the system.
2) An interface through which the driver can change the vehicle's set speed while cruising.
3) Switches on the accelerator and brake pedals that can be used to disengage the CCS and return control to the driver.

For the purpose of the study described here, the specification of the CCS was simplified such that the vehicle was assumed to be always in "cruise" mode. While in cruise mode, a "speed dial" was available to allow the driver to dynamically change the car speed. The embedded control process ensured that the vehicle would travel at the desired (set) speed.

---

[11] Please note that an additional case study that also compares the simulated results and the hardware implementation for a six-node X-by-Wire system was carried out in Ayavoo *et al.* (2005a). This study is not included in this chapter, but the results obtained are presented in Appendix A.

In this study, a computational model was used to represent the environment in which the CCS operates. This car environment model had one input (current throttle) and one output (a train of pulses representing the speed of the car). Figure 3-1 illustrates this.

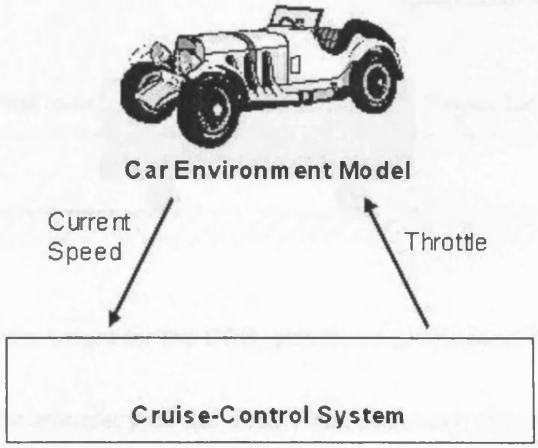

Figure 3-1 A basic cruise-control system for Case Study 3A (adapted from Ayavoo *et al.* 2004, Figure 1).

The core of the car environment simulation was a simplified physical model based on Newton's laws of motion. First the instantaneous acceleration of the vehicle was calculated (Equation 1). Once this acceleration was obtained, the new speed of the car was then calculated (Equation 2).

$$a = ((\theta\tau) - (v_i Fr))/m \qquad (1)$$

$$v_f^{\,2} = v_i^{\,2} + 2a\Delta x \qquad (2)$$

a	Acceleration
Fr	Frictional coefficient
m	Mass
v_f	Final speed
v_i	Initial speed
Δx	Displacement
θ	Throttle setting
τ	Engine torque

Please note, that in this case, it was assumed that the vehicle was under the influence of only two forces, the torque exerted on the car engine and the frictional force that acts in the opposite direction to the motion (see Figure 3-2). The engine torque was assumed to be constant over the speed range. These models can clearly be made more realistic (for

example, see Short *et al.*, 2004a; Short *et al.*, 2004b), but - for the purpose of this study - this simplified model was sufficient.

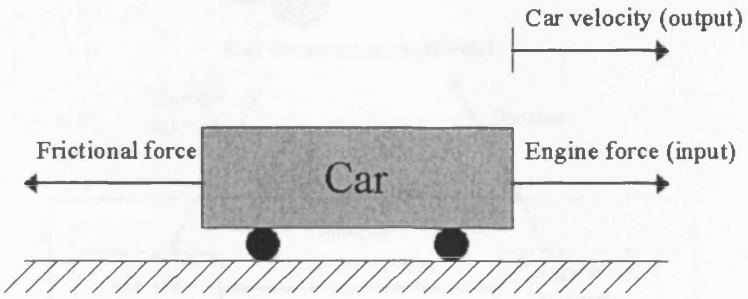

Figure 3-2 The car environment for the CCS (adapted from Ayavoo *et al.* 2005c, Figure 2).

In order to investigate the accuracy of the TrueTime software simulation process, a Hardware-in-the-Loop (HIL) testbed was used to implement the X-by-Wire system. In this case, a network of Infineon 16-bit microcontrollers (Phytec C167CS/CR development boards) was used to implement the CCS: such devices are widely used in the automotive sector (Siemens, 1996).

The car environment was implemented on a basic desktop PC (Intel Pentium II 300 MHz processor). The use of a PC hardware for such studies have several advantages that includes cost effectiveness and flexibility (see Pont *et al.*, 2003).

A detailed description of the HIL design and implementation of the CCS is illustrated in Appendix C.

3.2.2 Implementation of the distributed CCS

In the present case study, the CCS was designed to operate as an X-by-Wire system consisting of two nodes: a sampler node and a controller & actuator (CA) node (Figure 3-3).

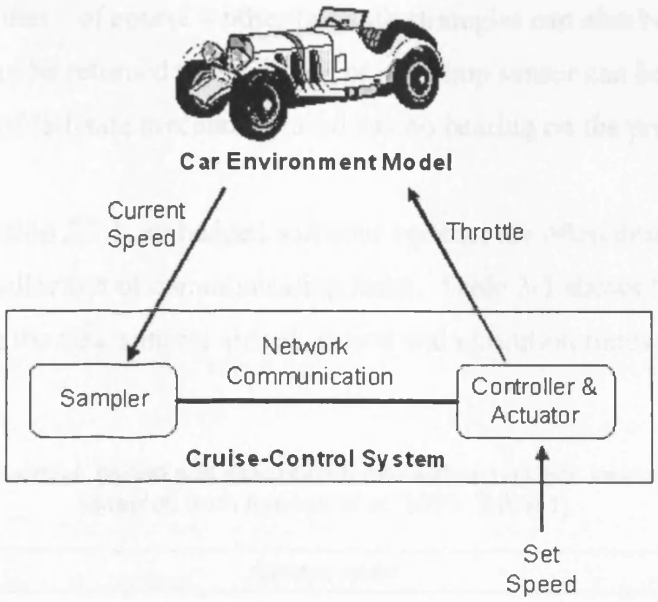

Figure 3-3 A distributed cruise-control system for Case Study 3A (adapted from Ayavoo *et al.* 2004, Figure 2).

The sampler node was used to calculate the vehicle speed: this is assumed to be carried out by counting pulses from an optical/magnetic sensor (which is usually attached to one of the vehicle's wheels). Some noise filtering and the necessary scaling were also performed on this node. The calculated car speed was then sent over a network to the CA node. On the CA node, a Proportional, Integral and Derivative (PID) algorithm was used to calculate the required throttle position, which was then sent back to the car environment model. The CA node was also responsible for obtaining the required "set speed" value (from the driver).

Besides the basic computations of sampling, controlling and actuation, the CCS must also be capable of detecting and handling any software or hardware errors in an appropriate manner. For example, in a safety-critical system where a sensor failure occurs and no error management is performed, the result may be a cruise speed which is much higher than desired. With appropriate error detection and management techniques, the likelihood of such problems can be greatly reduced. In the CCS study, a failure of the speed sensor was simulated, and the impact of this failure on the performance of the different design options was compared. In each case, it was intended that the sensor failure can be detected, and that the vehicle can be brought to a halt "as quickly as possible" (by setting the throttle to 0). This mechanism of error handling was felt to be adequate for this initial

study. Please note that – of course – other fail-safe strategies can also be applied: for example, control can be returned to the driver, or a backup sensor can be deployed. However, the type of fail-safe mechanism used has no bearing on the present study.

As discussed in Section 2.2.1, embedded software systems are often designed and implemented as a collection of communicating tasks. Table 3-1 shows the set of tasks to carry out, including the task's initial arrival, period and execution times on the sensor and the CA nodes.

Table 3-1 Task initial arrival, period and execution times with a 1ms tick interval for Case Study 3A (adapted from Ayavoo *et al.* 2004, Table 1).

Sensor node				
Priority	Task initial arrival (in ticks)	Task period (in ticks)	Task description	Task execution time (in µs)
1	0	1	Check Sensor Failure	7
2	0	50	Compute Speed	46

Controller/Actuator node				
Priority	Task initial arrival (in ticks)	Task period (in ticks)	Task description	Task execution time (in µs)
1	0	1	Indicate Sensor Failure	6
2	1	50	Compute Throttle	168
3	0	1000	Get Ref Speed	38

Choosing the software architecture and communication technique of the described CCS was not a trivial process even for this simple two-node system. This is because the choice of implementation can have an impact on the overall system performance (see section 2.2).

Here, four different implementation options were compared (see Table 3-2). In all cases, the nodes were linked using a CAN bus running at 333.3 kbps. The CAN bus was used as it provides high-reliability communications at low cost (Fredriksson, 1994; Sevillano *et al.*, 1998; Thomesse, 1998; Farsi and Barbosa, 2000). Moreover, since the CAN protocol is widely used in many sectors (see Section 2.2.3), most modern microprocessor families now have members with on-chip support for this protocol (e.g. Philips, 1996; Siemens, 1997; Infineon, 2004; Philips, 2004).

Table 3-2 Combination of the possible implementations for the CCS in Case Study 3A (adapted from Ayavoo *et al.* 2004, Table 2).

	Sampler Node	Communication	CA Node
"T-T-T"	Time	Time	Time
"T-E-T"	Time	Event	Time
"E-T-E"	Event	Time	Event
"E-E-E"	Event	Event	Event

In the "T-T-T" system, the scheduling on both nodes was time-triggered, where the tasks were scheduled to execute periodically. The network protocol was also time-triggered. Please note that although CAN networks are generally treated as event-triggered (Leen and Heffernan, 2002), some previous work has demonstrated that CAN can also be used in a time-triggered fashion by employing the Shared-Clock CAN (SCC) strategy (see Pont, 2001; Ayavoo *et al.*, 2005b). Here, a "tick" message was sent from the sensor node at the beginning of every sensor node "tick". This message was used to synchronise the CA node. The sensor status and the car speed data were also included in this message. An acknowledgement message from the CA node was then sent back to the sensor node. Please note that this is more complicated than it may appear. These techniques are described more thoroughly in Appendix B.

In the "T-E-T" system, the scheduling policy on both of the nodes was time-triggered but the network protocol was event-triggered (fixed priority). The scheduling of all the tasks was similar to the "T-T-T" system, the difference being that messages were sent at the end of the task execution, instead of in pre-determined time "slots".

In the "E-T-E" system, the scheduling policy on the nodes was event-triggered but the network communication was TDMA. Again (because of the nature of the control system) most of the tasks were executed periodically. However, tasks "Check Sensor Failure" (on the sensor node) and "Indicate Sensor Failure" (on the CA node) were triggered (asynchronously) via interrupts. Because of the TDMA protocol, messages were exchanged between the nodes periodically, at predetermined times.

In the "E-E-E" system, the scheduling on the nodes was event-triggered and the network communication was event-triggered. The task execution on both the nodes was similar to

the "E-T-E" system, but in this case messages were sent at the end of the task execution, instead of having predetermined time slots.

3.2.3 Measurements for Case Study 3A

In this study, the focus of the measurements was on the following aspects of the system performance:

1) Control performance.
   This is the measure of the effectiveness of the CCS in maintaining the vehicle speed at the desired value.
2) Event response time.
   In the CCS, the event response time is a measure of the time between the detection of sensor failure (on the sensor node) and the reaction to the failure (on the CA node).
3) Control delay.
   For the CCS, this is a measure of the time between the measurement of the current vehicle speed and the application of a new throttle setting.
4) Control jitter.
   This is a measure of the variations in start times of the periodic control tasks.

It was to be expected that all of these measures will be affected to a greater or lesser extent, by the implementation options such as type of scheduling policies used (for example, time-triggered or event-triggered) as well as the choice of network protocols (for example, TDMA, or fixed-priority).

3.2.4 Results for Case Study 3A

Having carried out the experiment (on both the software simulation and hardware implementation), this section reviews the results obtained for each of the measurements discussed in Section 3.2.3.

The control performance of all systems was found to be very similar. As an example, Figure 3-4 shows the results from the "E-E-E" system. The car reference speed was set to be at 30 m/s initially and then changed to 45 m/s. It can be seen from Figure 3-4 that the CCS was able to follow the desired (set) speed in both the TrueTime simulator and the HIL testbed.

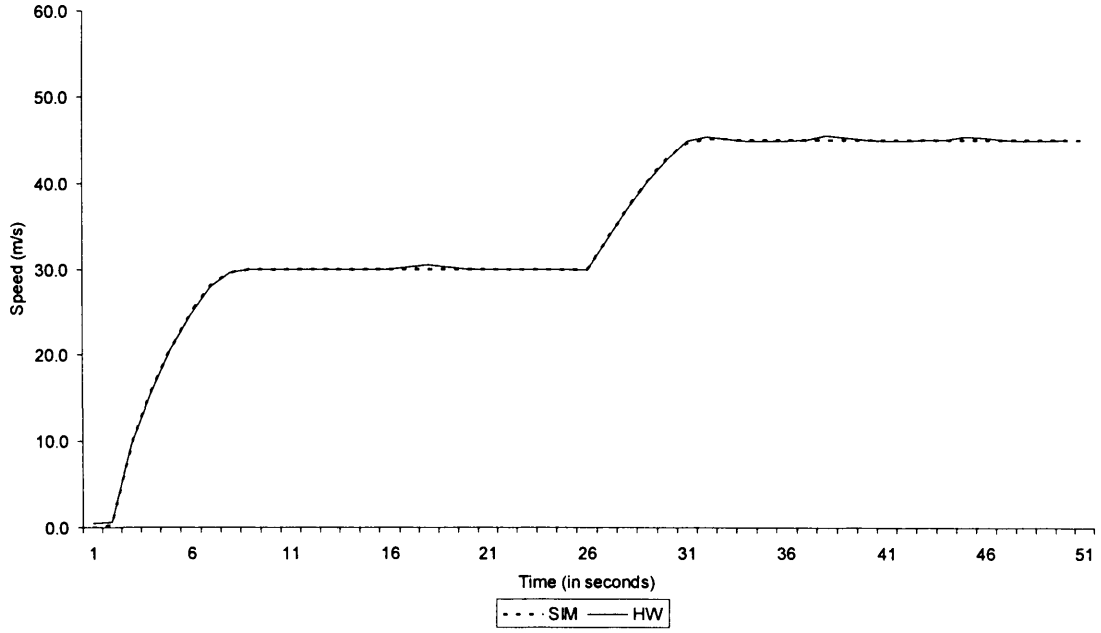

Figure 3-4 Control performance using the TrueTime simulator and HIL testbed for Case Study 3A (adapted from Ayavoo *et al.* 2004, Figure 3).

Table 3-3 shows the recorded event response times for the simulation and HIL systems.

Table 3-3 Comparison of TrueTime simulation results and the HIL testbed of an "event message" response time (in μs) for Case Study 3A (adapted from Ayavoo *et al.* 2004, Table 3).

	Minimum		Maximum	
System	HIL	SIM	HIL	SIM
"T-T-T"	1384	1384	2383	2384
"T-E-T"	419	1000	2243	2000
"E-T-E"	392	391	1390	1384
"E-E-E"	384	384	391	496

The results show that, for the xTx systems, the simulation match the results from the HIL testbed very closely (within 1%). The match between the simulated and HIL results for the xEx systems are less close. This was due to the fact that the different boards (in the HIL design) had independent crystal oscillators, which moved out of step. This possibility was not taken into account in the simulator used in this study. Further discussion on this issue is presented in Appendix F.

Table 3-4 displays the control delay for the two systems that had employed a time-triggered approach for the network communication. In this case the measurements were

taken from the time the "Compute Speed" task started (on the Sampler node) until the "Compute Throttle" task finished executing (on the CA node).

Table 3-4 Comparison of control delay (in μs) between the TrueTime simulator and the HIL testbed for Case Study 3A (adapted from Ayavoo *et al.* 2004, Table 4).

		HIL	SIM
TTT System	Minimum	1388	1373
	Maximum	1398	1373
	Mean	1393	1373
ETE System	Minimum	1390	1381
	Maximum	1400	1381
	Mean	1395	1381

Once again, the simulation results for the xTx systems followed those from the HIL testbed quite closely (the mean error is less than 2%).

Table 3-5 shows the percentage of deviation between the TrueTime simulator and the HIL testbed for two periodic control tasks ("Compute Speed", on the sensor node and "Compute Throttle" on the CA node).

Table 3-5 The percentage of mean deviation between the TrueTime simulator and the HIL testbed for the periodic control tasks in Case Study 3A (adapted from Ayavoo *et al.* 2004, Table 5).

	Task Compute Speed	Task Compute Throttle
T-T-T System	0.002	0.002
T-E-T System	0.002	0.006
E-T-E System	0.002	0.002
E-E-E System	0.002	1.789

The results show that deviation between the simulated and HIL values in the "E-E-E" system were slightly larger than those from the other systems. As before, this was caused by clock drifts in the HIL testbed.

3.3 Case Study 3B: The inverted pendulum system

In Section 3.2, Case Study 3A demonstrated that the TrueTime simulator can closely match the results of the hardware implementation for an automotive cruise-control system. To substantiate this claim for a wider range of studies, Case Study 3B was carried out.

In Case Study 3B, a real control problem was used to evaluate the TrueTime simulator, instead of a "simulated" HIL system. The testbed employed in this case was based on the control of an inverted pendulum. A description of Case Study 3B and its results are presented in this section.

3.3.1 The inverted pendulum testbed

An inverted pendulum is an inherently unstable system, and the objective of the control system is to balance the rod at an upright position. Previous work has discussed ways in which an inverted pendulum may be used as an effective testbed for experimenting with different design options involving embedded control systems (Edwards *et al.*, 2004; Bautista *et al.*, 2005; Bautista and Pont, 2006). In addition, it has also been argued that this testbed can be suitable to experiment with future automotive X-by-Wire systems (see Edwards *et al.*, 2004).

In this study, the inverted pendulum testbed used (see Figure 3-5) was custom made in the ESL. The length of the track was 0.9 metres. The pendulum (or rod) weighed 0.05 kilograms with a height of 0.305 metres. A brushed DC motor with integrated gearbox was used to move the cart along the track. A pulse encoder attached to the motor was used to measure the position of the cart. Another encoder was mounted at the base of the rod to measure the angle of the rod. More details of the testbed can be obtained from Bautista *et al.* (2005).

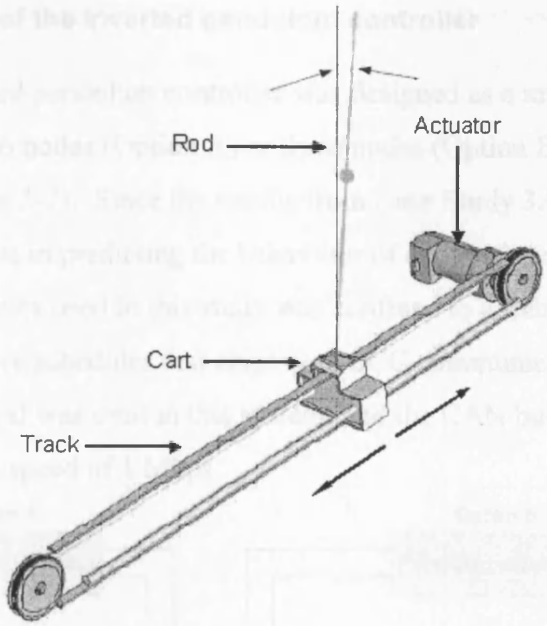

Figure 3-5 The testbed of an inverted pendulum for Case Study 3B (adapted from Bautista *et al.*, 2005, Figure 1).

In the setup used in this study, the inverted pendulum testbed produced two outputs: the position of the cart and the angle of the rod. The rig also required a signal that represented the speed of the cart and another signal that indicated the direction of the cart as inputs into the motor. To control the rod such that it remained balanced at the middle position, a 32-bit ARM7 microcontroller (Philip's LPC2129) was used (see Figure 3-6). The microcontroller reads in the position of the cart and the angle of the rod, and used these values in a control algorithm to calculate the required control value. This value was then translated to a Pulse Width Modulation (PWM) output, which in turn enabled the motor to move the cart at the required speed and direction.

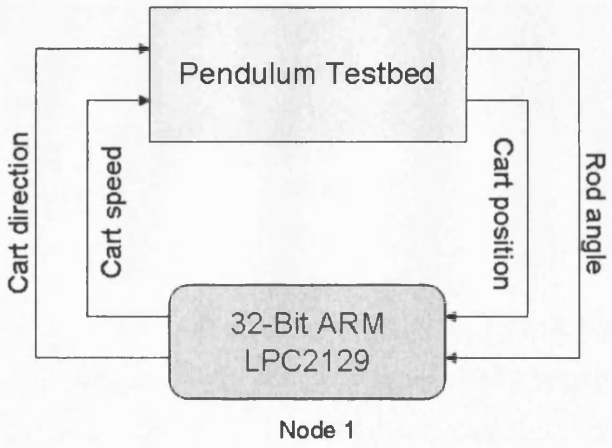

Figure 3-6 The input/output of the pendulum testbed (plant) and the ARM microcontroller for Case Study 3B. Figure adapted from Ayavoo *et al.* 2006, Fig. 2.

3.3.2 Implementation of the inverted pendulum controller

In this study, the inverted pendulum controller was designed as a multiprocessor system, comprising of either two nodes (Option A) or three nodes (Option B) to represent an X-by-Wire system (see Figure 3-7). Since the results from Case Study 3A shows that the simulator is less efficient in predicting the behaviour of event-triggered systems, the implementation techniques used in this study was confined to a time-triggered solution. To do this, a co-operative scheduler that employs a SCC communication protocol was used. A 5ms tick interval was used in this system, and the CAN bus was configured to operate at its maximum speed of 1 Mbps.

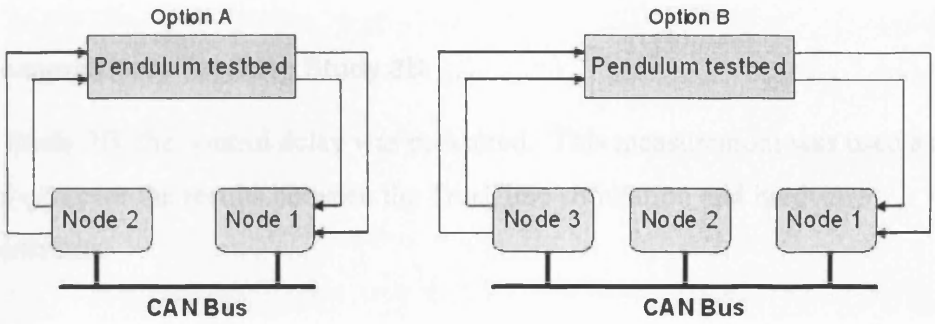

Figure 3-7 Distributed implementation options for the pendulum control in Case Study 3B.

Table 3-6 illustrates the set of tasks related to the pendulum controller. For a two-node system, Node 1 was the sensor node while Node 2 carried out the control and actuation process. For a three-node system, the sensor, controller and actuator were assigned to Node 1, Node 2 and Node 3 respectively. In all cases, a heartbeat LED task was used on each microcontroller to indicate the status of the board.

Table 3-6 Task structure of the pendulum controller for Case Study 3B.

Task Names	Task Description	Task Period (in ms)
Obtain sensor values	Obtains the cart position and rod angle from the sensors	10
Calculates control value	Calculates the required control value using an "LQR" algorithm	10
Actuates PWM output	Actuates the PWM output that determines the speed and direction of the cart	10
Checks cart track safety	Checks that the cart does not hit the boundaries of the track	10
Heartbeat LED update	Periodic flashing of an LED to indicate that the system is "alive"	1000

3.3.3 Measurements for Case Study 3B

In Case Study 3B, the control delay was measured. This measurement was used as a means to compare the results between the TrueTime simulation and hardware implementation.

3.3.4 Results for Case Study 3B

Table 3-7 illustrates the results of the control delay for the various implementations.

Table 3-7 Comparison of the control delay (in ms) between the TrueTime simulation and implementation for Case Study 3B.

	SW SIM	HW IMP
One-node min	6.467	6.492
One-node max	7.246	7.269
Max - Min	0.779	0.777
Two-node min	16.307	16.321
Two-node max	17.086	17.090
Max – Min	0.779	0.769
Three-node min	20.018	19.834
Three-node max	20.018	19.845
Max - Min	0.000	0.011

The results indicate that the TrueTime simulation can closely predict the behaviour of the control delay for the one-node, two-node and three-node implementations of the inverted pendulum controller.

3.4 Discussion and conclusions

The results from Case Study 3A suggest that the TrueTime simulator is capable of predicting the behaviour of the actual hardware implementation very closely. In particular, this was the case for distributed systems that employed a time-triggered approach for its message communication. For systems that used an event-triggered communication approach, the results of the TrueTime simulator were less reliable. This was, in part, due to the lack of support for individual clock drifts on the current version of TrueTime (V1.2) used in this study (see Appendix F).[12]

In Case Study 3B, a real hardware testbed was employed as opposed to a simulated HIL system. The results of the study again indicate that the TrueTime simulator was capable of predicting the behaviour of a real X-by-Wire system.

Overall, the case studies presented in this chapter show a close match between the behaviour of the embedded systems simulated on TrueTime and the corresponding measurements obtained from the hardware testbeds. Of course, the TrueTime simulator is imperfect: for example, the presence of independent crystal oscillators in the two processor nodes was not taken into account (see Appendix F for further discussion on the limitations of TrueTime). Nevertheless, the results obtained suggest that the TrueTime simulation technique is an effective way of predicting the performance of X-by-Wire systems.

Of course, although it has been necessary to verify that the TrueTime simulator works, this condition alone is not enough to demonstrate the full potential of the tool. This issue is discussed in more detail in Chapter 4, in conjunction with suggestions for techniques that could be used to carry out further evaluations.

---

[12] Please note that at a much later time after these studies were conducted, the Department of Automatic Control at Lund Institute of Technology, Sweden, introduced a newer version of the TrueTime simulator (V1.3) which supports clock drifts.

4. How can we answer more specific questions about the role of simulation?

The initial aim of this research was to develop a simulation tool to support the development of X-by-Wire systems. However, in Chapter 3, it was demonstrated through two case studies that the available TrueTime simulator is capable of predicting the implementation behaviour of X-by-Wire systems. The work presented in this chapter discusses other ways in which the simulation approach could be further assessed.[13]

4.1 Introduction

As briefly discussed in Chapter 1, the initial aim of the work described in this thesis was to develop a simulation tool to support the development of X-by-Wire systems. As discussed in Chapter 1, it might be expected that the use of an effective simulator can assist in the development of such systems. However, the contributions made by simulation towards the development process are rarely explored in depth and is incompletely understood. Indeed, there is very little empirical data available that can demonstrate the efficacy of such a simulator.

This lack of empirical evidence is not, of course, a problem unique to the use of simulators and the software engineering community is becoming increasingly aware of the need to seek evidence of any form of new technology (Oman and Pfleeger, 1997; Pickard *et al.*, 1998; Wood *et al.*, 1999; Endres and Rombach, 2003), rather than relying on sweeping statements about its "obvious" effectiveness (e.g. see Turski, 1986; Fenton *et al.*, 1994). Resorting to bad practices such as gut feelings, intuition or analytical advocacy has sometimes led to the popular belief that a particular method is useful, when in fact, subsequent empirical research seems to show otherwise (Basili *et al.*, 1999; Dyba *et al.*, 2005). For example, Fenton *et al.* (1994) has previously described some studies that have contradicted the widely held beliefs on techniques like O-O and formal methods.

With respect to a "simulation first" methodology which is the focus of this thesis, evidence is also essential to evaluate its efficacy in the development process of future X-by-Wire systems. For example, in Chapter 3, evidence was presented to suggest that the TrueTime simulation can closely match the behaviour of X-by-Wire systems. However, fulfilling

[13] Parts of this chapter also appears in Ayavoo *et al.* (2006) and Ayavoo *et al.* (submitted).

this condition alone – clearly – does not mean that a "simulation first" approach is effective. On the contrary, the work presented in Chapter 3 has only described a necessary, but insufficient condition to demonstrate the overall effectiveness of the tool.

In Chapter 1, the other factors that affect the efficacy of a simulator were briefly discussed. Researchers have suggested that a "simulation first" approach should – for example – reduce the development effort involved (Castelpietra *et al.*, 2002; Henzinger *et al.*, 2003; Redell *et al.*, 2004), or improve the quality and reliability of the system (El-khoury and Törngren, 2001; Cervin *et al.*, 2003; Henzinger *et al.*, 2003). Yet, in order to substantiate these claims, evidence that can demonstrate the effectiveness of the tool in use (with regards to effort or quality) is required. Unfortunately, such evidence is scarce. Indeed, to date, very little work has been carried out to gather evidence to indicate that a "simulation first" approach is going to be effective when used in practice.

According to Endres and Rombach, "*Rather than developing theories, they [theorists] frequently argue in favour of specific methods and tools, many of which demonstrate technical virtuosity. They often do this without sufficient regard to their usefulness in practice nor after validating them by meaningful trials. Practice should be the true measure of a method's usefulness* ..." (Endres and Rombach, 2003, p. 1). Putting this in context of the current research, the more interesting question is essentially how useful the TrueTime simulation approach is going to be when it is used in practice to develop X-by-Wire systems. Hence, to verify the true usefulness of the approach, more specific questions need to be answered.

Based on these discussions, the focus of this chapter is on the use of empirical investigations as a vital process in gathering evidence on the efficacy of any new technology in software engineering (see Oman and Pfleeger, 1997; Basili *et al.*, 1999; Endres and Rombach, 2003). However, carrying out empirical software studies is not straightforward. Some of these challenges are illustrated in Section 4.2.

With respect to X-by-Wire system development, the challenges faced in the empirical software studies are slightly unique. These problems are discussed in detail in Section 4.3. In order to address some of these issues, a low-cost empirical methodology is proposed in

Section 4.4. For ease of reference, the technique is referred to as the "Small Group Methodology" (SGM).

4.2 Challenges of empirical software studies

Carrying out empirical software assessments and gathering the necessary evidence can be notoriously difficult (Fenton *et al.*, 1994; Basili *et al.*, 1999). Although the work presented in this thesis is not intended to address all of these problems, it is useful to appreciate some of these challenges faced by researchers in this area.

One of the difficulties is caused by the dependency on human-based studies (Basili and Reiter, 1979; Sheil, 1981; Seaman, 1999; Lethbridge *et al.*, 2005). Since the skill of a software engineer is often involved in most software development processes, it is crucial that the interaction of the engineer with a particular development methodology is taken into account in the analysis (Brooks, 1980). This is important because the likelihood of success or failure may depend on the way the methodology is applied by the software engineer. However, identifying these characteristics is not always straightforward and may be dependent on the way that the experiment is designed.

For instance, a previous study on the use of flowcharts in programming suggested that such charts do not assist the programmer in comprehending the documentation any more than pseudocode (Shneiderman *et al.*, 1977). Subsequent research later suggested that this may not necessarily be the case (Scanlan, 1989). The difference of outcome in these two research publications, at the very least, illustrates the construction of a suitable experiment that involves human subjects can be extremely difficult.

Such studies are also very often complex and time consuming (Fenton *et al.*, 1994). For example, depending on the nature of the study, it can be difficult to identify and control all the influential variables that may affect the outcome of the empirical software experiment (Fenton *et al.*, 1994; Pfleeger, 1999). The experiment may also require much effort in planning and the experimentation process itself may take a long time. These factors also tend to influence the cost involved in such studies, making it very difficult to conduct large-scale empirical experiments. Drawing a general conclusion from empirical studies is also not easy. Very often, the studies are conducted within certain constraints, and

therefore the results cannot be generalised for all circumstances (see Basili *et al.*, 1999; Pfleeger, 1999).

Nevertheless, some researchers have attempted to provide some rules and strategies on how to prepare and carry out these complex studies (Brooks, 1980; Basili *et al.*, 1986; Kitchenham *et al.*, 2002; Lethbridge *et al.*, 2005).

For example, Brooks (1980) discussed the various problems involved in the selection of subjects, design of the experiment and choice of measurements for empirical software studies. Although various solutions have been proposed, each one has its own drawback and no universally accepted technique is available.

To help carry out better empirical studies, some researchers had proposed a framework of the various stages involved such as project definition, planning, operation and interpretation (Basili *et al.*, 1986; Kitchenham *et al.*, 2002). However, the variation caused by the different goals, methodology, experience, problem domain and constraints of the experimental environment can still affect the techniques used at the various stages of the empirical study. For example, the cost of the study may influence some of the techniques used at the operation stage, or limitations in the personnel may influence the way the analysis is carried out.

Recently, Lethbridge *et al.* (2005) described various techniques of data collection for empirical software studies. A discussion was also provided on the pros and cons for each technique. In spite of this, not all the techniques are applicable in all empirical studies. For example, the conceptual modelling technique that requires the participant to conceptualise their mental thoughts on paper might not be suitable as it may force a particular development methodology on the test subjects. Likewise, the use of "think aloud" techniques that require the test subjects to "verbalise" their thoughts, may be unnatural for the participants and may cause other side effects on the measurement process.

Overall, carrying out empirical studies is difficult and no universally accepted technique is available. Hence, it remains the case that the approach used to carry out empirical

experiments in software engineering is strongly dependent on the characteristics of the study.

4.3 Towards a "small group" methodology

One way that researchers have found to be effective in carrying out empirical investigation for a range of software engineering fields, is through the use of laboratory studies in conjunction with techniques to observe the process of software development (see Robillard *et al.*, 1998; Henry and Stevens, 1999; Seaman, 1999; Marcias *et al.*, 2002; Marcias *et al.*, 2003; Germain and Robillard, 2005).

4.3.1 Large industrial studies

In this thesis, the focus is on the comparison of different development methodologies for automotive X-by-Wire systems. In general terms, it is clear how researchers could conduct an effective scientific study in this area. For example, the study could employ (say) 50 different groups of experienced developers each working on a different design for a Steer-by-Wire system: the researcher might then argue that – after analysing the results – they would be able to identify the "best approach".

Sometimes such studies can be carried out effectively. Indeed, various researchers have used an industrial site as their base for collecting data (see Shepperd and Schofield, 1997; Subramanian and Corbin, 2001; Subramanyam and Krishnan, 2003). These studies involved the analysis of software development data obtained through a large team of professional software engineers working on site.

As an example of a study using industrial data, Subramanyam and Krishnan (2003) analysed the implications of O-O design complexity metrics on software defects. Industrial data was used here because the research required a huge amount of data to investigate the effects of software metrics and the different analysis techniques that could be employed on the data obtained (see also Shepperd and Schofield, 1997; Subramanian and Corbin, 2001). In general, these studies did not require a change in the software development methodology employed by the respective industries. More importantly,

these experiments have not been designed to compare different software development approaches.

Industrial data have also been useful for studies that require longitudinal data (data spread across several years). As noted by Cook and colleagues, there is much benefit to be reaped from analysing in-place software processes (Cook *et al.*, 1998), and this has been shown effectively by Kemerer and Slaughter (1999), where the evolution of software over 20 years in a particular industrial site was explored.

In the case of embedded systems, technology changes very rapidly (in periods measured in months or years, not decades); hence, the turn around time for empirical results must be equally fast. In addition, since it is intended that the benefits of new technologies are explored, experiments need to be carried out with at least two groups, operating in controlled conditions with little or no communication between the members of different groups. Finding significant numbers of suitable subjects for empirical studies is often seen as problematic (Pickard *et al.*, 1998; Basili *et al.*, 1999): adding a requirement for a controlled environment also tends to mitigate against the use of a company setting for the type of studies required in this thesis.

4.3.2 Using students as test subjects

Faced with the challenges outlined in the previous section, one option would be to use students as subjects, and to conduct controlled experiments in a university (or similar) setting. Clearly, this raises a number of questions, not least the fact that the interest (in this case) is the impact of new technologies and methodologies on professional developers rather than students. While some researchers have argued that there are only minor differences between the students and professionals (see Holt *et al.*, 1987; Höst *et al.*, 2000), others have argued differently (see Brooks, 1980; Arisholm and Sjöberg, 2004). One must therefore ask if studies with students can yield useful data.

For example, in the Arisholm and Sjöberg (2004) study, the researchers found that the level of correctness and effort involved in maintaining a particular software program varied between senior professional engineers and undergraduate students. However, the criteria used to select the students and the professional engineers were ambiguous. It

could be the case that some of the student subjects selected for that study were not very good, or the distribution of the students was not equally balanced. Indeed, the study showed that there were cases where the results of some students were similar to the professional engineer. This may suggest that student studies are still possible, if the subjects are selected with equal backgrounds and abilities.

Certainly, the available data does suggest that student studies are capable of generating useful results. As quoted by Kitchenham and colleagues, "*Some practitioners may feel the use of student subjects in formal experiments reduces the practical value of the experiments. In our view, this is not a major issue as long as you are interested in evaluating the use of a technique by novice or non-expert software engineers. Students are the next generation of software professionals and, so, are relatively close to the population of interest.*" (Kitchenham *et al.*, 2002, p. 732).

Students have previously been used in a number of empirical studies (see Robillard *et al.*, 1998; Henry and Stevens, 1999; Prechelt and Unger, 2000; Marcias *et al.*, 2002; Sobel and Clarkson, 2002; Marcias *et al.*, 2003; Germain and Robillard, 2005). For example, Prechelt and Unger (2000) have successfully showed the effects of Personal Software Process Training on software development. In a different study, Sobel and Clarkson (2002) have explored the benefits of using formal methods in software development using students. Henry and Stevens have also conducted a useful study on different team structures in software projects using only student subjects and have concluded that the results obtained can be more widely applied: "*This research provides guidance to managers in forming successful teams...*" (Henry and Stevens, 1999, p. 248 – 249).

4.3.3 Large student studies

Having decided that students can be used to yield useful data, Sobel and Clarkson (2002) used 20 groups (40 people) to study the impact of formal methods on O-O software development. Here, a large number of students were easily employed because the test case used in the study was integrated with the teaching syllabus. Large numbers of student test subjects have also been employed by several other researchers (see Henry and Stevens, 1999; Prechelt and Unger, 2000).

Large numbers of student test subjects were available for such empirical experiments for two main reasons. First, the subjects for the studies were already exposed to a necessary "treatment" as a result of having taken previous course modules prior to the experiment: for example, in the Sobel and Clarkson study, half the class had already taken a course in formal methods, the other half had not. Secondly, such experiment can often be designed in such a way that these students participate as part of the normal running of the module: for example, the experiment conducted by Germain and Robillard (2005) was part of an optional course offered to senior students in computer engineering. This similar scheme has also been employed in other studies (see Holcombe *et al.*, 2001; Marcias *et al.*, 2002; Marcias *et al.*, 2003).

Given the rate of change in the field of embedded implementation for automotive systems (see Section 1.2.1), neither of these two conditions can be easily satisfied in this area of research. Therefore, empirical research in this field needs to rely on custom-made studies: if such studies are conducted with students, these have to usually take place during university vacations. Such studies have advantages, not least because a much wider range of hypothesis can be tested in a controlled study of this nature. However, as such studies rely on the use of volunteers, the numbers of subjects will be greatly reduced.

4.3.4 Small student studies

Although studies employing large number of students have been successful, it is extremely unlikely that any institute would consider employing such an approach, primarily on grounds of cost (Ciolkowski *et al.*, 2003). By contrast, various successful studies have been carried out in the past using small numbers of subjects.

For example, Robillard *et al.* (1998) carried out an empirical study by observing the meetings of four full-time software engineers participating in a professional software development project. In a different empirical study, five student subjects were employed instead of professional engineers (Robillard *et al.*, 2004). Several other researchers have also used a small number of test subjects in their study (see Fitter and Cruickshank, 1983; Vessey and Conger, 1994; Dawson and Swatman, 1999). In fact, there is a study that used only one test subject due to the difficulty in obtaining suitable candidates for the study (Strong, 1995). Despite this – very – small sample size, the study succeeded in providing

useful insight on effective design principles for speech prosthesis by observing the human-computer interaction.

4.4 The "Small Group Methodology"

The "Small Group Methodology" (SGM)[14] builds on the findings from previous successful studies in which small numbers of subjects have been employed. Such an approach has obvious advantages in that it reduces costs, and makes it possible to conduct research when only limited numbers of volunteers are available.

A set of "good practices" from previous studies was tailored to suite the current experimental approach that involves the development of embedded systems. An overview of the SGM is given in Figure 4-1. A detailed discussion on the six key phases of the SGM is presented in this section.

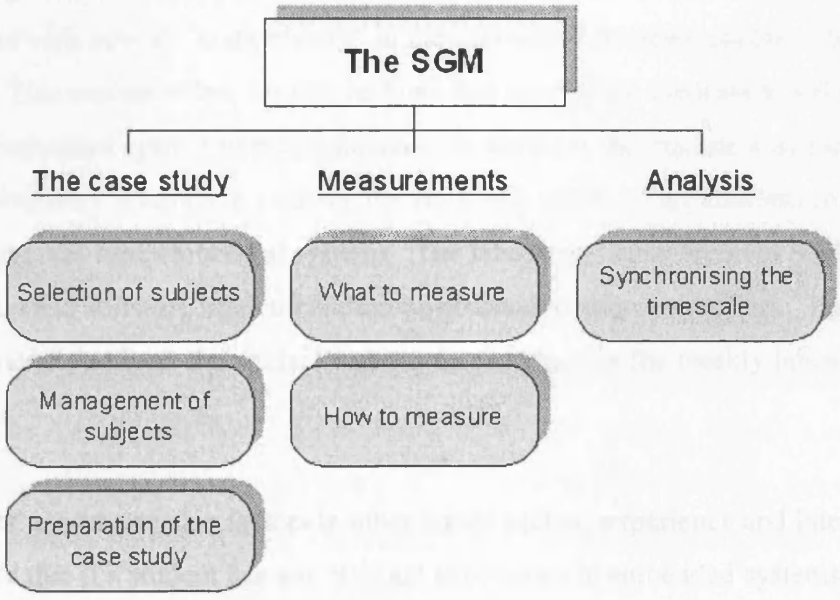

Figure 4-1 Overview of the SGM.

4.4.1 Selection of subjects

A key idea behind the SGM is to use small "balanced" groups of students with equal backgrounds and capabilities as test subjects.

[14] Please note that an early version of the SGM was described by Ayavoo *et al.* (2005a). This work is presented in Appendix A of this thesis.

Most of the small studies described previously tried to match up the backgrounds of the test subjects (Polson *et al.*, 1986; Vessey and Conger, 1994; Robillard *et al.*, 2004). To do this, account was taken of the subjects' prior experience (Robillard *et al.*, 2004) and the courses they had undertaken (Polson *et al.*, 1986). Questionnaires were also used (Vessey and Conger, 1994).

In the SGM, the fact that students have an associated "mark history" was used. This means the knowledge of which university modules the subjects have previously studied and the marks they have obtained for these modules are available. [15] This allows for the possibility of assembling small groups of students who are extremely well matched in terms of background and skill level (in a particular area).

In this case, the research studies that were conducted required the students to implement embedded systems. To minimise the bias in the studies, the students were chosen such that they had very similar "mark history" in the Embedded Systems module offered by the university. This module offers weekly lectures that discuss the theoretical and practical aspects of embedded system implementations. In addition, the module also includes intensive laboratory sessions to evaluate the skills and ability of the students in implementing real-time embedded systems. The laboratory work involves hardware configuration and software implementation on embedded microcontrollers. The "marks history" generally reflects the student's ability in carrying out the weekly laboratory assignments.

The students' marks can also factors in other issues such as experience and interest. Here, it is assumed that if a student has any relevant experience in embedded systems then their marks will be able to reflect this. Similarly, it is unlikely that students who are disinterested in the subject will perform very well.

Please note that, in the studies reported in this thesis, the students were paid expenses (approximately £20.00 / day / student). It is worth noting that even this modest level of expense payment could soon mount up if larger groups were used.

---

[15] Please note that this is – in most cases – "public knowledge": the module marks used are published.

4.4.2 Management of subjects

Selecting suitable test subjects for the study is not enough to ensure its success. In addition, the test subjects must be managed in an appropriate manner to ensure that there is minimal bias in the experiment.

For example, some studies may require the students to work in pairs. This raises the issue of the compatibility within the group. A recent study conducted by Katira *et al.* (2004) looked at issues that could affect the compatibility in student groups such as personality types, actual skill level, perceived technical competence and self-esteem. In that study, the authors have found that in 90% of the cases, student programmers who are randomly paired are compatible. The study has also revealed that the student subjects prefer to pair off with someone they perceive to have similar technical competence. This suggests that pairing student subjects randomly, when the test subjects come from the same background and ability are unlikely to produce incompatible pairs.

Another issue that must be taken into account is the inter-group interaction. While conducting observational studies, it is essential that the student subjects do not discuss their findings with other group members. Sharing information this way could affect the outcome of the results. To avoid this potential bias, the students were advised against this at the beginning of the study. However, having complete control over this is very difficult. For instance, it may be possible to prevent one student from discussing the work with another student in a different group during their lunch break by simply staggering the lunch hours between the groups. However, it is more difficult to curb any form of inter-group discussion at the end of a daily laboratory session. Crucially, this required the cooperation from all the participating students.

It is also important to make sure that the subjects are not aware of the objective of the study as this may compromise the results. This eliminates the possibility of subjects behaving differently because of their knowledge of the experimenter's expectations. (This is also sometimes known as the Hawthorne Effect; see Kitchenham *et al.*, 2002; Berry and Tichy, 2003). To avoid this, the test subjects were only informed of the experiment's objectives and motives at the end of the experiment.

Finally, using the same students in cross-over design must be avoided at all cost. This is the case where the same students are subjected to several different treatments. In this case, the test subject may have gained knowledge from the first treatment and is likely to find the next task slightly easier. As suggested by Kitchenham *et al.* (2002), the second attempt of debugging would be easier than the first, regardless of the technique that is being applied. To avoid this bias, a fresh set of students that had not taken part in any of the prior research studies were selected for each new study.

4.4.3 Preparation of a suitable case study

In order to carry out an empirical investigation effectively, a suitable case study is required. The description of the problem for the case study needs to be carefully designed such that it is feasible for the students taking part in the project. The feasibility of the study must take into consideration the size of the groups, the duration of the study and the experience and ability of the students involved. For example, it is fruitless to prepare a difficult case study that may require ten students to work in a group for six months, when the group size is only two and the students are only available for four days.

The case study must also ensure that the new technology can be tested fairly. To do this, an experiment that does not use any "treatment" can be used as an effective "control". This can then be compared with the results of the experiment that has the "treatment" under test (see Kitchenham *et al.*, 2002; Dyba *et al.*, 2005). That is, the study should be designed such that the problem given to the students can be solved, with or without the new technology.

It is also generally agreed that the case studies used in empirical software engineering need to scale up to real systems and to avoid "toy" problems (Fenton *et al.*, 1994; Basili *et al.*, 1999). For example, Brooks (1980) suggested that software programs that are less than 500 lines of source code can usually be deemed as "toy" programs. Equally, if the problem is made to be too complex, then the results may become more difficult to analyse, and may make the cost of the study itself prohibitive.

In light of this, the programs that the students were required to work on consisted of thousands of lines of source code. Please note that in most cases, some basic structure and

sample code was available so that the subjects did not need to start coding from scratch. However, the subjects had to understand the source code and program structure before they could begin their coding. It is assumed that this approach is similar to most embedded software development process, where the embedded software is rarely created from scratch.

4.4.4 Deciding what to measure

After suitable groups of students and an appropriate case study have been identified, evidence can now be gathered. The evidence can take the form of "tangible" evidence or "testimonial" evidence (Pfleeger, 2005). According to Pfleeger (2005), evidence that can be examined directly to see what it reveals can be classified as being tangible. Testimonial evidence refers to observational reports on how the experiment transpired. To obtain the necessary evidence, a choice of relevant measurements must be made.

One way of doing this is to use the Goal-Question-Metric (GQM) approach (Basili and Weiss, 1984). GQM is a methodology that assists the researcher in determining the suitable software metric that should be measured. To begin with, the goals of the study must be identified, followed by the generation of set of questions to be answered, and proceed step-by-step to identify the type of data to be collected.

For example, with respect to the "simulation first" approach, the fundamental goal is to assist in the development of X-by-Wire systems. One of the essential questions is can simulation predict the behaviour of X-by-Wire systems? This question was answered in Chapter 3 where it was demonstrated how the TrueTime simulator can be used to understand the behaviour of different implementation options. To do this, a measurement of various attributes were carried out, such as control delay, control jitter and response time.

The specific measures used in this research are discussed in Chapters 5, 6, 7 and 8.

4.4.5 Deciding how to measure

Having decided on what to measure, the next step is to decide how to carry out the measurements. In general, in order to collect as much data as possible a continuous

sampling technique would seem to be ideal. However, carrying out the measurements is likely to have an impact on the process under observation (SEL, 1995): that is, the more frequently a measurement is taken, the more likely it is that this will influence the development process itself.

In order to collect as much useful data as possible from the study without causing undue interference, three data collection techniques were employed:

- Progress observation:
  The progress of each team is observed, and records are made on pre-prepared "progress forms" (see Table 4-1). This will allow the observer to note any difficulties faced during the development process. This technique is sometimes used to make qualitative measurements of software development (Seaman, 1999). This method contributes to testimonial evidence from the researcher's point of view.
- Email:
  Each team is asked to e-mail their project source code periodically to the observer. The source code is saved, and subsequently analysed. This method is crucial to SGM because it provides tangible evidence throughout the software development process.
- Questionnaire and interview:
  At the end of the experiment, the students are given a questionnaire (please refer to Appendix I for examples of the questionnaire) to complete, and a short (recorded) interview session is held with each test subject (such techniques have been discussed in Lethbridge *et al.*, 2005). This session is intended to help elicit any additional information that may have been missed out at the other phases of the observation process. This technique offers testimonial evidence from the perspective of the test subjects.

Table 4-1 "Progress form" to categorise the activities undertaken by the students during the experiment. Generally, a tick is marked on the relevant box to indicate that "Activity X" is being carried out at a particular time slot. Notes were also made to record any difficulties and anomalies observed throughout the development process.

Activities\Time	0930	1000	1030	1100	1130	1200	1230	1300	1330	...
Activity 1										
Activity 2										
Activity 3										
...										

4.4.6 Time-scale synchronisation

The software development process can usually be divided into several stages such as requirements analysis, high-level design, low-level design, coding, unit test, integration test, system test and acceptance test (Weller, 1994). In this case, the focus is mainly on the design, coding and testing stages.

The development phases (at the design, coding and testing stages) for the different groups may not necessarily align. For example, when developing an embedded control system, one group may choose to design the controller first before configuring the necessary Input/Output (I/O) ports, while another group may choose to do the opposite, or develop the control and I/O sections in parts. The lack of a common development process makes a direct comparison of the "raw" results more difficult.

Some previous studies have used a classification scheme to formally categorise and analyse the "raw" data obtained from the software empirical study. For example, Kemerer and Slaughter (1999), used a classification scheme to categorise the events that take place at a particular point in time in their longitudinal study of software evolution. In a different study, Germain and Robillard (2005), used a classification scheme to categorise the cognitive activities involved in software development. Source code classifications have also been used to analyse small source code changes (see Purushothaman and Perry, 2005).

The approach here is slightly different from those described previously. The aim here is to allow a comparison of the results by synchronising the data obtained from two development processes (see Figure 4-2). The solution involves dividing the entire

development process into small, identifiable, phases. The number of the phases involved may vary depending on the testbed and case study used. Please note, that by doing so, a development process is being elicited rather than being enforced.

Next, each set of measurement taken at a particular point in time is mapped to the necessary sub-development phases. This mapping is necessary to categorise the phases that were involved for each measurement. Each measurement may have one or more sub-development phases mapped to it. To carry out the mapping process, tools to compare two version of software such as Araxis Merge[16] or Windiff can be used. Finally, measurements can be grouped together in areas where there is more than one measurement in the same sub-development phase.

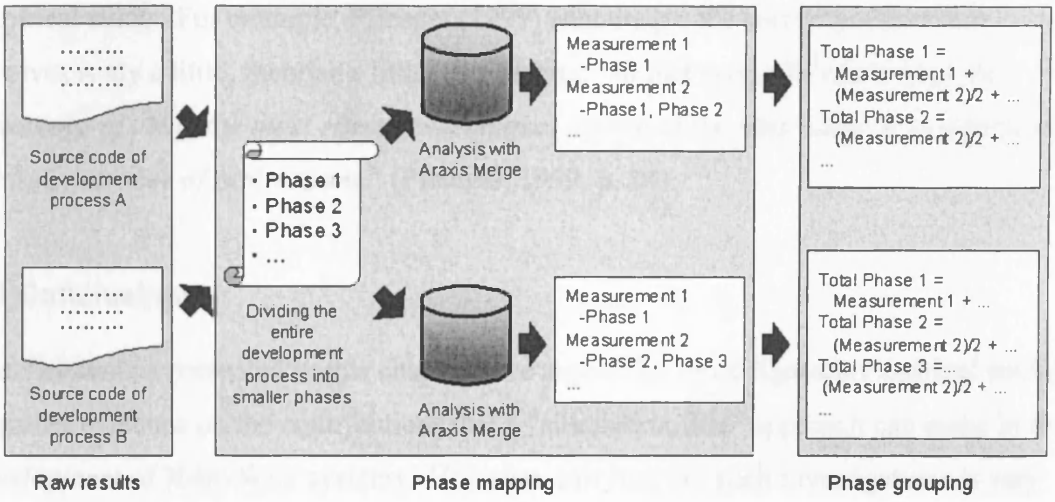

Figure 4-2 The synchronisation technique.

4.5 Discussion

In order to address some of the challenges and difficulties involved in carrying out software empirical research in the field of embedded systems development, the SGM has been proposed. Although the SGM has various advantages, the disadvantages of the approach must also be appreciated.

The main drawback of the SGM is the inability to generalise the findings when such an approach is employed to carry out empirical studies. This is due to the small sample size

[16] http://www.araxis.com/ (last accessed: 27/02/2006)

used in such research projects. Although generalising the findings is a common problem in software empirical research (Basili *et al.*, 1999), it is more severe when the SGM is used.

In addition, due to the small number of students used in the SGM, there is a risk that the human variation in characteristics may make it difficult to interpret the results of the study, and obscures the conclusion. For example, Sheil (1981) has discussed the individual variability among test subjects and its effect on the results.

However, due to the various difficulties that have been discussed in Section 4.3, the SGM provides a practical technique for gathering empirical evidence. In this case, it may be better to rapidly obtain some evidence than to wait indefinitely to conduct an "ideal" empirical study. For example, Pfleeger (1999) encourages the sort of practice that involves study a little, theorise a little, then iterate: "*In this way, educators have the advantage of using the most effective techniques known at the time, without having to wait for large number of replications.*" (Pfleeger, 1999, p. 34).

4.6 Conclusions

The discussions presented in this chapter have argued for more rigorous empirical studies to gather evidence on the contributions that a "simulation first" approach can make in the development of X-by-Wire systems. However, carrying out such investigations is very difficult. Based on the assumption that it is better to gather "some" evidence instead of no evidence at all, the SGM has been proposed. Overall, the approach is intended as a way of rapidly carrying out empirical assessments at low cost, and to help suggest new hypotheses.

Having proposed a methodology to carry out empirical evaluations, the subsequent chapters explore how the SGM can be employed to answer more specific questions involving the comparison of different development approaches for embedded systems.

5. Can the use of simulation reduce effort?

The work presented in this thesis focuses on evaluating the effectiveness of a "simulation first" approach to the development of X-by-Wire systems. In this chapter, the "Small Group Methodology" (introduced in Chapter 4) was used to assess if a "simulation first" approach can reduce the effort involved.[17]

5.1 Introduction

The initial goal of this project was to build a suitable simulation tool for X-by-Wire systems. To avoid reinventing the wheel, the TrueTime simulator was provisionally chosen (see Chapter 2), and it has been shown to be capable of predicting the behaviour of a range of X-by-Wire systems (see Chapter 3). However, as discussed in Chapter 4, this condition alone is not enough to demonstrate that a "simulation first" approach is effective.

One factor that marks out the success of a "simulation first" approach, or any given software engineering project for that matter, is the development effort involved (Jørgensen and Sjøberg, 2001; Moløkken-Østvold and Jørgensen, 2003; Grimstad *et al.*, 2006; Huang and Chiu, 2006). Indeed, some researchers have suggested that a "simulation first" approach is likely to reduce the development effort involved (see Castelpietra *et al.*, 2002; Henzinger *et al.*, 2003; Redell *et al.*, 2004). For example, Redell *et al.* (2004), p. 181 concluded "*The toolset hence allows users to evaluate control systems implementations before realisation, which should help lowering development times ...*".

However, there is very little empirical evidence that can support such claims. For instance, although Henzinger *et al.* (2003) claimed that the use of the Giotto tool helped to significantly reduce the effort involved, there were no empirical results to demonstrate this, or any discussion on the techniques used to obtain such measurements. Overall – in most cases – the developers of simulation tools for embedded systems have sought to demonstrate the technical merits of the tool, but have not attempted to study the efficacy of the approach when used in practice.

[17] Parts of this chapter also appears in Ayavoo *et al.* (submitted).

The aim of the work presented in this chapter was therefore to begin to seek evidence of the effort involved when employing a "simulation first" development approach. The use of simulation in the studies presented here was restricted to the verification of the correct design implementation of a preferred solution. To do this, the SGM was employed where the development of two versions of an embedded control system by independent groups of students was observed in Case Study 5. One group was asked to implement the system directly while the second group was asked to use a simulator first, before carrying out the hardware implementation.

5.2 Case Study 5: Assessing the effort involved

As outlined in Section 5.1, Case Study 5 involved the observation of the development of a non-trivial embedded control system. The SGM was applied in the present study to assess the contribution of a "simulation first" approach towards the development process. The study is described in this section.

5.2.1 The testbed

The testbed employed in this study was based on the cruise-control system (CCS) hardware-in-the-loop (HIL) testbed described in Chapter 3. The CCS testbed was chosen for the following reasons:

- The testbed was based on an embedded automotive application.
- The testbed presented a realistic implementation problem of a control system.
- The CCS can be developed with and without the use of a software simulator.
- The CCS testbed was successfully used (in Case Study 3A) to evaluate the performance of the simulator.

In this study, the testbed was modified such that the CCS node consists of a single Infineon 16-bit microcontroller. This was done in order to remove the associated complexity of distributed embedded systems, and hence to make the study feasible for the selected test subjects within the allocated time and cost constraints (Section 5.2.2 and 5.2.3 discusses the selection of subjects and the time allocated for the study).

5.2.2 Selection of subjects

Having selected the CCS as the testbed for this study, the next step was to select suitable test subjects.

In the study presented here, students that had previously undertaken a one-semester module in embedded systems were used. From the list of volunteers, the student subjects were specifically chosen such that their marks for this module were very similar (in the range 70%-80%) to ensure that the all the groups were – as far as possible – well matched in background and ability.

Four groups of students were employed to develop the CCS. The breakdown of the groups (and the allocated tasks) is shown in Table 5-1.

Table 5-1 Group structure for Case Study 5.[18] Table adapted from Ayavoo *et al.* (submitted), Table II.

Group	Number of members	Scheduling approach	Software simulation	Hardware implementation
A	1	Pre-emptive		√
B	2	Co-operative		√
C	1	Pre-emptive	√	√
D	2	Co-operative	√	√

One way of classifying embedded software is through the labels "pre-emptive" and "co-operative" (see Section 2.2.1). Previous studies have suggested that co-operative designs have more predictable behaviour, and may be easier to simulate (Nissanke, 1997). In order to reduce the impact of the choice of system architecture on the results obtained, the CCS was developed using two scheduling approaches, either a time-triggered co-operative software architecture (Pont, 2001) or a time-triggered pre-emptive software architecture (Bate, 1998). For each architecture, one group was instructed to perform a software simulation first followed by the hardware implementation while the other group was asked to perform the hardware implementation immediately.

[18] As shown in Table 5-1, some groups had two students while some had one. This discrepancy was caused by some volunteers who pulled out of the study at the last minute. However, this did not have a huge bearing on the study since the comparisons were carried out in a non-bias way. For instance, comparison between Group A and Group B was avoided because the number of subjects in those groups was not the same.

Every effort was made to ensure that the groups were supported equally. All groups were provided with a real-time task scheduler (co-operative or pre-emptive) suitable for use in the implementation phase. For groups that were involved in the software simulation, an example of the scheduler running on the simulator and the car environment model was also provided. All groups were also provided with a documentation of the necessary formulae and variables required to develop the CCS.

5.2.3 Measurement of effort

In the present study, the work involved in the development of both the "simulated" and "real" systems, had to be measured and compared.

The work involved in the simulation phase included creating block diagrams (with the appropriate connections as shown in Figure 5-1) for the system and subsequently writing MATLAB source code for the TrueTime simulator. An example of a task in MATLAB is shown in Listing 5-1.

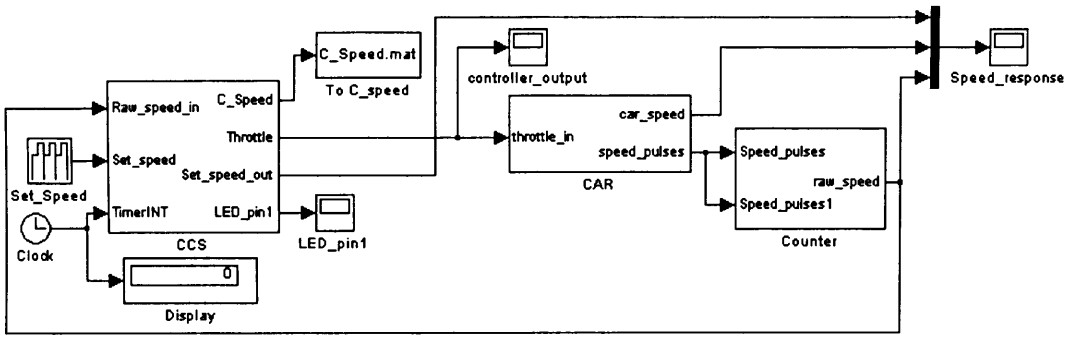

Figure 5-1 The control system setup on TrueTime.

```
function [exectime, data] = CCS_Sens_Compute_Speed(seg, data)
switch seg,
   case 1,
      data = ttTryFetch('Data_Box');
      exectime = 0.000001;
   case 2,
      % Get raw speed from car
      % The raw speed is scaled to obtain the
      % calculated speed of the car
      data.raw_speed = ttAnalogIn(data.inpChan(1));
      data.scaled_speed = data.scaling_factor*data.old_speed
      + (1-data.scaling_factor)*(data.raw_speed * 4);
      data.old_speed = data.scaled_speed;
      ttAnalogOut(data.outChan(1), data.scaled_speed);
      ttTryPost('Data_Box', data);
      exectime = 0.000046;
   case 3,
      exectime = -1;
end
```

Listing 5-1 An example of a task specification created using the TrueTime simulator.

All of the work involved in the implementation phase required coding in C (the source code was developed and compiled on the Keil IDE). For example, the task specified in TrueTime in Listing 5-1 is shown in C in Listing 5-2.

```
void Sens_Compute_Speed(void)
   {
   tWord raw_speed;

   raw_speed = Get_Raw_Speed();
   Scaled_speed_G = ((float)(SCALING_FACTOR * Old_speed_G) +
   (float)((1 - SCALING_FACTOR) * (raw_speed * 4)));
   Old_speed_G = Scaled_speed_G;
   }
```

Listing 5-2 The task specified in TrueTime in Listing 5-1 is shown in C here.

Since both the simulation and implementation phases required coding, code-based metrics were used as the basis of the comparisons (McCabe, 1976; Stark *et al.*, 1994; Weller, 1994)[19]. In addition, for the simulation phase, the time required to build the block diagrams was included in the measures of effort required to develop the source code since the source code and diagrams are interdependent: in this case development time was felt to be an appropriate metric as it has previously been used to measure effort (Solingen and Stalenhoef, 1997; Basili *et al.*, 2004).

[19] Please note that although the languages of the simulation and implementation were different, the structure and logic of the tasks were similar.

Having decided what to measure, it was possible to then decide when to measure. In the present study, the students were allocated (up to) four days to implement the CCS. Taking daily measurements would not be appropriate as the sample points would be too coarse. Conversely, taking the measurements (say) every five minutes would be too intrusive. In this study, it was decided that the effort was measured approximately every 30 minutes by asking each team to e-mail their project source code. This duration was felt to be appropriate as it could capture the progress involved in sufficient detail without compromising the study. The source code submitted in this way was saved, and subsequently analysed.

5.3 Synchronising the timescale

After carrying out the relevant measurements, the raw data were analysed. As noted in Section 5.2.3, a record of all the software versions was recorded, along with information about the time taken to create each version. Since each group produced different code versions at different rates, it made comparison of the results difficult. To make the comparison of the results possible, synchronisation of the development phases was carried out, as discussed in Chapter 4 (see Section 4.4.6).

To do this, the total software development of the CCS was divided into smaller phases. In this case, the key software development phases were identified to be:

- Gaining familiarity with the development environment (FAMENV).
- Writing the "Compute Car Speed" task (COMSPD).
- Writing the "Get Ref Speed" task (GETREF).
- Writing the "Compute Throttle" task (COMTHR).
- Debugging the tasks to make the system work (DBGTSK).

The source code was then compared and analysed (using Araxis Merge for Windows). Each version was then linked to one (or more) of the system development phases listed.

Table 5-2 shows the phases that were developed for each version.

Table 5-2 Results for Case Study 5 after each version was mapped to its corresponding phase and its respective time taken in minutes.

Group A		Group B		Group C		Group D	
COMSPD	30	COMSPD	30	FAMENV	30	FAMENV	40
COMSPD	35	COMSPD	35	FAMENV	40	FAMENV	40
COMSPD	25	COMSPD	25	FAMENV/COMTHR/COMSPD	30	FAMENV	30
GETREF	45	COMSPD/COMTHR	45	COMTHR	30	FAMENV	30
GETREF/COMTHR	30	GETREF	30	COMTHR/COMSPD/GETREF	30	FAMENV	45
COMTHR	30	GETREF	35	DBGTSK	30	FAMENV	10
DBGTSK	35	DBGTSK	35	DBGTSK	30	COMSPD/COMTHR	40
DBGTSK	35	DBGTSK	25	DBGTSK	40	COMTHR	20
DBGTSK	25	DBGTSK	25	DBGTSK	20	GETREF/DBGTSK	30
DBGTSK	25	DBGTSK	20	DBGTSK	20	DBGTSK	30
DBGTSK	35	DBGTSK	25	COMTHR	40	COMSPD	30
DBGTSK	15	DBGTSK	25	COMTHR/COMSPD/GETREF	30	COMSPD	30
DBGTSK	30			COMTHR/COMSPD/GETREF	30	COMSPD	30
DBGTSK	20			COMSPD/DBGTSK	30	GETREF	40
DBGTSK	40			DBGTSK	40	COMTHR	30
				DBGTSK	30	DBGTSK	30
				DBGTSK	30		
				DBGTSK	30		
				DBGTSK	30		
				DBGTSK	30		
	455		355		620		505

From Table 5-2, it can be observed that the effort expended on the various phases varied between groups. To allow a more detailed analysis of the effort involved in each phase, 'similar' phases were grouped together. Where necessary, the total development time was divided by the number of phases.

Table 5-3 illustrates the results after similar versions were grouped together.

Table 5-3 Results (in minutes) for Case Study 5 after grouping versions together. Table adapted from Ayavoo *et al.* (submitted), Table III.

	Phases	Group A	Group B	Group C	Group D
SW SIM	FAMENV	0	0	80	195
	COMSPD	0	0	20	20
	GETREF	0	0	10	15
	COMTHR	0	0	50	40
	DBGTSK	0	0	140	45
HW IMP	FAMENV	0	0	0	0
	COMSPD	90	112	35	90
	GETREF	60	65	20	40
	COMTHR	45	23	60	30
	DBGTSK	260	155	205	30
	TOTAL	455	355	620	505

5.4 Analysis of the results for Case Study 5

Once the results have been synchronised, it is now possible to analyse the results.

Five factors were considered:

- Total development effort.
  This is the time taken to complete the project by each group. For Groups A and B, this only involves the hardware implementation. For Groups C & D, this involves the software simulation and the hardware implementation.
- Total hardware implementation effort.
  This is the effort involved at the implementation stage only. For Groups A and B, this value should be similar to the total development effort. For Groups C and D, the simulation effort is ignored.
- Hardware implementation effort in stages.
  This value measures the hardware implementation time based on the various software development phases. The phases are COMSPD, GETREF, COMTHR and DBGTSK.
- Comparison of the total simulation and total implementation effort.
  This is the comparison of the total effort spent at the simulation stage and the total effort involved in the implementation for Groups C and D only.
- Comparison of simulation and hardware implementation effort in stages.
  Here, a comparison was done between Groups C and D of the simulation and implementation effort involved at the various development phases (COMSPD, GETREF, COMTHR and DBGTSK).

5.4.1 Mean results

The mean results obtained from the experiment are summarised in Table 5-4.

Table 5-4 The mean results for effort (in minutes) for Case Study 5. Table adapted from Ayavoo *et al.* (submitted), Table IV.

	Mean of HW Groups (A & B)	Mean of SIM Groups (C & D)	Effort of SIM groups compared to HW groups
Total hardware implementation effort	405	255	37% less
Total development effort	405	563	39% more

If the mean implementation effort for the groups is considered, it can be seen that although the effort involved to implement the hardware was 37% less than the "hardware-only" groups, the overall development effort was 39% more for the groups using the TrueTime simulator.

Inevitably, given the small group size, it would be expected that there may be differences in the individual student behaviour. Nonetheless, it is useful to examine the performance of individual students, to see if any further lessons can be learned from this study. The results of this analysis are reported in the following sections.

5.4.2 Total development effort

From Table 5-3 it can be seen that the total development time for Group C and Group D was much higher than for Group A and Group B.

In addition, it should be noted that the student subjects were not familiar with TrueTime simulation but had previous experience implementing embedded systems (this is confirmed by the null values for FAMENV for all the groups for hardware implementation).

If the effort taken in the FAMENV stage is ignored for Group C and Group D (that is, the time taken to gain familiarity with the simulator is discounted), then the overall effort for Group C was 19% more than Group A while Group D took 13% less effort than Group B.

These (apparently contradictory) results will be considered again in Section 5.5.

5.4.3 Total hardware implementation effort

If only the time taken to perform the hardware implementation was considered (Figure 5-2), then it can be seen that the groups that embarked on the "simulation first" approach (Group C and Group D) took a much shorter time to implement the system compared to the "implementation only" groups.

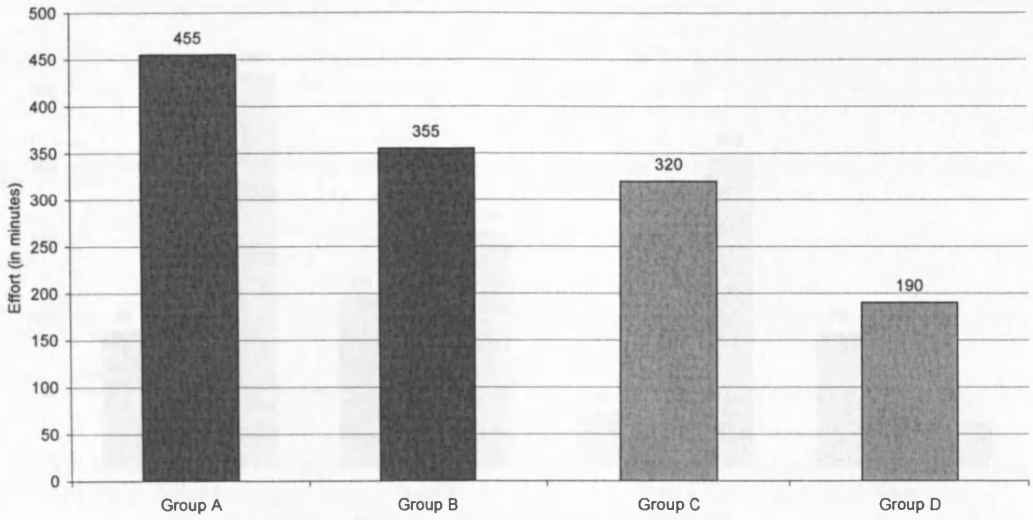

Figure 5-2 Total hardware implementation time for all the groups in Case Study 5.

For the groups that used the pre-emptive scheduler, the effort reduction (Group C) was 30%. For the groups that used the co-operative scheduler, the time reduction (Group D) was 46%.

This suggests that simulation had reduced the effort required to subsequently implement the system on the hardware.

5.4.4 Hardware implementation effort in stages

By plotting the progress made in the hardware implementation in stages (Figure 5-3), it can be seen that Group A, Group B and Group C had similar characteristics. In each of these cases, the development effort increased towards the end of the hardware implementation phase.

By contrast, in the case of Group D, the development effort seemed to have declined towards the end of the implementation phase.

This result will be considered again in Section 5.5.

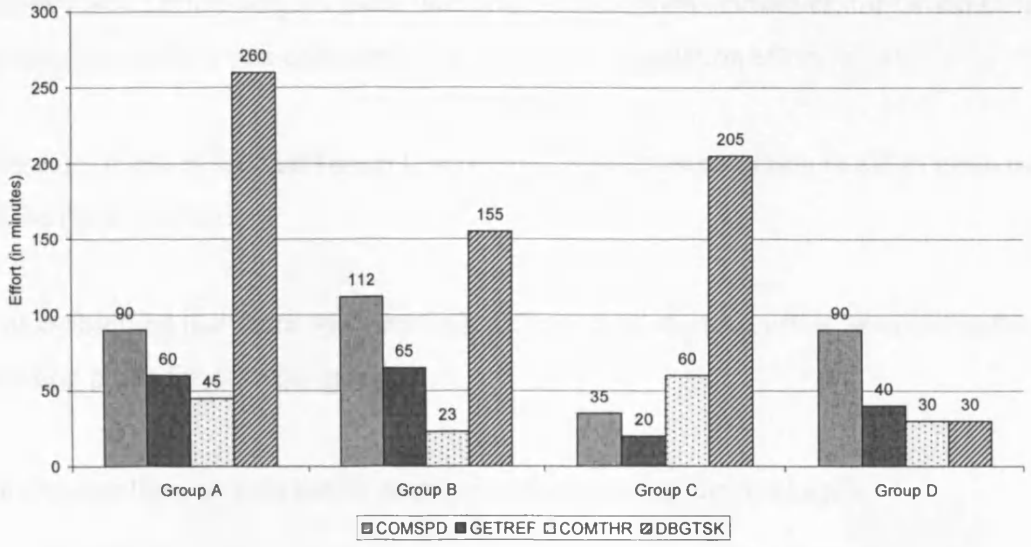

Figure 5-3 Hardware implementation effort in stages for the four groups in Case Study 5.

5.4.5 Comparison of simulation and implementation effort

By comparing the total development effort of the simulation and implementation process for Group C and Group D, two different results can be observed (Figure 5-4).

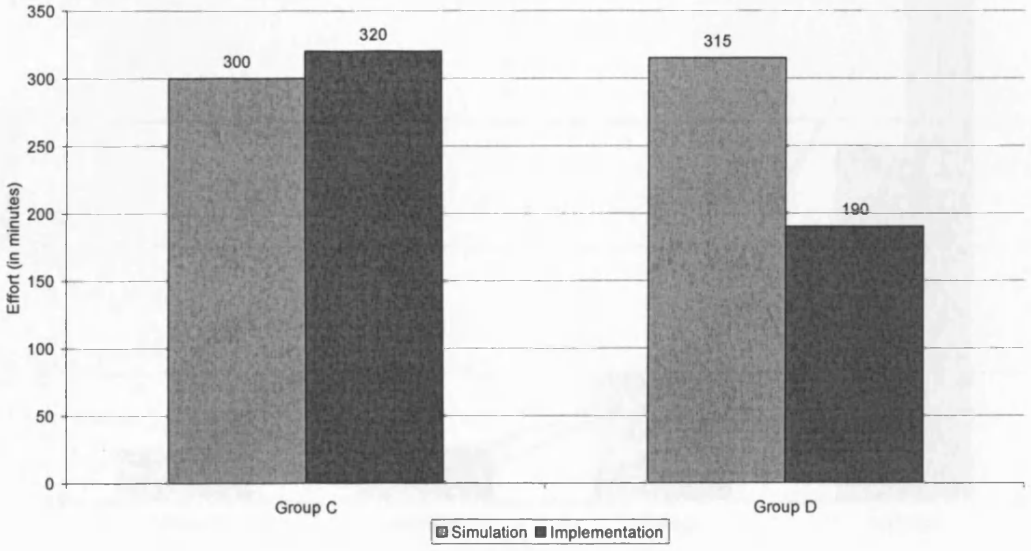

Figure 5-4 Comparison of total simulation effort and total implementation effort for Group C and Group D in Case Study 5.

The duration of the simulation and implementation phases for Group C was very similar (implementation effort was 7% more than simulation effort). However, for Group D, the implementation effort was considerably less than the simulation effort (by 40%).

In this case, it was noted that Group D showed a significant reduction in effort when using the TrueTime simulator.

It is also observed that there was very little difference in the total effort involved in the simulation phase for both the groups.

5.4.6 Comparison of simulation and implementation effort in stages

Next the results from each stage of the development process (COMSPD, GETREF, COMTHR, DBGTSK) were considered.

The results for Group C (Figure 5-5) reveal similar simulation and implementation effort characteristics.

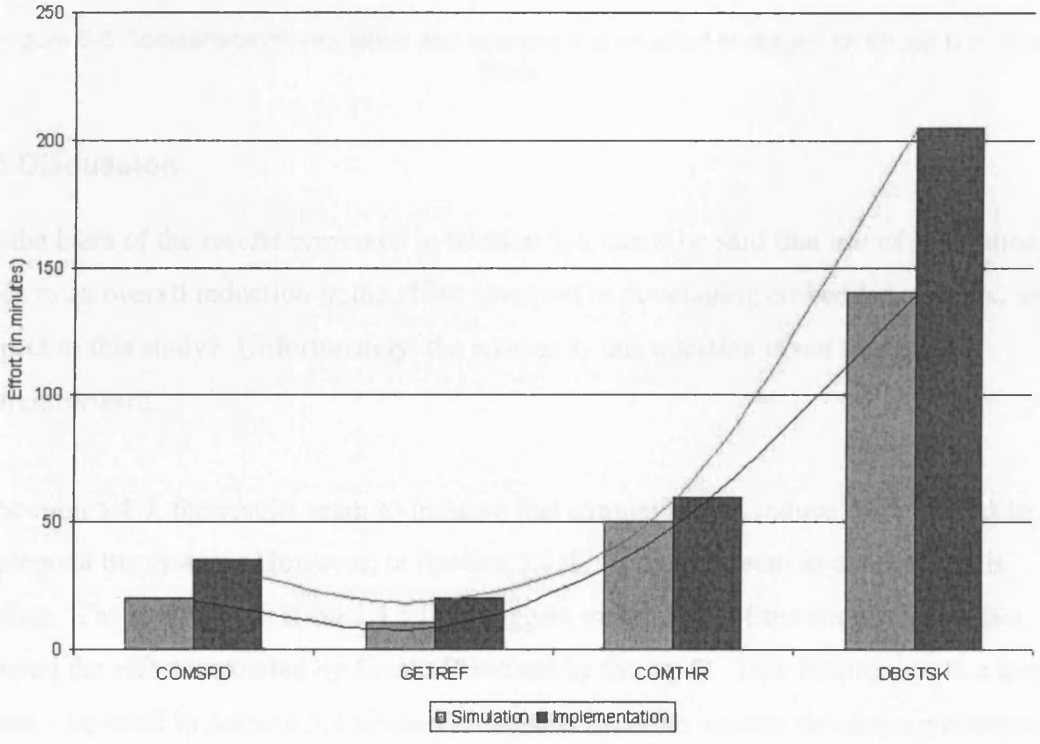

Figure 5-5 Comparison of simulation and implementation effort in stages for Group C in Case Study 5.

However, the results for Group D (Figure 5-6) show different effort characteristics for the two phases.

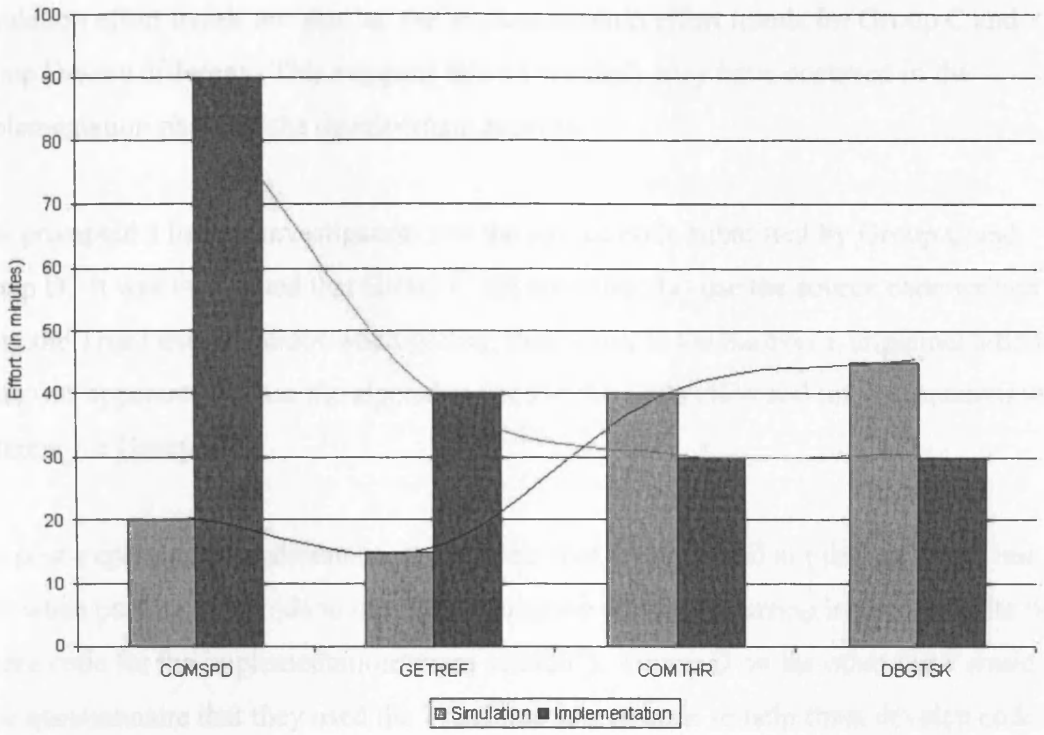

Figure 5-6 Comparison of simulation and implementation effort in stages for Group D in Case Study 5.

5.5 Discussion

On the basis of the results presented in Section 5.4, can it be said that use of simulation leads to an overall reduction in the effort involved in developing embedded systems, with respect to this study? Unfortunately, the answer to this question is not totally straightforward.

In Section 5.4.3, the results seem to indicate that simulation did reduce the effort taken to implement the system. However, in Section 5.4.4, the results seem to contradict this finding. The results in Section 5.4.5 then suggest that the use of the simulation in fact reduced the effort expended by Group D but not by Group C. This finding is – to a large extent - repeated in Section 5.4.6 when the results from the various development stages are compared.

Overall, although both Group C and Group D had used the same simulator, the impact on the working patterns of the groups was different. The results suggest that although the simulation effort trends are similar, the implementation effort trends for Group C and Group D were different. This suggests that an anomaly may have occurred in the implementation phase of the development process.

This prompted a further investigation into the source code submitted by Group C and Group D. It was then noted that Group C did not (directly) use the source code written using the TrueTime simulator when porting their work to the hardware implementation. (This was apparent because the algorithm used in the simulation and implementation was different for Group C).

In a post-experiment questionnaire, it was clear that Group C did not use the TrueTime files when porting their code to the implementation phase (preferring instead to write the source code for the implementation "from scratch"). Group D on the other hand stated in their questionnaire that they used the TrueTime source code to help them develop code for the hardware implementation.

Overall, looking back at the individual results, the following lessons can be learned:

- The actual progress and effort needed in each phase depends on the approach the groups take. For example, Group D generally took more time to plan and understand the problem, and reaped the benefits in later stages. Group C, on the other hand, spent less time in planning and therefore had to debug for a longer time. Group C also made less effective use of the simulator, changing their algorithm at the hardware implementation stage.
- Learning how to use the simulator inevitably contributes to the project overhead. Disregarding this overhead (that is, assuming that the developers are already familiar with the simulator) a "simulation first" approach does appear to reduce the overall effort.
- Using a "simulation first" approach can lead to a reduction in the implementation time (at the hardware implementation stage). This could be due to the benefits of experimenting in the simulator environment, and may also be because the simulator promotes familiarity with the problem before beginning the hardware implementation.

- Developing the simulation models contributed to a significant amount of effort. In average, 55% of the overall development effort came from developing the relevant simulation models.
- The group that used the simulation source code for the implementation phase (Group D) demonstrated a significant reduction in overall effort. By contrast, the group that made less effective use of the simulator (Group C) actually saw their total effort increase.

5.6 Conclusions

The work presented in this chapter involves an empirical study conducted using the SGM to evaluate the effort involved when using a "simulation first" approach to develop an embedded control system. Overall, Case Study 5 suggests that the group that used the simulation source code for the implementation phase (Group D) demonstrated a reduction in overall effort. By contrast, the group that made less effective use of the simulator (Group C) actually saw their total effort increase. Specifically, the results suggest that the use of a "simulation first" approach can contribute to a significant reduction of the effort involved in the implementation phase.

The study also revealed that the process of developing the relevant TrueTime simulation models can contribute to a substantial amount of effort. The next chapter will explore ways in which the overall effort can be further reduced. In particular, the investigation will explore the possibility of reducing the effort involved in the simulation phase of the development process.

More generally, the results demonstrate that use of small groups of individuals can provide useful information about the development of complex embedded systems, if studies are carried out in an appropriate manner.

6. Can the development effort be further reduced?

In the previous chapter, it was shown that although simulation can reduce the implementation effort, the effort involved to create the simulation models was not trivial. In this chapter, another SGM-based investigation is carried out to assess if the effort involved can be further reduced. To do this, the TrueTime simulator was modified. [20]

6.1 Introduction

As discussed in Section 1.2.3, the research project described in this thesis began with the aim of developing a simulator to evaluate X-by-Wire designs very rapidly. The subsequent work (presented in Chapter 2 to Chapter 4) suggests that – instead of reinventing the wheel – the TrueTime tool can be used as it closely matches the initial requirements. Following this (in Chapter 5), further empirical investigations suggested that TrueTime – when used effectively – can reduce the effort involved. In particular, evidence was obtained which suggests that the "simulation first" approach that employs TrueTime can lead to reduction in the implementation effort.

However, the evidence presented in Chapter 5 also suggests that a considerable amount of effort was involved in the development and simulation of the TrueTime models. In particular, it was found that a substantial amount of effort was spent configuring the TrueTime block diagrams and writing the relevant MATLAB source code. It was also noted that learning to use TrueTime requires a significant amount of effort.

Other researchers have also suggested that developing the relevant simulation models can sometimes be rather tedious (Castelpietra *et al.*, 2002; Carson II, 2005). For instance, Castelpietra *et al.* (2002), p. 1263 concluded: "*However, simulation model development is quite time consuming, as it has to obey to the formalism of a general-purpose simulation tool that is not specific to an embedded system domain.*".

These observations suggest that, although the code-based approach of the TrueTime simulator provides flexibility in developing the system, it (inevitably) increases the effort required to develop the relevant simulation models.

---

[20] Parts of this chapter have previously been published in Ayavoo *et al.* (2006) and Ayavoo *et al.* (submitted).

Based on this discussion, the work presented in this chapter studies the impact of modifications to the TrueTime simulator on the development effort. The modified version of TrueTime (referred to here as TrueTime-Plus) is described in Section 6.2. An empirical evaluation using the SGM (Case Study 6) is presented in Section 6.3 to assess the efficacy of the TrueTime-Plus simulator.[21]

6.2 TrueTime-Plus

Intuitively, it seems likely that the overall effort could be further reduced if the effort involved in creating the simulation models is reduced. With the aim of reducing this effort, a code-generation technique was employed.

Code-generation techniques have also been used in other simulation tools. For example, Real-Time Workshop Embedded Coder from MathWorks generates C code from Simulink models. Similarly, tools such as TimesTool and Giotto (discussed in Chapter 2) also employ code-generation techniques for the final implementation (Amnell *et al.*, 2003; Henzinger *et al.*, 2003). Please note that code generation is – obviously – not limited to simulators, and has been used in other research areas (see Florijn *et al.*, 1997; Mwelwa *et al.*, 2006).

In this study, code generation was used to automate the development of the relevant simulation models (instead of the final implementation as carried out in previous studies). In particular, an "add-on" design-led code-generation package called TrueTime-Plus (TT-Plus) was developed using Visual C to complement the existing TrueTime simulator. TT-Plus prompts the developer with a series of possible high-level design options for the system implementation, and then generates the necessary MATLAB files that are required for the simulation process on TrueTime. This process is expected to assist the user in rapidly developing a working prototype of their system without having to immediately deal with the low-level complexities of choosing the right TrueTime library functions.

[21] Please note that by modifying the simulator, the initial aim of this thesis – which was to build a suitable simulator – was partially fulfilled.

The source files generated can then be edited to accommodate the task specific instructions. Figure 6-1 illustrates the differences in the simulation process of TrueTime and TT-Plus.

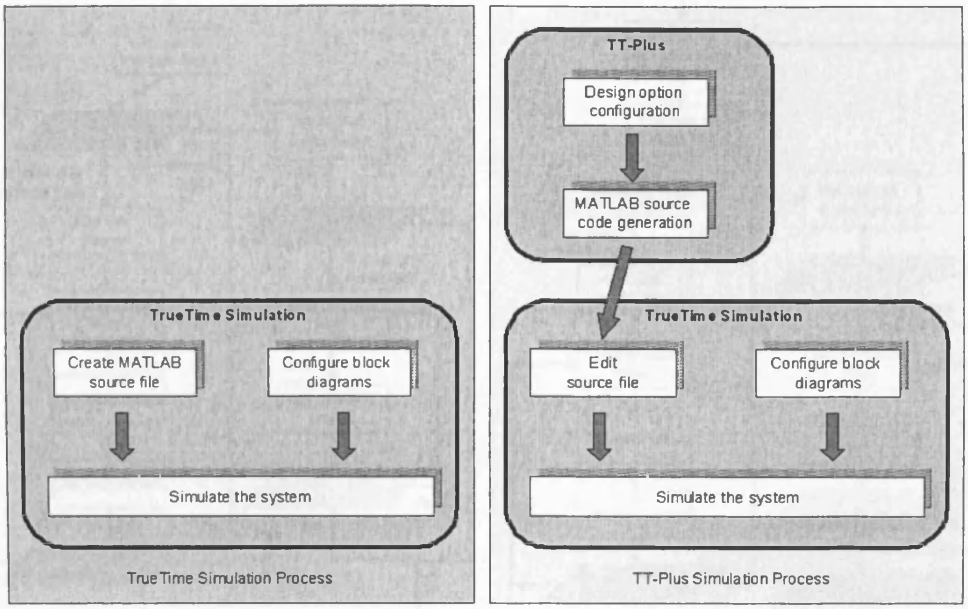

Figure 6-1 Comparison of TrueTime and TT-Plus simulation process.

Currently, TT-Plus takes the designer through the following design options: (1) number of nodes in the system, (2) number of inputs/outputs on a node, (3) processor scheduling strategy (time-triggered or event-triggered), (4) communication scheduling strategy (time-triggered or event-triggered), (5) number of tasks (periodic and aperiodic), (6) task execution times[22], (7) task delays, (8) task periods and (9) system tick interval. Figure 6-2 illustrates the process of leading the user through the design implementation options in TT-Plus.

[22] It is accepted that lack of knowledge about worst-case task execution time is a problem. However, it is one which faces the developers of many embedded systems. For example, as Gergeleit and Nett have noted: "Nearly all known real-time scheduling approaches rely on the knowledge of worst-case execution times for all tasks of the system" (Gergeleit and Nett, 2002). In this thesis – as in previous studies – it is assumed that the developer will have access to worst-case execution time information.

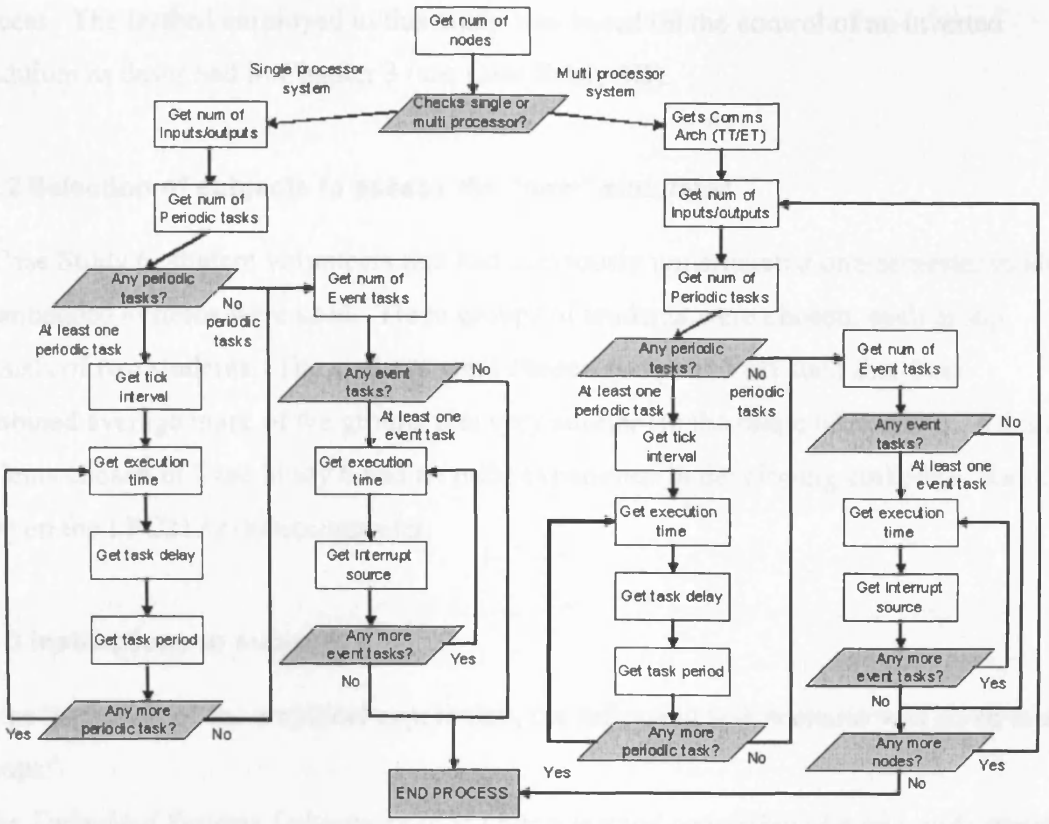

Figure 6-2 A flowchart describing how TT-Plus leads the user through the various implementation options.

Please note that TT-Plus was developed as a simple prototype, in order to investigate the impact of simulation on development effort. Further work would –clearly – be required to develop TT-Plus into a commercial-quality product. Such work was beyond the scope of the research project described in this thesis.

6.3 Case Study 6: Evaluation of the "new" simulation methodology

Having developed the TT-Plus package, the "new" simulation methodology can now be assessed. To evaluate if TT-Plus can indeed reduce the development effort, observation of the development process was carried out using the SGM. The evaluation method is described in this section.

6.3.1 Testbed to assess the "new" simulator

To assess the contribution of this "new" simulation methodology, it was decided that a different testbed is used instead of the CCS to obtain more variability in the evaluation

process. The testbed employed in this study was based on the control of an inverted pendulum as described in Chapter 3 (see Case Study 3B).

6.3.2 Selection of subjects to assess the "new" simulator

In Case Study 6, student volunteers that had previously undertaken a one-semester module in embedded systems were used. Three groups of students were chosen; each group consists of two students. The students were chosen and paired off such that their combined average mark of the groups was very similar (in the range 65%-69%). All the students chosen in Case Study 6 had no prior experience in developing embedded source code on the LPC2129 microcontroller.

6.3.3 Instructions to subjects

At the beginning of the empirical experiment, the following task scenario was given to all groups:
"*The Embedded Systems Laboratory (ESL) has a testbed consisting of a one-node inverted pendulum controller, which is based on a time-triggered software architecture. The ESL is now interested in converting the one-node pendulum controller to a distributed (multiprocessor) control system that involves either two or three microcontrollers connected over a Controller Area Network (CAN) bus. The distributed solution should use the Shared-Clock CAN (SCC) scheduling approach in its final implementation*".

The group's task was to determine – by means of some empirical results – which of the two implementations (two nodes or three nodes) would be a "better" option.

Each group was asked to come up with the solution using different development approaches, as illustrated in Table 6-1. Group A was asked to obtain the solutions by directly implementing the different systems on the hardware and measuring the performance. Group B was instructed to use the TrueTime simulator to decide which design option would be "better" before implementing their preferred solution on hardware. The development approach used by Group C was similar to Group B, the only difference being that Group C was given the additional TT-Plus package to guide them through the system design decisions and code generation during the simulation phase.

Table 6-1 Development methodology for each group in Case Study 6. Table adapted from Ayavoo *et al.* (2006), Table 1.

	Group	TT-Plus	TrueTime simulation	Hardware implementation
Case Study 6	A			√
	B		√	√
	C	√	√	√

Every effort was made to ensure that the groups were supported equally. Each group was provided with the same documentations. Each group was also provided with the working solution of a single-node pendulum implementation and the basic implementation of the SCC algorithm (for two and three nodes) for the LPC2129 microcontroller. For the groups using the "simulation first" approach, a simulation of the one-node pendulum control was provided (plant and controller). In addition, the necessary simulation block diagrams for a typical two-node and three-node control system connected over CAN were provided. However, the source code required to simulate these systems were not provided.

This also meant that all the groups had a working single processor control system to begin with, before attempting to make the necessary design changes for a multiprocessor system.

6.3.4 Measurement of effort

To measure – and compare – the effort involved in the development of both the "simulated" and "real" systems, the time taken by each group to complete the project was measured. In addition, the source code for each group was collected periodically. In this study, 30 minute intervals were felt to be adequate for a five-day study. A progress observation form was also used to record the development of each group.

6.4 Synchronising the timescale

As done previously in Chapter 5, the timescale of the development process was synchronised in order to analyse the results. To do this, the development of the pendulum control system was divided into different phases. In this study, the following key development phases were identified:

- Familiarising with the development environment (FEMENV).

- Using TTPLUS in the simulation process for a two-node system (2N_TTPLUS).
- Using TTPLUS in the simulation process for a three-node system (3N_TTPLUS).
- Communicating messages (transmitting and receiving) over CAN for two nodes using the simulator (2N_TXRX_SIM).
- Communicating messages (transmitting and receiving) over CAN for three nodes using the simulator (3N_TXRX_SIM).
- Porting the necessary tasks over to simulate the control of the pendulum for two nodes (2N_PEND_SIM).
- Porting the necessary tasks over to simulate the control of the pendulum for three nodes (3N_PEND_SIM).
- Measuring the control delay for a two-node system using the simulator (2N_MEA_SIM).
- Measuring the control delay for a three-node system using the simulator (3N_MEA_SIM).
- Implementing and testing the two-node pendulum control on hardware (2N_PEND_HW).
- Implementing and testing the three-node pendulum control on hardware (3N_PEND_HW).
- Measuring the control delay for a two-node system using the hardware (2N_MEA_HW).
- Measuring the control delay for a three-node system using the hardware (3N_MEA_HW).

Table 6-2 illustrates the synchronised results.

Table 6-2 Synchronised results for Case Study 6 after grouping similar phases together and its respective time taken in minutes. Table adapted from Ayavoo *et al.* (2006), Table 3.

	Phases	Group A	Group B	Group C
SW SIM	FEMENV	0	195	75
	2N_TTPLUS	0	0	90
	3N_TTPLUS	0	0	15
	2N_TXRX_SIM	0	765	90
	3N_TXRX_SIM	0	195	60
	2N_PEND_SIM	0	120	120
	3N_PEND_SIM	0	45	30
	2N_MEA_SIM	0	150	60
	3N_MEA_SIM	0	90	60
HW IMP	2N_PEND_HW	540	270	240
	3N_PEND_HW	255	0	0
	2N_MEA_HW	255	120	60
	3N_MEA_HW	300	0	0

6.5 Analysis of the results for Case Study 6

Once the results have been synchronised, it is possible to analyse the results for the following.

- Total development effort.

  This is the total effort spent from the beginning till the end of the project. For Group A, this only involves the hardware implementation effort. For Groups B and C, this includes the simulation and implementation effort.

- Effort taken to decide on the "best" system.

  This measure of the effort indicates the time taken by the groups to decide which of the two distributed control implementation (two-node or three-node) is better. This measure is especially useful in sectors where there are time-to-market pressures. In this situation, design decisions may have to be made before the product is built. For the group that used the "simulation first" approach, this decision can be made after simulating the various systems. For the group that did not use the simulator, they had no choice but to implement both the options and decide on the "best" one.

- Hardware implementation effort.

  This is the effort spent by each group for software implementation on the microcontroller. For Group A, this is similar to the total effort involved. For Groups B and C, the effort involved at the simulation phase is ignored.

- Comparison of total simulation and total implementation effort.
  This is the comparison of the total effort spent at the simulation stage and the total effort involved in the implementation for Groups B and C only.
- Simulation development effort.
  This is the measure of the effort that Groups B and C spent at the various software simulation phases. The effort for the hardware implementation is ignored.
- Comparison of the simulation effort for a two-node and three-node systems.
  This is the measure of the individual simulation effort involved for a two-node pendulum controller and a three-node pendulum controller.

Please note that the analysis excludes the effort involved in the FEMENV stage.

All the participating groups managed to successfully implement a stable pendulum controller of their choice. Please note that Group A had to implement both the options to decide which system was "better". Since the other groups (B and C) used a "simulation first" approach, they could decide on which of the two options was "better" based on the simulation results, before implementing their desired solution. Group B and Group C decided to implement the two-node system as their final solution.

6.5.1 Overview of the results

Table 6-3 illustrates the overview of the results for Case Study 6. The comparison of the effort for the various approaches (hardware, TrueTime and TT-Plus) is presented in Table 6-4. The result indicates that for all cases, the TT-Plus group (C) had the most reduction of effort.

Table 6-3 The results for effort (in minutes) for Case Study 6.

	Effort of HW group (A)	Effort of TrueTime group (B)	Effort of TT-Plus group (C)
Effort to decide on the "best" system (excl. FEMENV)	1350	1365	525
Hardware implementation effort	1350	390	300
Total development effort (excl. FEMENV)	1350	1755	825

Table 6-4 Percentage of the effort involved for the different development approaches for Case Study 6.

	TrueTime compared to HW	TT-Plus compared to HW	TT-Plus compared to TrueTime
Effort taken to decide on the "best" system (excl. FEMENV)	1% more	61% less	62% less
Hardware implementation effort	71% less	78% less	23 % less
Total development effort (excl. FEMENV)	30% more	39% less	53% less

The individual results for each group are reported in detail in the following sections.

6.5.2 Total development effort

Based on the results obtained in Table 6-3, it is observed that for Case Study 6, the group that took the shortest time to complete the project was Group C. This was followed by Group A and finally Group B. The overall effort for Group B was 30% more than Group A while Group C took 39% less effort than Group A. These contradictory results will be discussed again in Section 6.6.

6.5.3 Effort taken to decide on the "best" system

This measure of effort represents the time taken by the groups to decide which of the two distributed control implementation (two-node or three-node) is "better".

From the results obtained in Case Study 6, it is observed that Group B took 1% more effort than Group A, while Group C took 61% less effort than Group A. These contradictory results will be discussed in Section 6.6.

6.5.4 Hardware implementation effort

Comparing the results of the effort spent at the hardware implementation stage for all the three groups in Case Study 6, the result shows that Group B and Group C took 71% and 78% less effort respectively when compared to Group A.

By comparing the two phases involved in the hardware implementation individually, it can be seen that Group B and C took much less effort compared to Group A (Figure 6-3) for both the phases.

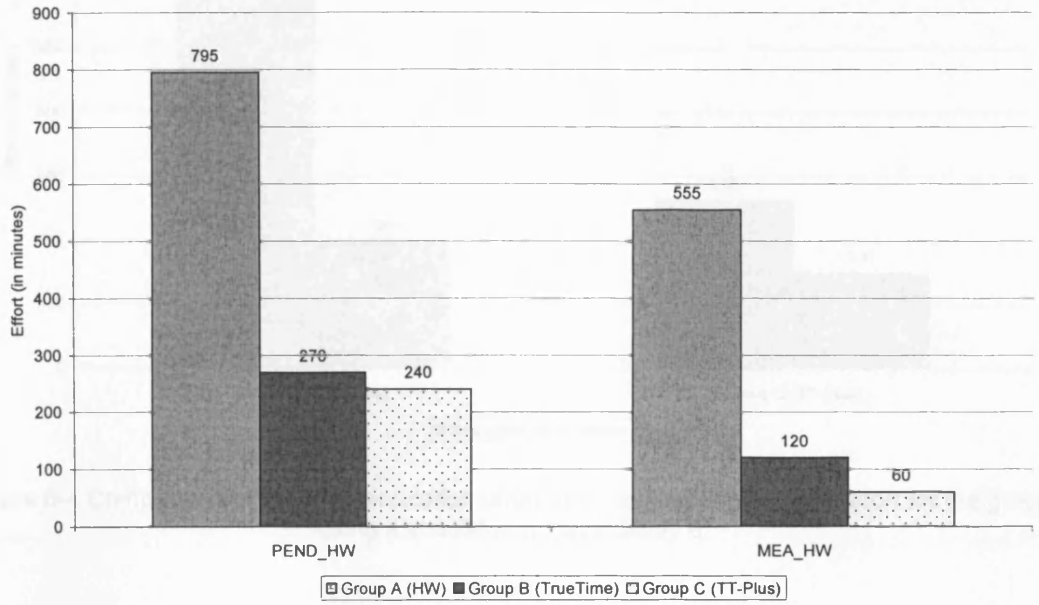

Figure 6-3 Hardware implementation effort for different stages (Case Study 6). Figure adapted from Ayavoo *et al.* (2006), Fig. 3.

6.5.5 Comparison of total simulation and total implementation effort

It can be seen from Figure 6-4 that the total implementation effort for Groups B and C was very similar. However, the total simulation effort for Group B was much higher than for Group C. This indicates that Group C had a more significant reduction of effort in developing the simulation models compared to Group B.

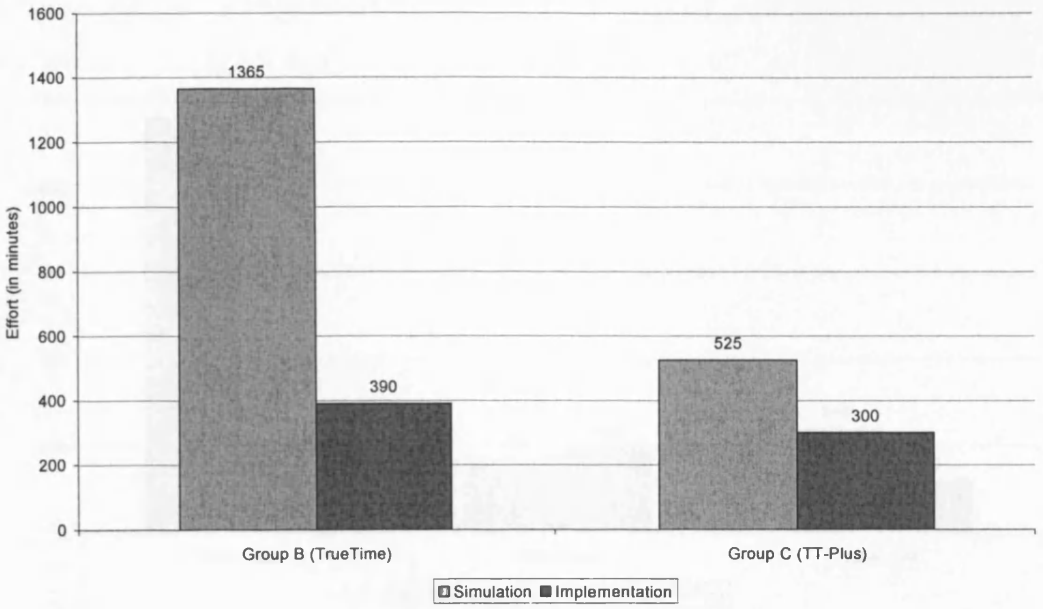

Figure 6-4 Comparison of the total simulation effort and total implementation effort for the groups using a simulator in Case Study 6.

6.5.6 Simulation development effort

This section considers the time taken to develop the simulation models for Groups B and C. The results in Table 6-2 show that Group C took 62% less effort when compared to Group B. The results also indicate that the time taken by Group C to familiarise themselves with the TrueTime simulation environment was about 62% less than Group B.

By comparing the three phases involved in the simulation development individually (Figure 6-5), the results indicate that the bulk of the effort reduction came from configuring the necessary communication messages (TXRX_SIM).

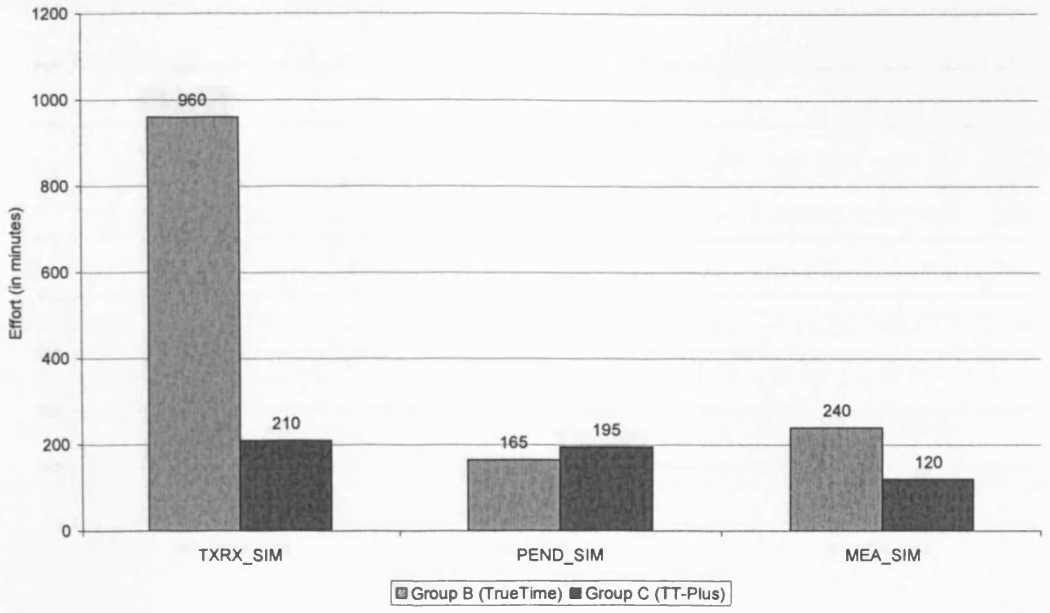

Figure 6-5 The simulation effort involved at different stages (Case Study 6). Figure adapted from Ayavoo *et al.* (2006), Fig. 4.

6.5.7 Comparison of the simulation effort for two-node and three-node systems

By comparing the total simulation effort involved for a two-node and three-node system individually for Case Study 6, the result indicates that the effort spent by Group C was less than Group B, by 65% and 50% respectively.

By comparing the two-node system at the different simulation phases, the result again indicates that most of the effort reduction for Group C came from the TXRX_SIM phase (Figure 6-6). The similar trend was also observed for the three-node system (Figure 6-7).

The results also indicate that the total effort to simulate a two-node system was more when compared to the three-node system. This was observed to be the case for both the simulation approaches.

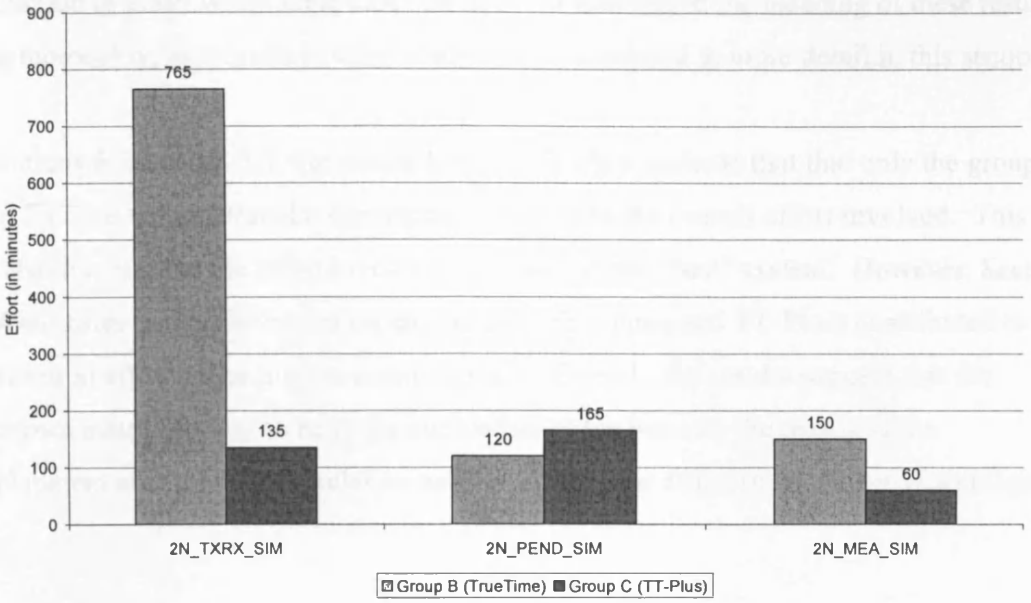

Figure 6-6 The simulation effort involved at different phases for a two-node system (Case Study 6). Figure adapted from Ayavoo *et al.* (2006), Fig. 5.

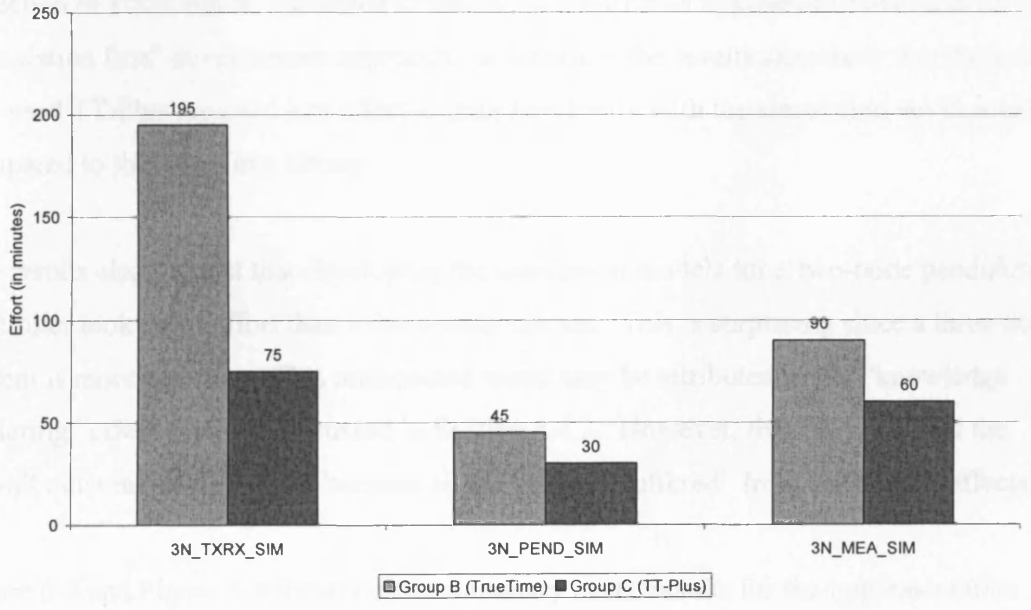

Figure 6-7 The simulation effort involved at different phases for a three-node system (Case Study 6). Figure adapted from Ayavoo *et al.* (2006), Fig. 6.

6.6 Discussion

Based on the analyses of the results obtained in Section 6.5, it is difficult to state definitively that simulation reduced the effort involved. This is because some cases show

a reduction of effort while some cases do not. To understand the meaning of these results more thoroughly, the results in Case Study 6 are considered in more detail in this section.

In Sections 6.5.2 and 6.5.3, the results for Case Study 6 indicate that that only the group using TT-Plus demonstrated a significant reduction in the overall effort involved. This was also the case for the effort involved to decide on the "best" system. However, Section 6.5.4 indicates that both simulation approaches (TrueTime and TT-Plus) contributed to a reduction in effort in the implementation phase. Overall, the results suggest that the difference mainly appear to be in the simulation phase because the trends of the development effort for the simulation models seem to be different for Group B and Group C.

By analysing the total effort involved in the simulation phase more closely, Group C had taken less effort compared to Group B. Specifically, the reduction in effort came from simulating the message transmission between the individual nodes (TXRX_SIM). This reduction of effort can be attributed to the use of TT-Plus as opposed to TrueTime for the "simulation first" development approach. In addition, the results also show that the group that used TT-Plus required less effort to gain familiarity with the simulation environment compared to the TrueTime Group.

The results also suggest that developing the simulation models for a two-node pendulum controller took more effort than a three-node system. This is surprising since a three-node system is more complex. This unexpected result may be attributed to the "knowledge gathering" effect that was discussed in Section 4.4.2. However, this did not affect the overall outcome of the results because all the groups "suffered" from the similar effects.

Figure 6-8 and Figure 6-9 illustrates the summary of the results for the implementation and simulation phases respectively.

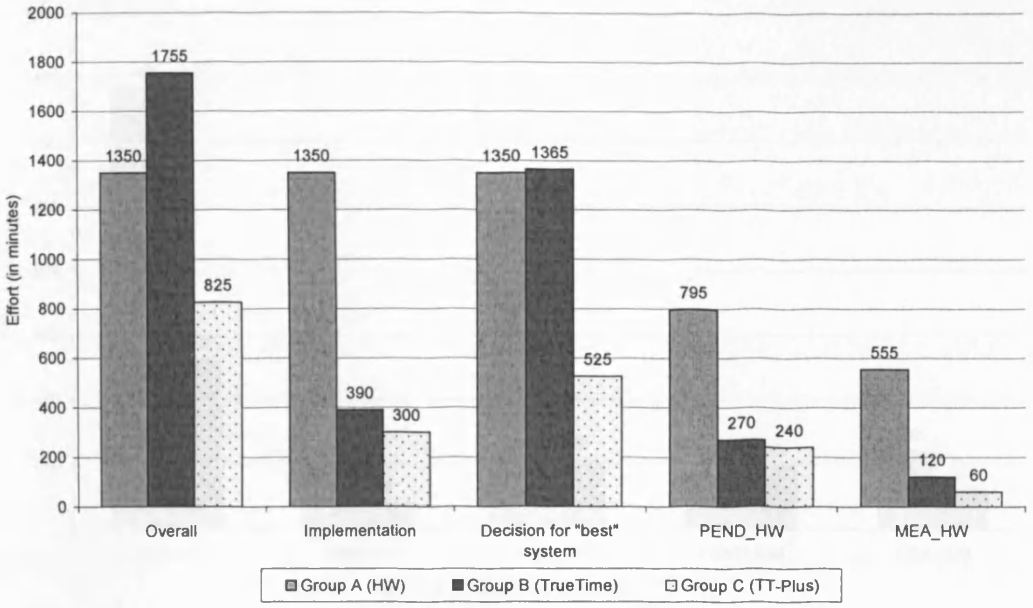

Figure 6-8 Summary of the implementation effort involved in Case Study 6.

A short summary of the results for each measure at the implementation stage is given below:

- Overall development effort.

  The group that used the TT-Plus simulator showed a reduction of the overall effort compared to the TrueTime group and the "hardware only" group.

- Implementation effort.

  The "simulation first" approach seems to benefit the implementation phase.

- Decision for the "best" system.

  The efforts for the TrueTime simulator group (Group B) were very similar to the effort of the "implementation only" approach (Group A). However, the TT-Plus simulation group demonstrated a reduction of effort in this area.

- PEND_HW effort.

  The simulation group showed a reduction of effort compared to the "hardware only" approach.

- MEA_HW effort.

  The simulation group showed a reduction of effort compared to the "hardware-only" approach in this category.

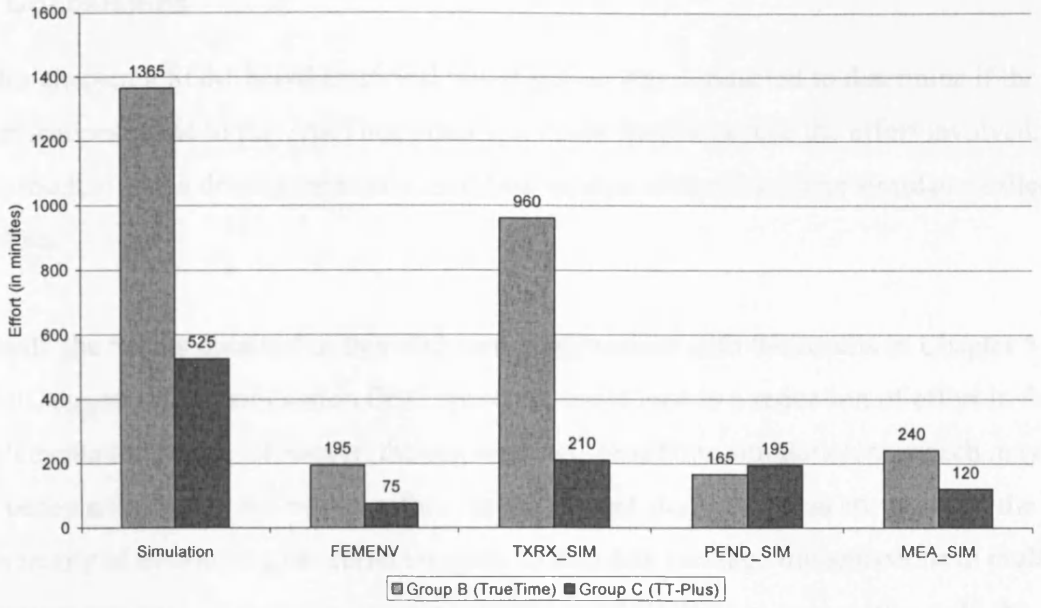

Figure 6-9 Summary of the simulation effort involved in Case Study 6.

A short summary of the results for each measure at the simulation stage is given below:

- Simulation effort.

  The results indicate that the TT-Plus group showed a reduction in the effort to develop the simulation models compared to the TrueTime group.

- FEMENV effort.

  The group using TT-Plus took less effort to familiarise themselves with the simulation environment compared to the TrueTime group.

- TXRX_SIM effort.

  The group using TT-Plus took less effort to develop the simulation models in this category.

- PEND_SIM effort.

  The results indicate that there were no significant differences in the effort for the simulation groups in this category.

- MEA_SIM effort.

  The result shows that the TT-Plus groups took slightly less effort compared to the TrueTime group.

6.7 Conclusions

In this chapter, a SGM-based empirical investigation was conducted to determine if the modifications made to the TrueTime simulator could further reduce the effort involved. This resulted in the development of a modified version of the TrueTime simulator called TT-Plus.

Overall, the results obtained in this study are in agreement with the results in Chapter 5 which suggest that a "simulation first" approach could lead to a reduction of effort in the implementation phase. However, the conventional TrueTime simulation approach may not necessarily reduce the overall effort. In the current study, this was attributed to the complexity of developing the correct models to simulate message transmissions in multi-processor systems. The results suggest that the use of TT-Plus in conjunction with the existing simulator can further reduce the effort involved in the development of simulation models for X-by-Wire systems. Hence, the overall effort was found to be reduced only if the effort required to carry out the simulation is limited.

The effort involved in learning the simulation environment and developing the simulation models depends on the characteristics and features of the simulator. With respect to this, the results suggest that a simulator that employs a design-led code-generation approach could potentially make it easier to create the relevant simulation models. Crucially, the effort involved can be further reduced by developing the simulation models at a high level of abstraction. The results suggest that TrueTime can provide the foundation to support this approach.

More generally, the SGM was effectively used to obtain evidence to suggest that the "simulation first" approach can be improved such that it can contribute to further reduction in the overall effort involved. However, with respect to the software development process, reducing the effort is necessary, but not sufficient to determine the overall efficacy of the approach. Indeed, as discussed in Chapter 4, the use of a simulator is also expected to improve the quality and reliability of the system. In light of this, the next chapter will explore the contributions that a "simulation first" approach can make towards software quality.

7. Can simulation improve software quality?

In Chapter 5 and Chapter 6, the SGM was used to investigate the effort involved in the development of embedded control systems. Although development effort is critical to any software development process, it is also often desirable that the software produced is of the highest quality. In this chapter, the "simulation first" approach is evaluated to investigate if it can improve the software quality.[23]

7.1 Introduction

On August 6th 1997, Korean Air flight 801 crashed into Nimitz Hill while attempting to land at Guam International Airport and 228 people lost their lives (NTSB, 2000). The results of subsequent research suggest that the accident could have been prevented had it not been for problems with the software in the "minimum safe altitude warning" (MSAW) system: specifically, it appears that software configuration changes were incorporated in the MSAW, and that the modified system was not re-tested before it was used (Greenwell, 2003). This, and other fatal software-related accidents such as the Panama radiotherapy accident (IAEA, 2001), the Patriot missile defence problem (GAO, 1992) and the Therac-25 incident (Leveson and Turner, 1993) highlight the importance of developing high-quality software, especially when it is to be employed in safety-related applications.

In this case, the development of embedded automotive control is often linked with safety-critical applications (von Hanxleden *et al.*, 1998; Zheng *et al.*, 2004; Kandasamy *et al.*, 2005). As a consequence, it is important that the use of a "simulation first" approach in the development of such applications produce high-quality embedded software systems. Indeed, some researchers have suggested that simulation can achieve this (see El-khoury and Törngren, 2001; Cervin *et al.*, 2003; Henzinger *et al.*, 2003), although very little empirical evidence is available to support such claims.

In light of these discussions, the focus of this chapter is to investigate the contributions that a "simulation first" approach can make towards the software quality of the embedded system. Some background work on software quality is presented in Section 7.2. The evaluation of software quality (Case Study 7) is illustrated in Section 7.3.

[23] Parts of this chapter is currently being prepared for publication in Ayavoo *et al.* (in preparation).

7.2 Background on software quality

According to Jørgensen (1999), software quality is commonly defined by the following three statements:

- Software quality is determined by a set of quality indicators (such as ISO 8402-1986 and IEEE 610.12-1990).
- Software quality is determined by user satisfaction.
- Software quality is determined by errors of unexpected behaviour of the software.

Much research has been carried out over recent years on software quality measurements (see Schneidewind, 1992; Coleman *et al.*, 1994; Henry *et al.*, 1994; Bevan, 1999; Briand *et al.*, 2000; Amasaki *et al.*, 2005; Thwin and Quah, 2005). However, it remains the case that a universally accepted definition of software quality has proved elusive. Jørgensen (1999) suggests that this is mainly due to the complex nature of what is perceived as being "software quality".

For example, Martin and Shafer (1996), mapped out a general framework to assess software quality (see Figure 7-1). Later, Bevan (1999), illustrated a slightly different framework to express software quality from a user's perspective (see Figure 7-2). Similarly, Kan *et al.* (1994) described software quality from various perspectives such as the customer satisfaction, the product quality and the process of the product development.

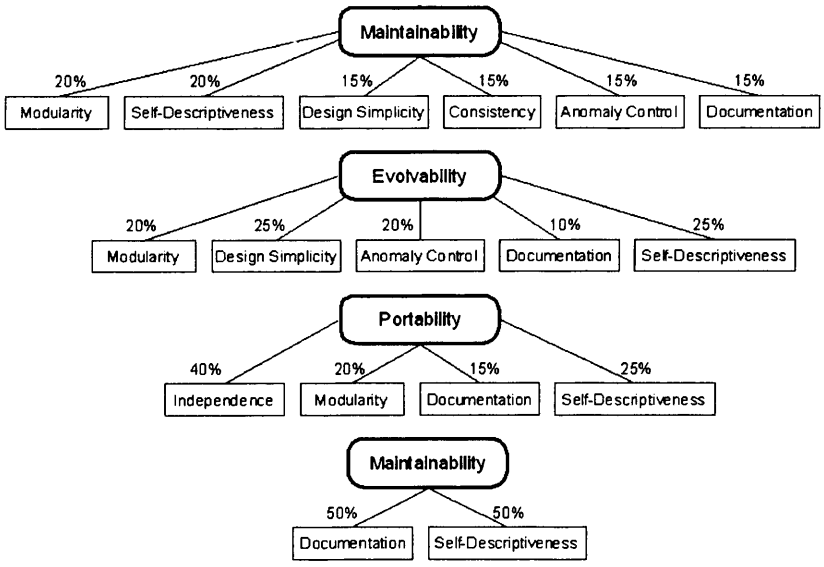

Figure 7-1 An example of a framework for software quality. Redrawn from data in Martin and Shafer (1996), Figure 3.

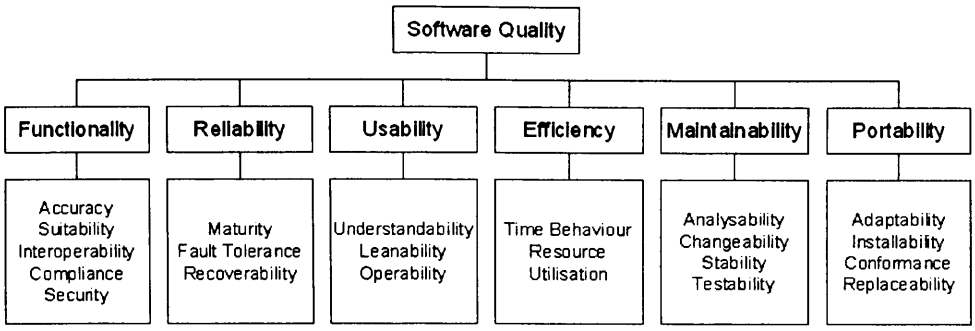

Figure 7-2 The ISO/IEC 9126 software quality characteristics from a user's perspective. Redrawn from data in Bevan (1999), Fig. 1.

Besides this, quality measurements can also be influenced by the software technology in use. For example, in Basili *et al.* (1996), the authors described some traditional metrics (such as McCabe's cyclomatic complexity measure) as being inappropriate for assessing the quality of O-O software development. Instead, they attempted to validate metrics specific to the O-O technology such as coupling, member functions and number of children (these metrics were initially proposed by Chidamber and Kemerer (1994)).

As a consequence, given a piece of software, it is very difficult to allocate an overall quality value. However, it is generally considered to be possible to make specific comparative measurements from (say) two pieces of software and thereby determine which is of the highest quality (Jørgensen, 1999). This approach is described in more detail in Section 7.4.

7.3 Case Study 7: Evaluation of software quality

Case Study 7 was carried out to evaluate the contributions of a "simulation first" approach to software quality. This required the development of a suitable embedded application to carry out the comparisons between a "simulation first" approach with a "hardware only" technique. Instead of carrying out a new empirical investigation, it was decided that the data from Case Study 6 (described in Chapter 6) was to be used for the following reasons:

- Case Study 6 is suitable as it involved the development of an X-by-Wire system where reliability is essential (Edwards *et al.*, 2004; Bautista *et al.*, 2005).
- Case Study 6 had compared the development of an inverted pendulum controller using three different development methodologies ("hardware only", TrueTime and TT-Plus); two of them being a "simulation first" techniques and one being a "hardware-only"

approach. This makes the data from Case Study 6 suitable to compare the impact that a "simulation first" approach may have on the development of high-quality software for reliable X-by-Wire systems.

- By employing the SGM, a repository of all the data collected from the previous study was readily available. This allowed for the reanalysis of the results obtained from the perspective of software quality.
- It was cost effective to use the data from Case Study 6.

Hence, an inverted pendulum was used as the testbed, and the student groups were required to decide if a two-node or three-node pendulum controller would be "better" (please refer to Section 3.3 for details on the testbed and Section 6.3 for details of the case study).

7.4 The measurement of quality indicators

Having decided on a suitable case study in the previous section, the next step was to extract the measurements that indicate the quality of software. To do this, the measure of quality must first be identified.

Ideally, it would be preferable to measure the quality of the software implementation of each group by looking at issues such as the error density (errors per 1000 lines of code), field stability (problems per user months) and percentage of customer satisfaction (see Endres and Rombach, 2003). However, evaluating these quality factors are not always feasible, primarily because they rely on the product to be used by customers over a period of time. As argued in Chapter 4, this is not always possible due to – for example – the rapid growth of the technology and the difficulty of employing suitable test subjects. Moreover, checking for failures in this study might not be a suitable metric to measure quality since all groups had successfully completed the task to control the pendulum (see Chapter 6).

One way to measure quality in the present study was to compare the results of the control delay obtained from the software simulation and the hardware implementation. Besides this, since all groups were initially given the same sample code to start from, then in an ideal case, all groups that completed the study should produce the same source code.

However, given that the groups used different development approaches, it was expected that variations in the source code would occur as a result of this.

Therefore, the approach here was to use indirect measurements to indicate the quality of the overall system. In the current study, four code-based metrics[24] were chosen as a measure of the quality indicators for the implemented system:

- Software size.
- Software complexity.
- Software modularity.
- Software stability.

Please note that in this study, these quality indicators were not used to analyse the software quality for a three-node system. This is because the simulation groups decided to implement the two-node pendulum controller as their final solution.

7.4.1 Software size

"Software size" can be used to predict the likely reliability of a particular system. Put simply, the more software there is, the greater the risk of errors (Rosenberg *et al.*, 1998), with the consequence that – given two systems with equivalent behaviour created using the same programming language – it could be argued that the system implemented using less code is likely to have code of higher quality.

The number of non-commented lines of source code (LOC) was used to measure the size of the software implementation. This measurement had previously been used in other studies (see Coleman *et al.*, 1994; Stark *et al.*, 1994). In the present analysis, only the lines that contributed to 'C' statements were counted; blank lines and comments were ignored.

7.4.2 Software complexity

The complexity of the software can be used to determine the ease of maintainability for a particular piece of source code (Martin and Shafer, 1996). As the complexity of the

[24] Please note, that the student subjects were not informed of these metrics as being the method of evaluating software quality, since this could have biased the outcome of the results.

software increases, the ease of maintainability decreases (McCabe, 1976). Hence, given that two pieces of software have the same behaviour, then it could be argued that the source code with lower level of complexity is likely to be easier to maintain.

To measure the complexity of the software implementation, McCabe's Cyclomatic Complexity was used (McCabe, 1976; Munson and Khoshgoftaar, 1992), where $v(G)$ is the cyclomatic complexity and π is the number of conditions.

$$v(G) = \pi + 1$$

This translated to calculating the number of conditional statements in each task and adding 1 to it. The complexity of all the tasks were then added together to represent the overall source code complexity. Please note that in this case, the complexity of the system scheduler was ignored, as all groups used the same scheduler without performing any modifications to it.

7.4.3 Software modularity

The modularity of the software can be used to determine the portability and maintainability of the software solution (Martin and Shafer, 1996). One way to determine the modularity of software is to measure the coupling level. As the coupling level decreases, the modularity of the system tends to improve (Rosenberg and Hyatt, 1997).

To measure the software modularity, a scoring system was used to quantitatively assess the coupling between the different modules. For each task that was observed to be sharing the same function or variable with another module, a mark of +1 was given to indicate the level of coupling in the system. For example, Listing 7-1 illustrates an example source code in 'C'. The example here shows that there is an external variable used locally within this task (Var_A1). In addition, Task_K also calls an external function from another file (Function_A1). Hence in this example, the coupling level of Task_K is two.

```
#include "File_A1.h"     // Includes an external file
extern int Var_A1;       // <--Uses an external variable

void Task_K ()           // Task K
   {
   int x = 5;            // Local variable
   if (Var_A1 == x)      // Condition
      {
      Function_A1();     // <--Calls an external function
      }
   }
```

Listing 7-1 An example code illustrating how coupling in 'C' source code was calculated.

7.4.4 Software stability

The stability of the software development process may provide an indication of the reliability of the system. According to Schneidewind, 2004, p. 14, "*A high rate of software change can be detrimental to software reliability*". Indeed, from a software maintenance point of view, the reliability tends to be at a maximum when the maintenance of the code stabilises (Schneidewind, 1999). It has also been previously shown that an increase in small source code changes made to a module is likely to introduce errors into the system (Purushothaman and Perry, 2005). In the present study, it could be argued that high number of changes made to the source code may indicate the lack of clarity from the developer's point of view in implementing the system. As a consequence, this could affect the stability of the software development process.

To measure stability, the number of changes made to a particular segment in the source code was used. Specifically, the stability of the software was analysed in terms of the number of adaptive, corrective and perfective changes made throughout the software implementation stage. Such technique was adapted from a previous study that analysed small source code changes (see Purushothaman and Perry, 2005). Adaptive changes were attributed to changes that involve adding new functionality to the system. Corrective changes were generally made to fix defects in the software. All other changes that contributed to the enhancement of performance were categorised as perfective changes.

7.5 Analysis of the results from Case Study 7

After identifying and extracting the relevant measures for software quality, the results can now be analysed. The results and analyses for Case Study 7 are presented in this section.

7.5.1 Comparison of the control delay for simulation and implementation

To compare the quality of the results for the simulated systems with the "actual" hardware testbed, the measurement of control delay (see Sandfridson, 2000) was used. The results in Table 7-1 indicate that the simulated results of the groups using TrueTime closely matched the implementation results. However, it can also be observed that the simulated result of the three-node pendulum controller for Group B was incorrect (the control delay was too short).

Table 7-1 The control delay (in ms) recorded by all the groups in Case Study 7.

Case Study 7	Group A		Group B		Group C	
	SIM	HW	SIM	HW	SIM	HW
Two-node min	x	15.926	16.307	15.946	16.307	16.315
Two-node max	x	16.726	17.086	16.714	17.086	17.109
Max – Min	x	0.800	0.779	0.768	0.779	0.794
Three-node min	x	29.613	16.467	x	25.018	x
Three-node max	x	29.615	17.246	x	25.018	x
Max - Min	x	0.002	0.779	x	0.000	x

This may suggest that the group that used the basic TrueTime simulator developed simulation models that were susceptible to errors. On the other hand, the group using TT-Plus have a higher probability of producing more robust simulation models.

7.5.2 Software size

Based on the non-commented LOC produced for the groups that implemented a two-node pendulum control (see Figure 7-3), the result shows that the group that produced the largest code size was Group A, followed by Group B and finally Group C. This may suggest that a "simulation first" approach can support in the production of software which is smaller in size.

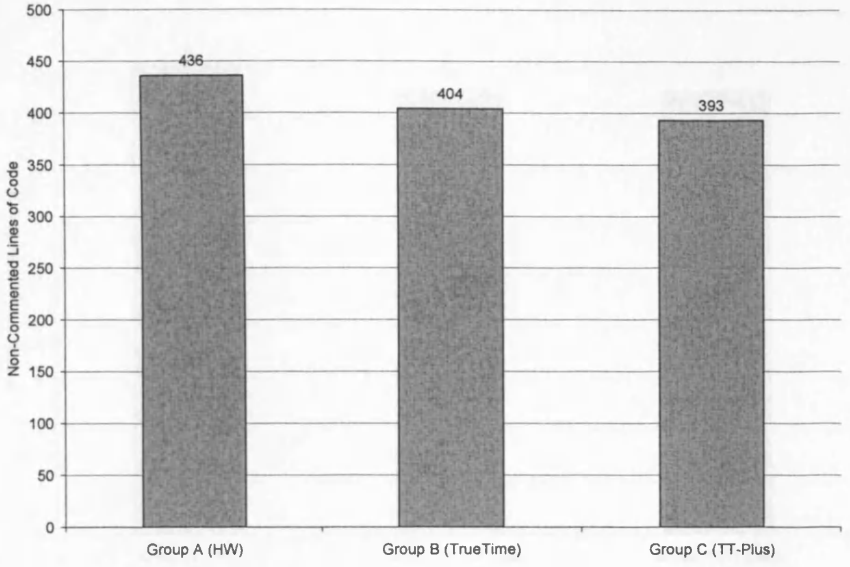

Figure 7-3 Results of the implemented lines of code for each group in Case Study 7.

Please note that the software size shown in the graph is not representative of the real software size, but rather the sections that the students had worked on. However, the students were required to understand the behaviour of the entire software in order to make the necessary modifications to selected sections of the source code.

7.5.3 Software complexity

Based on McCabe's Cyclomatic Complexity, the result in Figure 7-4 shows that there was very little difference between all the groups. This may suggest that simulation did not have a significant impact in helping the developers to produce source code of lower complexity.

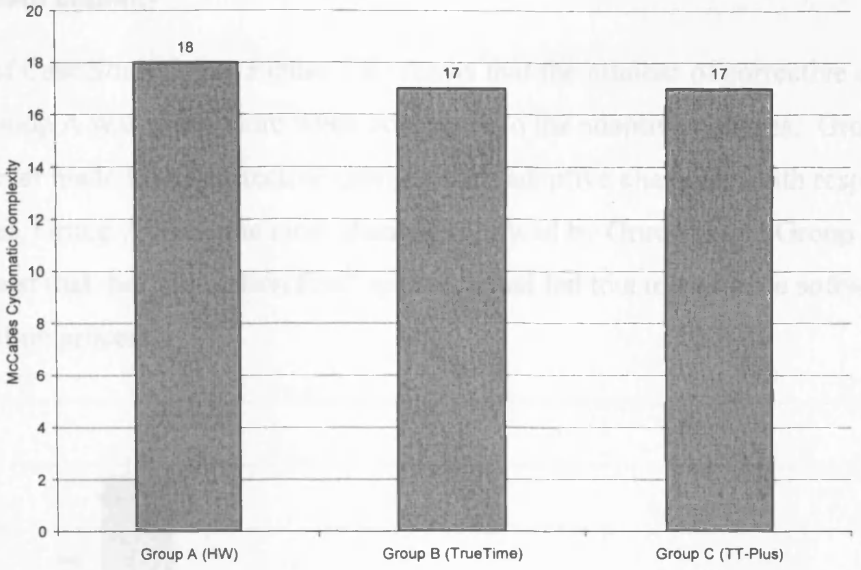

Figure 7-4 Results of McCabe's Cyclomatic Complexity for each group in Case Study 7.

7.5.4 Software modularity

After analysing the modularity of the source code produced, the results suggest that the group that produced the most modular code was Group C, followed by Group B and finally Group A (see Figure 7-3). This result may suggest that simulation had assisted the developer to produce modular source code.

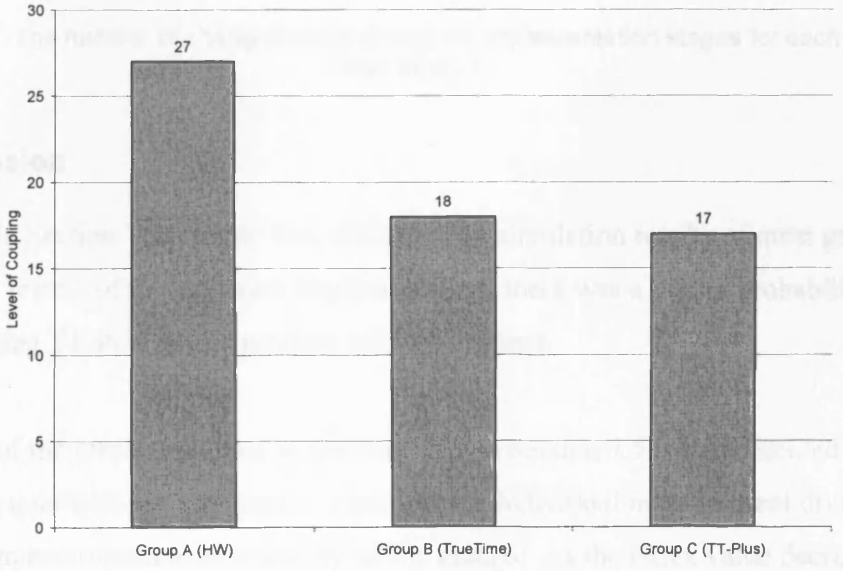

Figure 7-5 The level of source code coupling for each group in Case Study 7.

7.5.5 Software stability

The result of Case Study 7 (see Figure 7-6) shows that the number of corrective changes made for Group A was much more when compared to the adaptive changes. Groups B and C however made fewer corrective changes than adaptive changes. With respect to the total changes, Group A made the most changes followed by Group B and Group C. These results suggest that the "simulation first" approach had led to a more stable software implementation process.

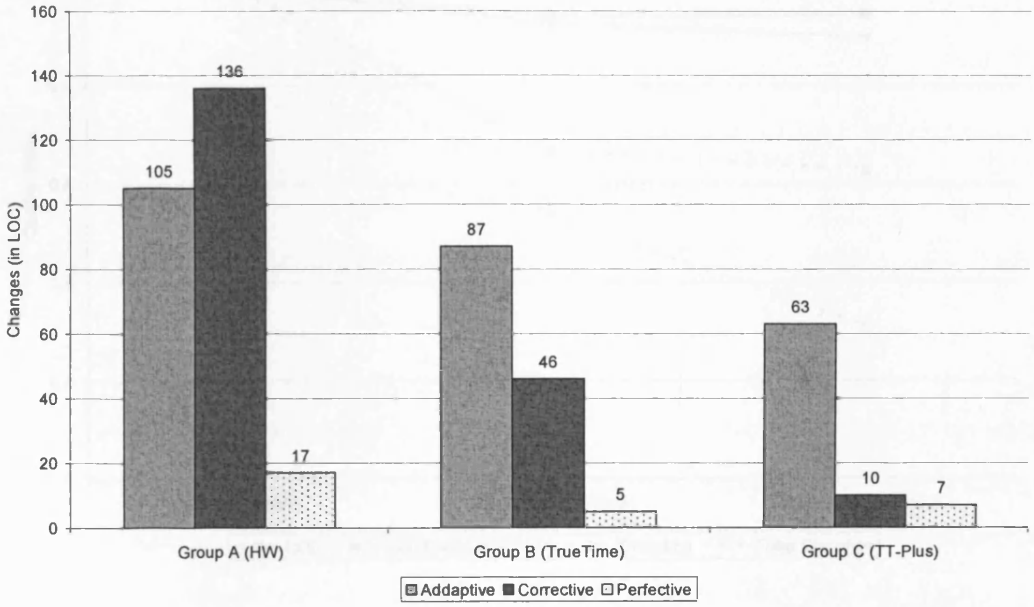

Figure 7-6. The number of changes made during the implementation stages for each group in Case Study 7.

7.6 Discussion

The results in Section 7.5.1 imply that, although the simulation results of most groups matched the results of the hardware implementation, there was a higher probability that the group using TT-Plus would produce reliable systems.

Summaries of the results obtained in Section 7.5.2 to Section 7.5.5 are presented in Figure 7-7. Here, a quality index was used to represent the individual measurement divided by the maximum measurement recorded by all the groups. As the index value decreases, the quality of the software improves.

Although the variations between the groups was small, the trend of these results suggest that the groups that used a "simulation first" approach produced software of higher quality compared to the "hardware implementation only" groups. The results also suggest that the use of simulation in conjunction with code-generation techniques tends to result in higher software quality.

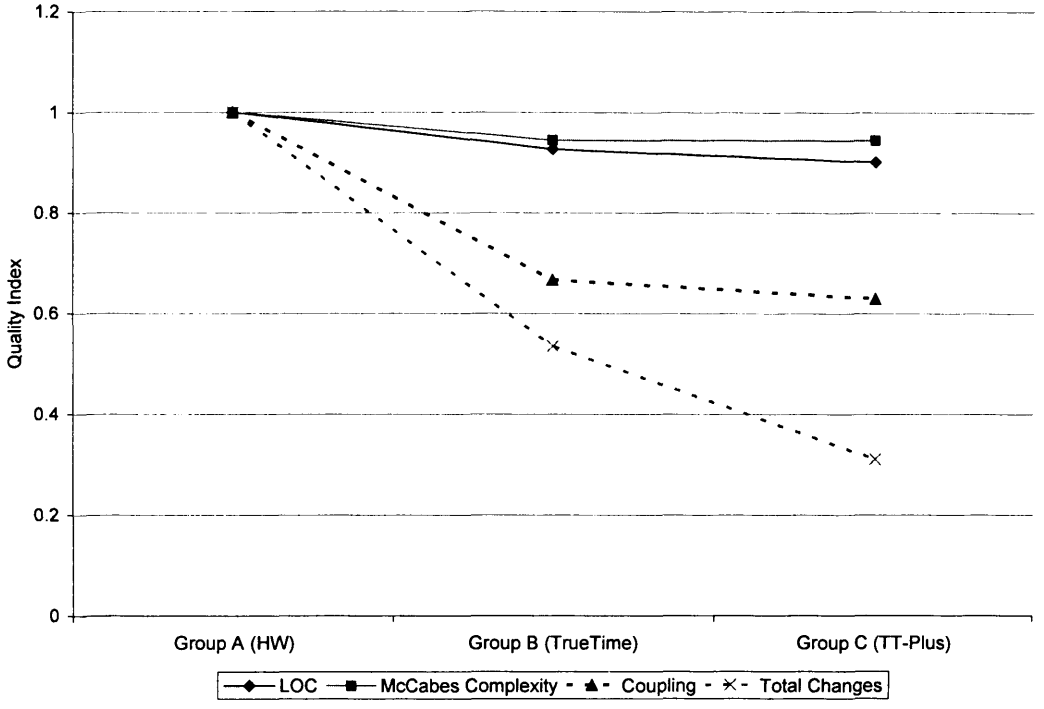

Figure 7-7 Trend of the overall results of software quality for a two-node system in Case Study 7.

By plotting the results of the total number of changes made (software stability) to the effort involved in the implementation stage for each group, the results in Figure 7-8 suggest that these two measurements are closely related. This can be seen from the similar trend for both measurements.

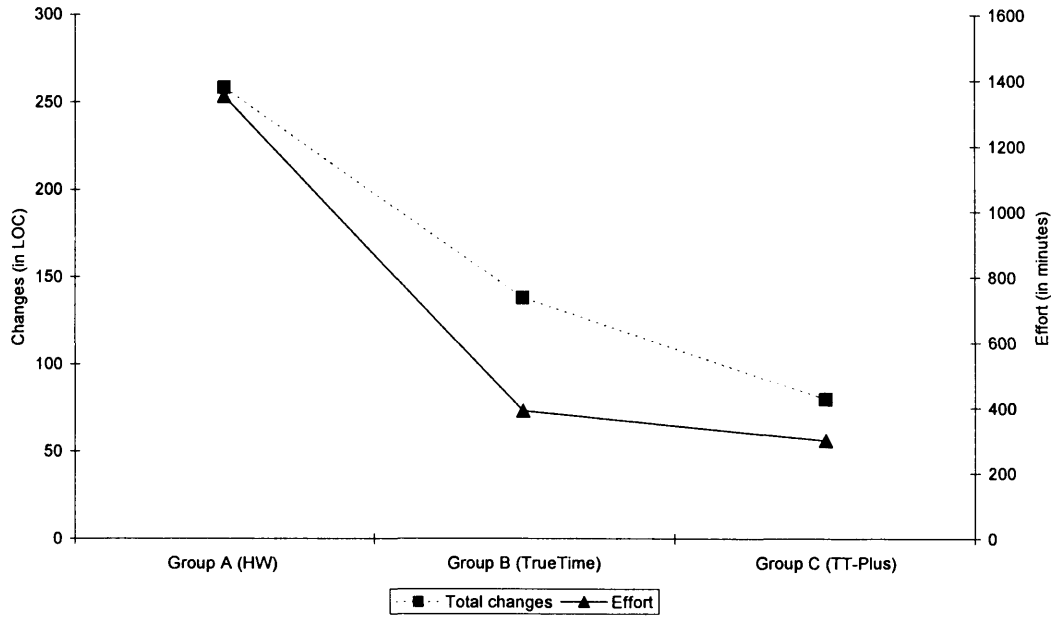

Figure 7-8 Trend of changes made and implementation effort for all the groups in Case Study 7.

7.7 Conclusions

The aim of the work presented in this chapter was to explore the impact that a "simulation first" approach may have on the quality of the software for reliable X-by-Wire systems. Overall the result suggest that such an approach may lead to the creation of high-quality software (when compared to software produced using a "hardware only" approach). The results from Case Study 7 also imply that a "simulation first" approach that incorporates code-generation techniques has the potential to assist in the implementation of high-quality software. Similarly, there is evidence that a "simulation first" approach that incorporates code-generation techniques is likely to produce more reliable simulation models. Finally, there is some evidence to suggest that the stability of the software development is closely related to the effort involved.

Up till now, the SGM has been used (in Chapters 5, 6 and 7) primarily to evaluate the efficacy of a "simulation first" approach in the development of X-by-Wire systems. For the SGM to be useful, it must be capable of being applied to a wider range of studies. In order to explore whether the SGM can be applied more widely, Chapter 8 describes a case study that employs the SGM in a non-simulation investigation.

8. Can the SGM be applied more widely?

In previous chapters, the SGM has been employed to explore the efficacy of a "simulation first" approach in the development of embedded control systems. In this chapter, an investigation is carried out to explore the extent that the SGM can be applied to other studies.[25]

8.1 Introduction

The aim of the work described in previous chapters was to explore a "simulation first" approach to software development for embedded systems. To do this the SGM was employed. However, the SGM has not yet been applied to a broader range of research studies that explores other development methodologies for embedded systems. This issue is of particular interest since the SGM is intended to be a "general" technique that can be applied to a wide range of studies. To substantiate this claim, it is necessary to demonstrate that the SGM can be employed in a non-simulation study.

To this end, research in the ESL has been exploring the use of automated pattern-based code-generation techniques to support the implementation of reliable embedded systems (see Mwelwa *et al.*, 2003; Mwelwa *et al.*, 2004a; Mwelwa *et al.*, 2004b; Mwelwa *et al.*, 2005; Mwelwa *et al.*, 2006). Design patterns can be viewed as a collection of reusable software (and hardware) solutions for implementing a range of embedded systems. A review on some of the previous work on design patterns is provided in Appendix D. To assist in the implementation of the patterns for embedded systems, a code-generation approach was used. To this end, a tool called PTTES Builder (see Mwelwa *et al.*, 2005; Mwelwa *et al.*, 2006) was developed in the ESL to assist the developer of embedded systems in choosing the appropriate patterns and automatically generating the necessary source code (called Pattern Implementation Examples, or PIEs) for the system implementation process.

Although the research has demonstrated many technical virtuosities of the approach, no attempt has so far been made to evaluate the contributions of the tool when used in practice. As such, Case Study 8 investigates the extent to which SGM can be used to

[25] Parts of this chapter also appears in Pont *et al.* (submitted).

assess the impact of a pattern-based code-generation approach on the development of embedded systems.

8.2 Case Study 8: Evaluation of the PTTES Builder

As stated briefly in Section 8.1, the aim of the work described in this chapter was to consider the extent to which the SGM can be employed to a wider range of studies. To do this, the automated pattern-based code-generation technique to implement embedded systems was used as the case study. Specifically, the SGM was used to investigate the contributions that PTTES Builder may have in the development of embedded systems. Figure 8-1 illustrates the user interface of the PTTES Builder.

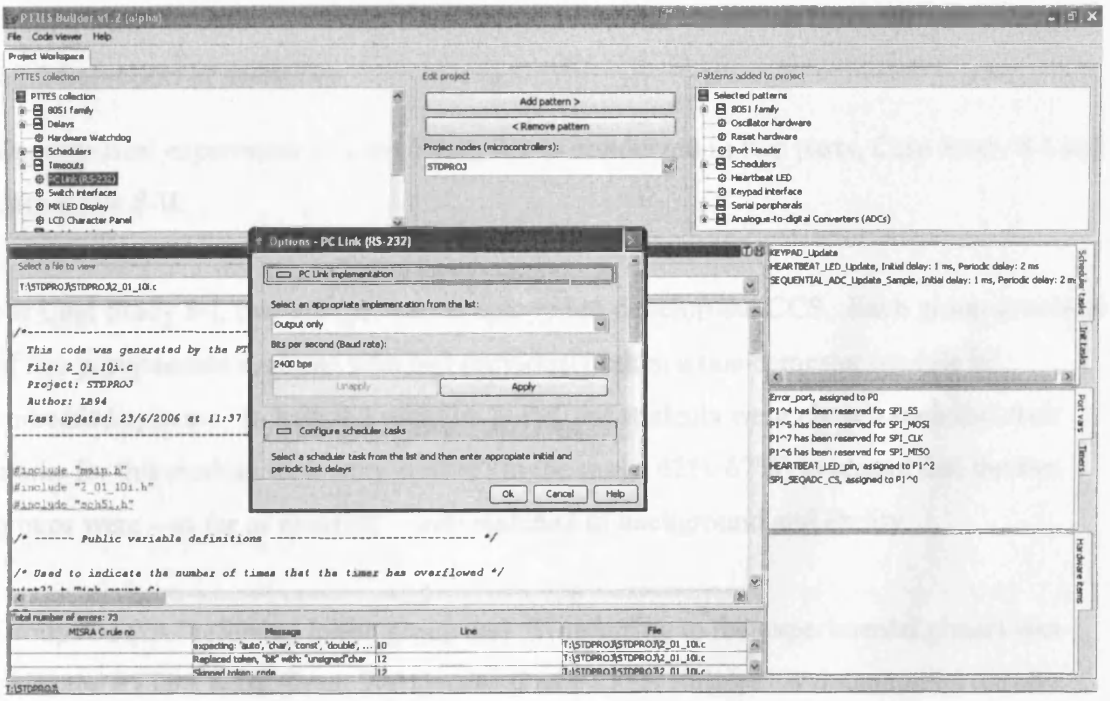

Figure 8-1 Example of the user interface for the PTTES Builder.

The study involved the comparison of two different development approaches of a non-trivial embedded application, one using PTTES Builder and the other using a manual approach (that is, only having access to the PTTES language in the book form). This required the use of an appropriate testbed.

8.2.1 The testbed

The testbed used in this study was based on a similar HIL automotive cruise-control system (CCS) testbed described in Chapter 3 and Chapter 5. In the current study, the CCS testbed consists of a single 8051 Infineon C515C microcontroller. An 8-bit 8051 microcontroller was used in this case since the PTTES Builder supports embedded code generation for that family of microcontrollers.

Please note that the tasks used in this case study was similar to that described in Table 3-1 (see Section 3.2.2). The exception being an additional task (Display Ref Speed) which was used to update and display the desired car speed on the terminal emulator every second.

8.2.2 Selection of subjects

The empirical experiment in Case Study 8 was conducted in two parts, Case Study 8-I and Case Study 8-II.

For Case Study 8-I, two groups were employed to develop the CCS. Each group consisted of two postgraduate students who had previously taken a one-semester module in embedded systems. In keeping with the SGM, the students were chosen such that their marks for this module were very similar (in the range 63%-67%) to ensure that the two groups were – as far as possible – well matched in background and ability.

Group A-I ("A" referring to the group and "I" referring to the experimental phase) was given the PTTES book (Pont, 2001) as well as PTTES Builder. A documented tutorial was also accompanied with the tool. The group was specifically asked to use PTTES Builder to implement the CCS. Group B-I was directed to implement the same system, but only with the help of the PTTES book.

The version of PTTES Builder used in these studies supported eight patterns while the associated book (which is around 1000 pages long) describes 72 patterns. As a consequence, it could be argued that the users of the book had the additional challenge of identifying the most appropriate patterns to use.

In order to address this potential bias, Case Study 8-II was carried out. In Case Study 8-II, four new groups of students were selected (average mark in the range of 70% - 80%). Each group consisted of a single student that had previously undertaken two modules in embedded systems. The students were again selected such that their ability and experience in embedded systems was similar.

As with Case Study 8-I, Groups A-II and B-II were asked to use PTTES Builder and the PTTES book to implement the CCS, while Groups C-II and D-II were asked to only use the PTTES book. However, in this phase, the books handed out to all the groups were marked to identify the eight patterns required (and that were supported by PTTES Builder). Groups C-II and D-II were also given the relevant PIEs (example code) for the patterns.

It was made clear to all the groups in Case Study 8-II that the CCS was to be implemented using the following eight patterns:

- EXTENDED 8051 pattern.
  The pattern describes the features of an Extended 8051 processor such as additional memory, on-chip ADC and the number of package pins.
- CO-OPERATIVE SCHEDULER pattern (p_Sch).
  This pattern describes how a time-triggered co-operative scheduler can be used for an application with the relevant source code.
- PORT WRAPPER pattern (p_Port).
  This pattern describes the use of a port library to declare the necessary port pins that are used for I/O.
- HEARTBEAT LED pattern (p_LED).
  The pattern illustrates how a Heartbeat LED can be implemented to indicate the status of a system.
- HARDWARE PULSE Count pattern (p_Counter).
  The pattern discusses the use of an onboard hardware counter to count external pulses.
- PID CONTROLLER pattern (p_PID).
  The pattern discusses how a PID algorithm can be implemented on the 8051 to perform real-time control.
- SEQUENTIAL ADC pattern (p_ADC).

The pattern shows how an onboard ADC can be configured to be used on an 8051 microcontroller.

- PC LINK (RS232) pattern (p_RS232).
  This pattern discusses the communication between an 8051 microcontroller and a desktop PC using the RS232 communication protocol.

8.2.3 Software Metrics

As described in Section 4.4.4, the Goal-Question-Metric (GQM) methodology (Basili and Weiss, 1984) was used to identify the necessary measurements to be collected, as illustrated in Table 8-1. Here, it is shown how the goal of the study led to some basic questions that resulted in several metrics that could be measured from the study.

Table 8-1 The GQM approach used in Case Study 8.

Goal	To evaluate the effectiveness of PTTES Builder tool for embedded software development.	
Questions	Can the tool reduce the effort? • Can the tool reduce the overall development effort? • Can the tool reduce the effort involved to implement the patterns? • Can the tool reduce the effort required to produce a functionally-correct system?	Can the tool produce better code quality? • Can the tool contribute to a more stable source code development process? • Can the tool help to produce a functionally correct system? • Does the tool produce source code that is more maintainable and portable?
Metrics	• Development time • Test case compliance ratio	• Source code changes • Test case compliance ratio • Software modularity

A description of each of these metrics is provided below:

- Development time: This is the measure of the amount of time taken for the software developer to implement various code segments (see Section 5.2.3).
- Test case compliance ratio: This is the measure of the number of passed test cases to the total number of test cases. Measurements of software reliability has previously been carried out in this way by Bassin *et al.* (2002). In the present study, a total of 26 test cases were created based on the requirements specification (see Appendix G for

details of the testcases). The higher the compliance ratio, the closer the solution is to a functionally-correct system.

- Software code changes: This is the measure of the number of changes made to a particular segment in the source code. In the current study, the PIEs used have already undergone rigorous testing and verification procedures. Modifying the PIE is therefore likely to make the source code "less reliable". Therefore, extensive source code changes to an existing PIE could potentially introduce more bugs into the software. It is concluded that - as the number of changes made to the PIEs increases - the system is more likely to contain errors.
- Software modularity: This is the measure of the coupling of the various modules in the system with respect to files, functions and variables (as discussed in Section 7.4.3). For each module observed to be sharing the same file, function or variable with another module, a mark of +1 is given to indicate the level of coupling in the system. As the coupling level decreases, the modularity of the system tends to improve (Rosenberg and Hyatt, 1997). Software modularity in turn can be used to indicate the maintainability and portability of the system (Martin and Shafer, 1996).

8.2.4 Observations and measurements

The following observation and measurement techniques were used in this study:

- Progress observation.
- Email.
- Questionnaire and interview.

Please refer to Section 4.4.5 for descriptions of these techniques.

8.3 Analysis of the results for Case Study 8

After the experiments, the results obtained from Case Study 8 (source code, completed progress forms and recorded interviews) were analysed.

8.3.1 Overview of the results

All groups developed the CCS system using the relevant PIEs, as illustrated in Figure 8-2.

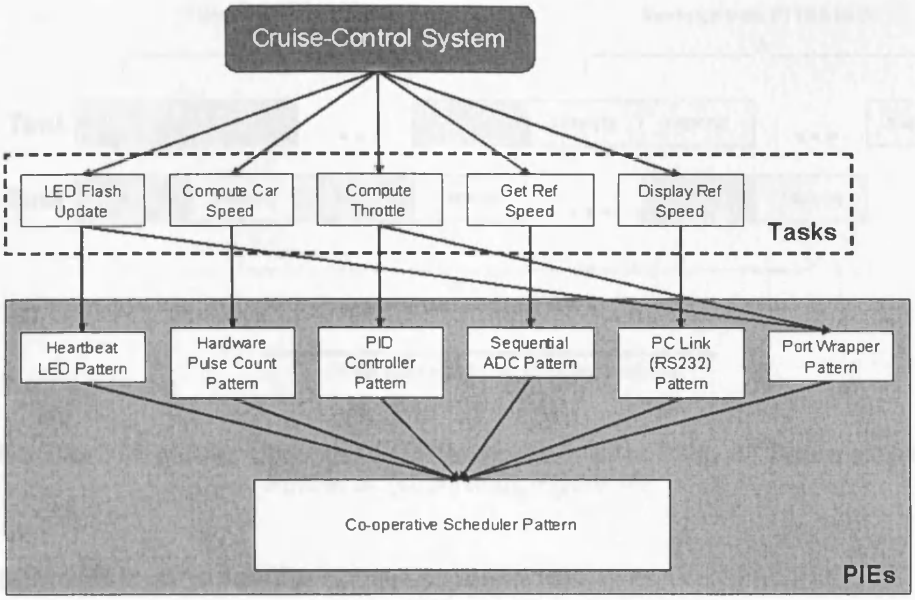

Figure 8-2 Developing the CCS using the associated PIEs for Case Study 8. Figure adapted from Pont *et al.* (submitted), Figure 15.

An initial observation suggests that the development characteristic of the groups using PTTES Builder and the groups using the PTTES book was different (see Figure 8-3). Based on the progress form, it was observed that the groups using PTTES Builder generally began developing their source code by using appropriate PTTES patterns as a starting point, before writing the relevant tasks[26] for the CCS. By contrast, the groups that did not use the tool began developing their source code in parts. It was also observed that the groups that used the tool initially spent approximately 60 minutes using it; 15 minutes of which was spent going through the PTTES Builder tutorial. For these groups, approximately 75% of the system's final source files were generated by the tool.

[26] The tasks had to be implemented manually as they were application specific and therefore not directly supported by the PTTES language. In this study, the groups were provided with documentation of the necessary formulae and variables required to implement these tasks.

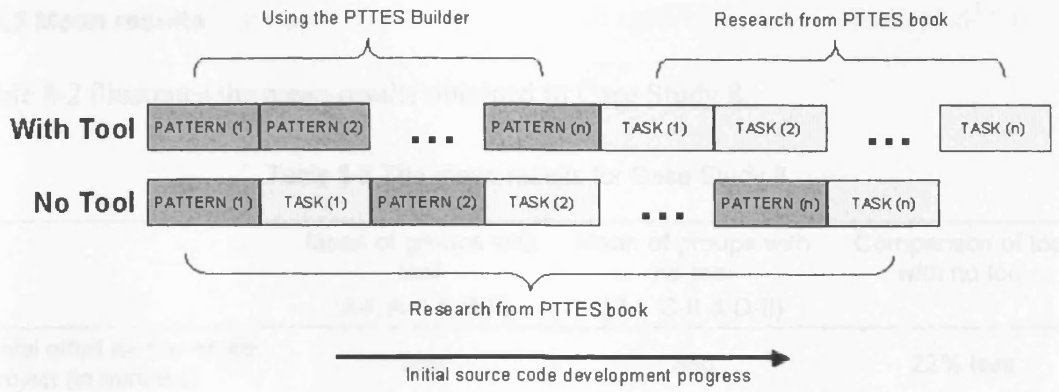

Figure 8-3 The CCS development phases for the teams in Case Study 8. Figure adapted from Pont *et al.* (submitted), Figure 16.

8.3.2 Synchronising the results

In order to determine whether PTTES Builder had been effective in the embedded software development process, synchronisation of the development phases was carried out (as described in Section 4.4.6). The analysis was conducted by dividing the development of the CCS into several development phases, as described:

- Implementing the scheduler pattern (p_Sch).
- Implementing the port library pattern (p_Port).
- Implementing the blinking LED pattern (p_LED).
- Implementing the hardware pulse counter pattern (p_Counter).
- Implementing the PID control algorithm pattern (p_PID).
- Implementing the PC link RS232 pattern (p_RS232).
- Implementing the task to compute the car speed (t_CSpd).
- Implementing the task to obtain the new throttle position (t_CThr).
- Implementing the task to obtain the set (or reference) speed (t_GRef).
- Implementing the task to display the set speed on a desktop PC (t_DSpd).

The source files submitted by each team were compared and analysed with the subsequent submissions using Araxis Merge for Windows. Through the analysis, each file was associated (if relevant) with the development phase described. The synchronisation involved grouping source files with respect to their associated patterns in order to analyse the effort involved and the quality of the source code produced.

8.3.3 Mean results

Table 8-2 illustrates the mean results obtained in Case Study 8.

Table 8-2 The mean results for Case Study 8.

	Mean of groups with tool (A-I, A-II & B-II)	Mean of groups with no tool (B-I, C-II & D-II)	Comparison of tool with no tool
Total effort for the entire project (in minutes)	280	360	22% less
Total effort to implement the patterns (in minutes)	149	240	38% less
Total changes made to the patterns (in LOC)	32	79	59% less
Total coupling level for all the patterns implemented	3	8	63% less

The results suggest that for all cases, the groups that used the tool required less effort and produced software of higher quality. However, these results do not tell the whole story. A breakdown of the individual results is presented in the following sections.

8.3.4 Results of Case Study 8-I

In Case Study 8-1, neither of the groups (A-I nor B-I) completed all of the required system features within the allocated time. As a consequence, the analysis was carried out up to the point at which the groups decided to begin testing their source code on the testbed. This point of time was chosen as an indication of the confidence level of each group with their software development stage, before moving on to the hardware-in-the-loop testing. Table 8-3 shows the results obtained up to this point (after the synchronisation process) for each development phase. From this, Table 8-4 was constructed to illustrate the results for the pattern implementations only.

Table 8-3 The synchronised results of the effort and changes made for Case Study 8-I.

Development Phases	Effort (in minutes)		Changes (in LOC)	
	A-I	B-I	A-I	B-I
p_Sch	42	67.5	11	15
p_Port	16	30	1	16
p_LED	10	16	0	3
p_Counter	26	41	8	17
p_PID	45	53.5	11	22
p_ADC	10	11	0	5
t_CThr	62	37.5	17	29
t_GRef	22	7.5	14	0
t_CSpd	37	66	27	26
Total	270	330	89	133

The results in Table 8-4 show that the total time required to implement all the patterns in the CCS was less for the group that used the tool. Similarly, the total number of changes made to all the patterns was less for the group that used PTTES Builder. By comparing the level of coupling in all the patterns implemented, Table 8-4 shows that the group that used the tool created source code with better modularity.

Table 8-4 The results (in total) for effort, reliability and modularity of the patterns in Case Study 8-I.

	Grp A-I (Tool)	Grp B-I (No Tool)
Total effort to implement the patterns (in minutes)	149	219
Total changes made to the patterns (in LOC)	31	78
Total coupling level for all the patterns implemented	0	5

To determine whether the tool did indeed reduce the effort required in implementing the patterns, each pattern was individually analysed. Figure 8-4 shows that individual patterns were implemented with less effort by the group that used PTTES Builder. This suggests that the tool was effective in reducing the effort involved in the development of the embedded system.

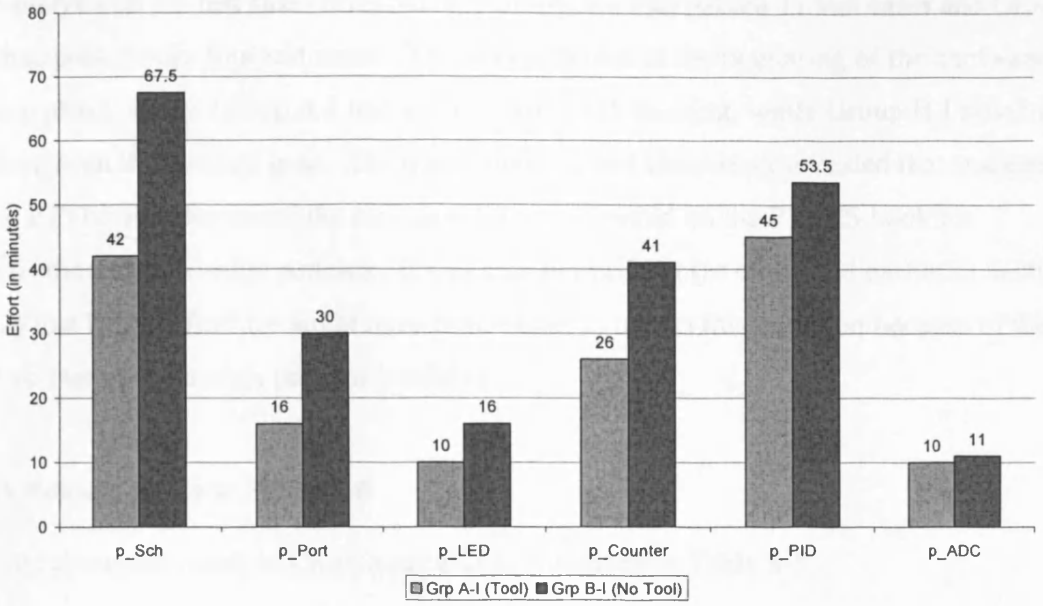

Figure 8-4 Effort involved in implementing the related patterns in Case Study 8-I.

In comparing the changes made to the individual patterns, Figure 8-5 shows that the patterns implemented by the group that used PTTES Builder required fewer changes to the generated code. This suggests that the group using PTTES Builder had a more stable pattern implementation process.

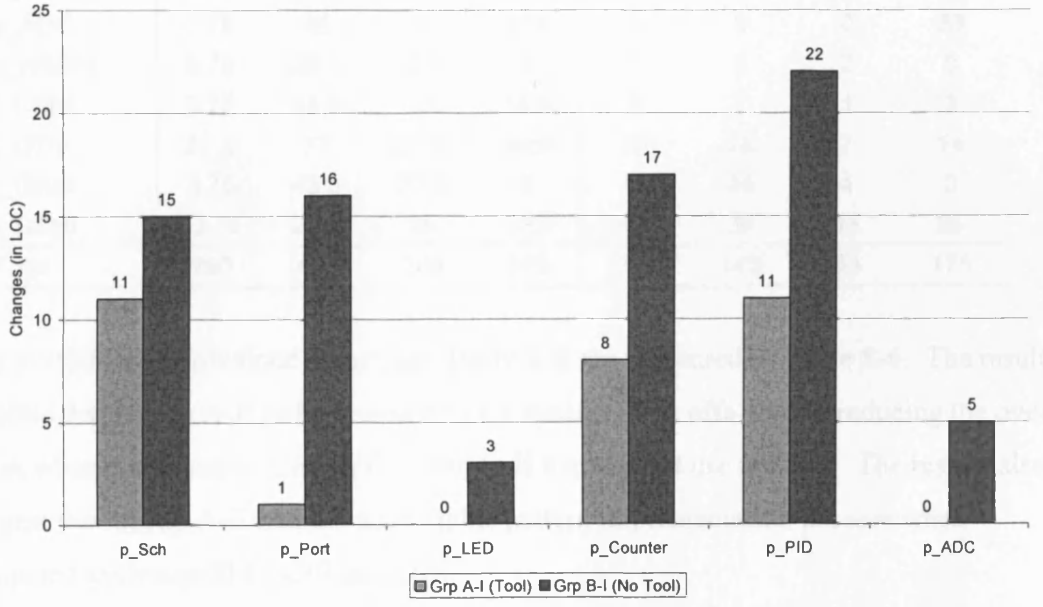

Figure 8-5 The number of changes made to the individual patterns in Case Study 8-I.

The analysis on the test cases revealed that Group A-I had passed 11 test cases and Group B-I had passed only four test cases. This was reflected at the beginning of the hardware testing phase, where Group A-I had an on-board LED flashing, while Group B-I failed to achieve even this modest goal. The questionnaires and interviews revealed that students using PTTES Builder found the tool useful, but still relied on the PTTES book for clarification of the design patterns. It was also pointed out (as explained earlier in Section 8.2.2) that PTTES Builder might have been easier to use on this occasion because of the limited number of design patterns available.

8.3.5 Results of Case Study 8-II

The synchronised result in Case Study 8-II is illustrated in Table 8-5.

Table 8-5 The synchronised results for effort and changes made for Case Study 8-II.

Development Phases	Effort (in minutes)				Changes (in LOC)			
	A-II	B-II	C-II	D-II	A-II	B-II	C-II	D-II
p_Sch	23.75	93	52.5	90.5	3	26	21	45
p_Port	18.75	6	15	23.5	3	0	2	3
p_LED	3.75	6	15	10	0	0	0	0
p_Counter	3.75	6	5	24.5	0	0	3	11
p_PID	8.75	42	27.5	69	4	18	5	15
p_ADC	3.75	46	5	134	0	9	0	53
p_RS232	8.75	28.5	25	5	1	2	2	0
t_CSpd	3.75	43.5	15	55.5	0	7	1	9
t_CThr	37.5	77	27.5	60.5	18	28	7	14
t_GRef	3.75	43.5	27.5	0	0	24	4	0
t_DSpd	33.75	28.5	25	37.5	45	29	18	25
Total	150	420	240	510	74	143	63	175

The overall results obtained from Case Study 8-II are presented in Table 8-6. The results indicate that Group A-II (which used PTTES Builder) was effective in reducing the overall effort when compared to Groups C-II and D-II that did not use the tool. The results also suggest that Group A-II had the most stable pattern implementation process when compared to Groups B-II, C-II and D-II.

However, Group B-II only showed overall improvements in the effort and software quality when compared to Group D-II (see Table 8-6). This is surprising since Group C-II (without tool) took less effort and made fewer changes to the patterns when compared to

Group B-II that did use the tool. Nevertheless, Group B-II produced source code with better modularity compared to C-II and D-II.

These apparently contradictory results will be discussed again in Section 8.4.

Table 8-6 The results (in total) for effort, reliability and modularity of the patterns in Case Study 8-II. Table adapted from Pont *et al.* (submitted), Table 3.

	Grp A-II (Tool)	Grp B-II (Tool)	Grp C-II (No Tool)	Grp D-II (No Tool)
Total effort for the entire project (in minutes)	150	420	240	510
Total effort to implement the patterns (in minutes)	71	228	145	357
Total changes made to the patterns (in LOC)	11	55	33	127
Total coupling level for all the patterns implemented	5	4	11	9

By comparing the development of the individual patterns for the different groups, the results (see Figure 8-6) indicates that Group A-II – in general – took less effort compared to the other groups. This was not the case for Group B-II where the effort involved for some of the individual patterns was higher when compared to groups that did not use PTTES Builder.

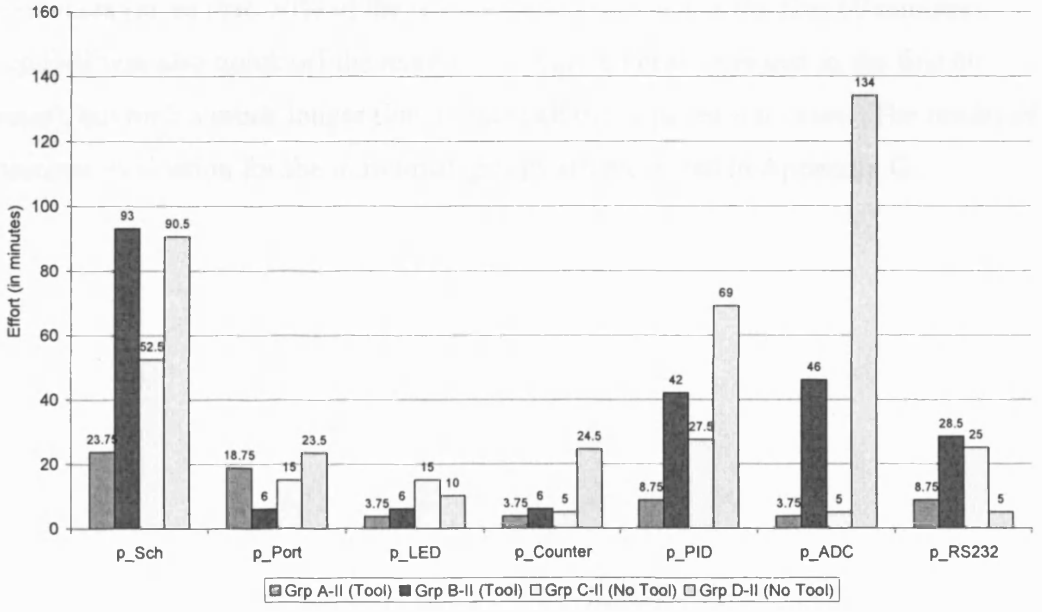

Figure 8-6 The effort involved in implementing the related patterns in Case Study 8-II.

The result in Figure 8-7 illustrates the changes made to individual pattern throughout the development process. The results indicate that Group A-II made minimal changes to the implemented patterns. All other groups had made some significant changes to the patterns.

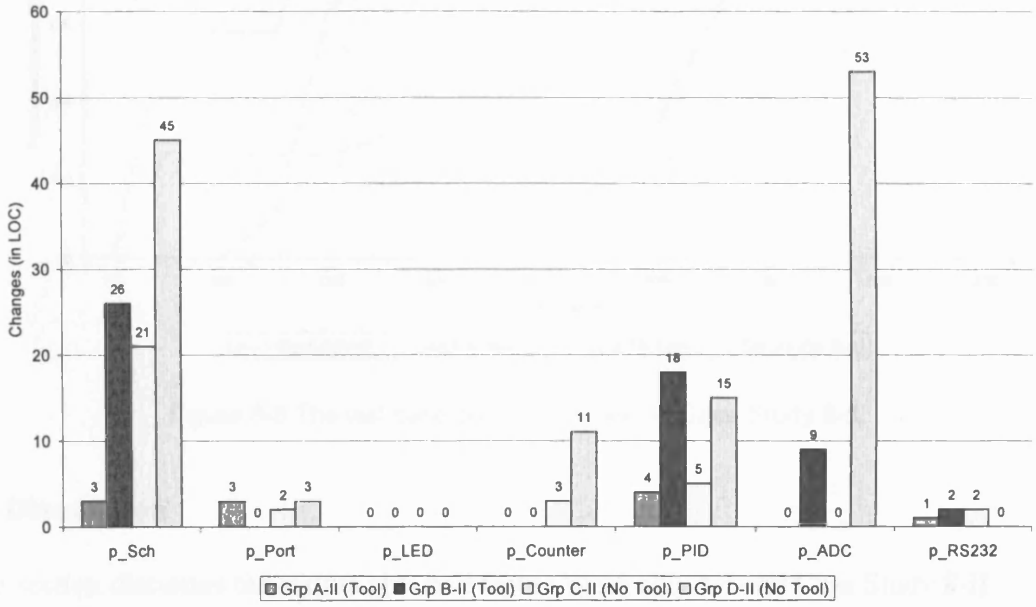

Figure 8-7 The number of changes made to the individual patterns in Case Study 8-II.

By observing the rate at which the groups met all the requirements (by checking it against the test cases: Figure 8-8), it was observed that Group A-II was the first to comply with the test cases (more than 50% of the requirements were met in the first 60 minutes). Group B-II was also quick off the mark (25% requirements were met in the first 60 minutes), but took a much longer time to meet all the required test cases. The results of the testcase evaluation for the individual groups are presented in Appendix G.

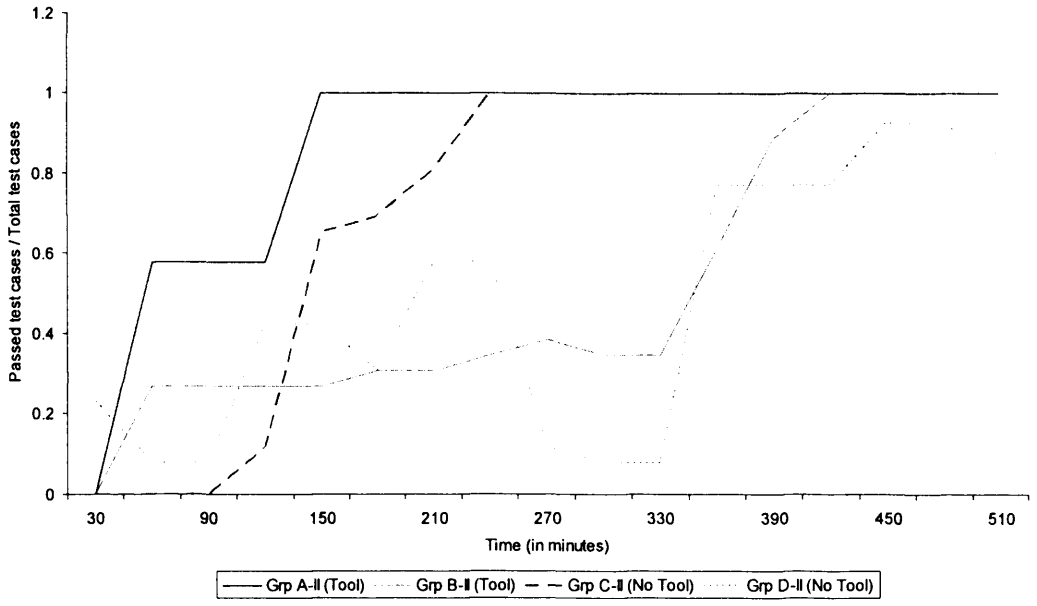

Figure 8-8 The test case compliance ratio in Case Study 8-II.

8.4 Discussion

This section discusses the results obtained from Case Study 8-I and Case Study 8-II.

The results from Case Study 8-I shows that the groups that used the PTTES Builder took less effort to implement the patterns. The results also suggest that the use of the PTTES Builder has led to better source code modularity and a more stable pattern implementation process. In Case Study 8-II of the study, out of the two groups that used the PTTES Builder (Groups A-II and B-II), only Group A-II showed an improvement in the development effort, modularity, source code stability and rapidly producing functionally correct source code. However, the results from Group B-II seem to be in contradictory to these findings. This may suggest that some other factor had influenced the development process of Group B-II.

Upon further investigation (through the interviews, questionnaire and code analysis), it became clear that Group B-II had only very limited confidence in the tool. As a consequence, the subject had attempted to make major changes to the scheduler and pattern tasks generated by the tool in the Keil IDE. Specifically, the subject had changed the declaration of the tasks and modified the task dispatch function. This, in turn, introduced many software bugs and caused a significant amount of delay in the development process. This finding is in line with our assumption in Section 8.2.3, where

the more changes made to the PIEs, the more likely it is to introduce bugs into the system. Overall, this result suggests that tool support will only be effective if users have confidence in the tools (something which is perhaps more likely with a commercial product than in a trial such as that described here).

It was also noted that the trend of results observed from Case Study 8-I and Case Study 8-II (Group A-II) were very similar. This may imply that the possibility of bias in Case Study 8-I was not a significant factor in the results obtained.

By plotting the results of the changes made and effort taken on the same graph, the results in Figure 8-9 illustrates that the trend of the development effort was very similar to the trend of the software stability. The result of each group's development process also showed a similar trend for the development effort and changes made (see Appendix E).

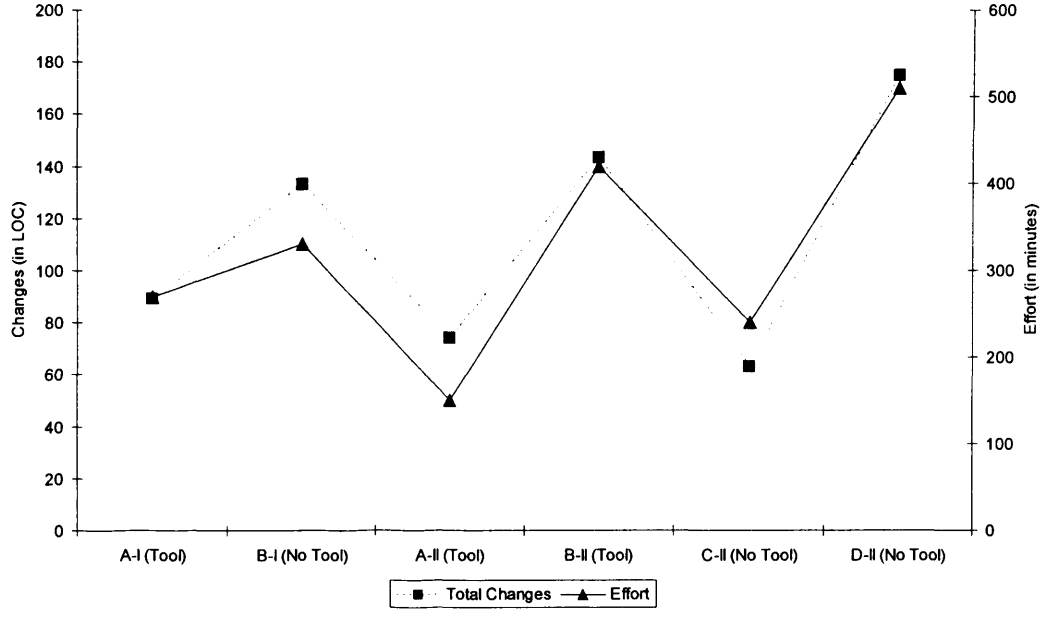

Figure 8-9 Trend of changes made and development effort for all the groups in Case Study 8.

Overall, the results suggest that implementing the patterns using PTTES Builder did reduce the effort: however, some patterns showed more reduction of effort than the others. To investigate this further, McCabe's Cyclomatic Complexity (McCabe, 1976) was used to determine the complexity level of the different PIEs. The PIE with the highest level of complexity was associated with the Co-operative Scheduler. By analysing the results in

Figure 8-6 and Figure 8-7, it was revealed that PTTES Builder contributed significantly to reducing the implementation effort of Co-operative Scheduler and at the same time improving the quality of the resulting code. However, for less complex patterns (such as Port Wrapper and Heartbeat LED), the contribution of the tool was not as significant. This may imply that PTTES Builder is much more effective in implementing patterns with a higher level of complexity. Further studies would have to be carried out to confirm this.

8.5 Conclusions

In this chapter, the primary goal was to investigate the extent to which the SGM can be applied to a wider range of studies. To do this, the SGM was employed in an empirical study to assess the efficacy of a tool that automates the implementation of reliable software patterns for embedded systems.

In a two-phase empirical study, the effectiveness of tool-based pattern development technique was compared with an equivalent "manual" approach. The results obtained suggested that – in almost all cases – the use of the tool reduced the development effort. The exception to this general rule was in one case where the subject opted to re-write the code generated by the tool (probably because of lack of confidence in the code-generation approach). In this circumstance it was noted that such a lack of confidence may be more likely in a laboratory setting (with an experimental tool) than in a commercial setting.

Three other observations were made from these studies: (1) there was some evidence that the use of the tool was likely to lead to improved code quality, (2) the contribution of the tool was most significant when implementing patterns with a high level of complexity, and (3) the development effort involved showed a very similar trend to the number of changes made to the source code. However, further studies are required to investigate these observations in more detail.

Overall, the results discussed in this chapter suggest that the SGM has the potential to be more widely applied in other research areas to rapidly assess new technologies in a cost-effective way.

9. Discussion and conclusions

9.1 Introduction

In this chapter, the conclusions from this research project are presented. Some suggestions for future work in this area are made.

9.2 Overview of the work conducted

The work described in this thesis began by exploring ways in which a software simulator could be used to support the development of reliable X-by-Wire systems. Specifically, the initial aim of the research was to develop a simulation tool that could support the development of X-by-Wire applications by allowing the comparison of different design options early in the product life cycle.

As outlined in Chapter 2, the work began by considering some of the available simulation tools. TrueTime was provisionally chosen as a starting point to explore the use of simulation in the implementation of X-by-Wire systems. To confirm that TrueTime was appropriate for use in the present project, two non-trivial case studies were carried (Chapter 3). The results obtained indicated that the TrueTime simulation technique provided an effective way of predicting the performance of X-by-Wire systems. This suggested that – instead of reinventing the wheel – the available TrueTime simulation tool could be used as a simulator for X-by-Wire designs.

Of course, although it was necessary to verify that the TrueTime simulator works, this condition alone was not enough to demonstrate the full potential of the tool. During the course of this research, it became apparent that appropriate (empirical) software-engineering techniques were required to carry out investigations to evaluate the effectiveness of the "simulation first" approach. The Small Group Methodology" (SGM) was then developed as a cost-effective means to rapidly assess various development methodologies and implementation techniques for embedded systems.

9.3 Simulation in practice

This section presents an evaluation of the "simulation first" approach. The inferences made from the results and the validity of the claims are discussed here.

9.3.1 The efficacy of a "simulation first" approach

In the research project described in this document, four aspects of the efficacy of a "simulation first" approach was evaluated.

First, studies were carried out to determine whether the TrueTime simulator could predict the behaviour of a range of X-by-Wire systems correctly (see Chapter 3). The results suggest that although the simulator is imperfect, it can still be used to predict the behaviour of a range of embedded implementation options. It was also noted that the simulator was more effective in simulating time-triggered designs, as opposed to event-triggered designs.

Second, an empirical study was presented (in Chapter 5) to assess the contribution of a "simulation first" approach towards the development effort. The results suggest that simulation can assist in lowering the development effort required to subsequently implement the system on the hardware. The results also suggest that the overall effort can be reduced if the simulator is used effectively.

Third, another case study was presented (in Chapter 6) to determine whether the development effort can be further reduced by modifying the simulation methodology. Here, the results suggest that the TT-Plus approach – that employed a high-level design-led code-generation technique – can further reduce the effort involved, especially in the simulation phase. In particular, the results suggest that the overall development effort can be further reduced if the effort required to develop the relevant simulation models is itself reduced.

Finally, the study presented in Chapter 7 assessed the impact of a "simulation first" approach on software quality. Here, indirect software quality measures were used. The results suggest that use of simulation may indeed improve the quality of software for

embedded systems. The results also suggest that the reliability of the simulation models is better when using a design-led code-generation approach.

Overall, the results suggest that using a "simulation first" approach can be an effective way of designing and implementing X-by-Wire systems.

9.3.2 Validity of the results

Although evidence have been provided to suggest that a "simulation first" approach is effective, the extent to which the claims can be generalised must be considered.

First, in the various simulation case studies, TrueTime has been used to simulate 16-bit C167 microcontrollers (see Case Study 3A and Case Study 5) and 32-bit ARM microcontrollers (see Case Study 3B and Case Study 6). This may suggest that TrueTime could be effective as a general purpose simulator for a wide range of embedded microcontrollers.

However, the case studies have only explored the use of the TrueTime simulator. Therefore, the results and discussions cannot be generalised to other simulation tools. This is because the methodologies for the various tools are different from one another. For example, TrueTime employs coding and the creation of block diagrams, whereas tools like TimesTool and AIDA use different approaches to simulate embedded systems (such as message sequence charts, timed automata and dataflow diagrams). Nonetheless, lessons learnt from the TrueTime simulator could be useful in the development and enhancements of other simulators. For instance, tools such as AIDA, might benefit from employing a "source-code development approach" as there is evidence that this can contribute to the reduction of effort in the implementation stage.

In addition to the fact that only one simulator was used, the work presented in this thesis only employed two testbeds: an automotive cruise-control system and an inverted pendulum. In both these studies, evidence was provided to show ways that simulation could contribute to the development of embedded systems. However, due to time and resource constraints, it was impossible to try a wider range of testbeds. As such, it cannot yet be claimed that the results will hold for all cases. For example, in a very large-scale

embedded application that involves various individuals, the conventional "simulation first" approach may not be effective. Instead, it could be the case that the simulation, implementation and testing processes will have to be staggered throughout the various development phases for the various sub-modules.

9.4 Evaluation of the SGM

One "by-product" of this research was the SGM. It was through this approach, that the findings in Chapter 5 to Chapter 8 of this thesis were made possible. The effectiveness of the SGM is discussed in this section.

9.4.1 Is the SGM useful?

One of the key characteristics of the SGM is that it employs small number of student subjects and matches them up in terms of their marks in previous university modules. Through this approach, empirical evidence can be obtained rapidly and in a cost-effective way.

Although the number of test subjects used is very small, the technique used is effective in producing large amounts of useful data. These include the investigation of software development effort and software quality when employing various development methodologies for embedded systems. Moreover, the use of SGM has successfully raised several questions about the various aspects of the development methodology, and assisted in generating hypotheses.

Keeping in mind that most of the case studies in this thesis employed the SGM to explore the efficacy of a "simulation first" approach, it should not be assumed that the SGM is confined to only simulation related studies. To investigate if the SGM can be applied more widely, a different study (Case Study 8) that involved design patterns and code generation for embedded systems was employed. This investigation successfully revealed various useful contributions of design patterns and code-generation approach. This suggests that SGM has the potential to be more widely applied in other studies to rapidly evaluate development methodologies and implementation techniques for embedded systems in a cost-effective way.

Overall, the SGM – which includes guidelines on how to carry out empirical experiments, – is effective in providing insight on the various development approaches for embedded systems. This may allow industrial firms to rapidly obtain some empirical evidence on the efficacy of a particular technology.

9.4.2 Using qualitative data to "fill the gaps"

In contrast with empirical studies using large sample sets, the impact caused by individual characteristics must be considered in studies using the SGM.

In the studies considered in this document, individual differences were taken into account through the use of qualitative data analysis, using progress observation forms, questionnaires and interviews. These techniques were shown to be effective in eliciting information that was not obvious through source code analysis, and provided a means of understanding the overall outcome of the quantitative measurements.

In this way, the following anomalies were documented:

1) Coding strategy: In Chapter 5, it was noted that the subject in Group C chose not to port the source code from the simulation environment (TrueTime/MATLAB) to the implementation environment (Keil). Instead, the student wrote the code from scratch at the implementation environment that resulted in functional errors in the implementation stage. The other group on the other hand had ported the source code over and made syntactical changes. This resulted in them getting their system working much quicker.
2) Lack of confidence: In Chapter 8, it was documented that one of the groups (B-II) had very little confidence in the tool given to implement the embedded system. This led the test subject to make major changes to the source code and introduced software bugs into the system. This caused severe delay in the implementation process.

9.4.3 The "chicken or egg" problem

The work presented in this thesis has evaluated the "simulation first" approach and the SGM through various trials. This however raises a causality problem, which is sometimes referred to as "which came first, the chicken or the egg".

In particular, the SGM was initially used to assess the "simulation first" approach, when the methodology itself had not been verified beforehand. Here, the inconsistency in the research is apparent because neither the SGM, nor the "simulation first" approach, had previously been verified. Instead, the approach employed in this research was to assume that the SGM was a valid technique, and subsequent studies were carried out to evaluate the effectiveness of the "simulation first" approach.

In this case, it could be argued that an unverified empirical approach used to evaluate an unverified development technique may introduce more uncertainties in the results. However, the results obtained from the various empirical evaluations are sensible. For example, in Chapter 5, it was shown that subjects that did not use the simulator efficiently, made more mistakes that eventually led to an increase in the overall effort. The similar sort of logical causality behaviour had also been recorded for all the other case studies. This suggests – by means of the results obtained – that the SGM had been indirectly verified through the various case studies.

Therefore, in light of this discussion, the work presented in this thesis has verified the SGM, in addition to concurrently providing evidence to demonstrate the efficacy of various development approaches for embedded systems. It is argued that the assumptions made in doing so were reasonable, and this was confirmed based on the results that were obtained. Overall, although the underlying "chicken or egg" problem had not been resolved, it is believed that this issue did not have a huge bearing on the outcome of this research.

9.5 Other "by products" of the research project

During the course of this research project, several other technical contributions were made. These contributions – although not directly related to the core of the work described in this thesis – resulted in several publications (see page viii).

The first "by product" of the research project was the development of new variants of the Shared-Clock CAN algorithm (Ayavoo *et al.*, 2005b; Ayavoo *et al.*, accepted). This work was carried out to address some of the limitations in the existing shared-clock algorithms.

Specifically, the two new algorithms that were developed (referred to as TTC-SC3 and TTC-SC4) has inspired the work of other researchers (see Short and Pont, 2006).

The second "by product" was the development of a non-trivial cruise-control testbed for a passenger car (Ayavoo *et al.*, 2005c). This work was carried out to obtain a low-cost testbed that could be used to assess different implementation techniques for embedded systems. The testbed has also been found to be useful by other researchers (see Mwelwa *et al.*, 2005; Vidler and Pont, 2006; Kurian and Pont, 2007).

9.6 Future work

As this thesis draws to a close, some suggestions for future work in this important area are made.

9.6.1 Improvements to the SGM

The version of the SGM described in this thesis has been effective in evaluating the development process of embedded systems. On hindsight however, there is still room for further improvements in the evaluation process.

One way to improve the SGM would be to employ a "blind" approach to analyse the results (Kitchenham *et al.*, 2002). In this case, the individual who analyses the results from the study is kept unaware as to which set of results have been subjected to a treatment. Such an approach can reduce the bias in the analyses. Please note that using this approach could incur additional cost in hiring a third party to analyse the results.

In addition, to improve the quality of the results in future studies, a video recorder could be used to digitally record the events in the experiments. The data collection technique may also be automated to reduce the level of intrusion to the study.

In the current version of the SGM, the case study was prepared followed by the experiment and data collection. The test subjects were then given questionnaires to fill and a short interview was held at the end of the experiment. Finally, the source code collected throughout the experiment was analysed. One drawback to this approach is in

the event where the subsequent source code analysis reveals an anomaly that was not picked up in the questionnaire, interview or progress observation forms. In this case, understanding the results will be difficult and may rely on the researcher's intuition. To avoid this, the SGM can be modified to incorporate two interview sessions, one before the source code analysis and another interview after the analysis (see Figure 9-1). This way, the researcher has a chance to cross-examine the test subjects for the second time to confirm the findings.

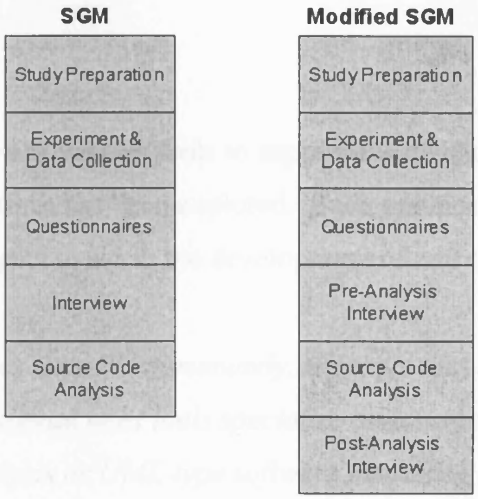

Figure 9-1 The modified SGM that incorporates pre- and post-analysis interviews.

9.6.2 Further studies on simulation

The work presented in this thesis on software simulation for X-by-Wire systems is only the tip of the iceberg. Indeed, there is much more to learn from the contributions that a "simulation first" approach can make towards the development of X-by-Wire systems.

For instance, although this research has suggested that simulation can potentially improve software quality, its impact upon other quality metrics like error density and field stability have not been explored. Moreover, the work on software quality described in this thesis has only looked at the software development for a final solution, and not an incremental system development that involves software maintenance. The contributions of a "simulation first" approach for such system may be different.

Besides this, the work described in this thesis has evaluated the "simulation first" approach when it is used only at the coding and testing phases. However, simulation

could also be used in different ways like the "A" and "V" development process (see Nossal and Lang, 2002). In such cases, the impact of the simulation on the development effort would be different.

Another area for future research is the comparison of the various simulation tools that was described in Section 2.3.3. In particular, the various properties of the different simulators that contribute to the development effort and software quality can be further explored.

9.6.3 Beyond simulation

In this thesis, the utilisation of various tools to support the development of a range of embedded control applications has been explored. Such practice has been identified as becoming an important aspect to aid in the development of embedded systems.

Hendriksson and colleagues note: "*Unfortunately, the tools that allow a co-design approach are quite few. Instead most tools specialize on a single domain, e.g., control design, schedulability analysis or UML-type software modelling and code generation*" (Henriksson *et al.*, 2005, p. 51). This implies that although there are various tools available to assist in the development process, many of them are too specific and limited to a single domain. This is particularly a problem in the development of embedded control system, where there is a close integration of control systems, hardware design, software architectures, network protocols and schedulability analysis.

The work presented in this thesis has touched on the use of simulation techniques, code generation and design patterns. Specifically, simulation was used at the design, verification and implementation phase, while code generations and design patterns were used to implement embedded systems. Future research should consider employing a more collective approach of tool support by combining some of these techniques to assist in the overall development of reliable X-by-Wire systems.

For instance, future research could investigate the extent to which autocode generation from design patterns can be used with TrueTime to simulate an embedded system. Similarly, the combination of various CASE tools to design, verify, compile and simulate an embedded system can be considered for future research projects.

9.7 Conclusions

Overall, the work described in this thesis has made three major contributions. First, evidence has been successfully provided to demonstrate the effectiveness of a "simulation first" approach when it is used in practice to design and implement embedded systems. Second, the research has described and demonstrated a set of techniques (called SGM) to rapidly carry out empirical evaluations for embedded systems at low cost. Finally, evidence has been presented to suggest that the SGM can be applied more widely to other studies. This thesis, in addressing some of the initial issues that were posted in the outset, has prompted many more interesting questions. This will hopefully provide the necessary inspiration for further research in this critical area.

References

Ahlmark, M. (2000). "Local Interconnect Network (LIN) – packaging and scheduling". M.Sc. Thesis. Department of Computer Engineering, Malärdalen University, Västerås, Sweden.

Altera (2005). "Quartus II simulation with VHDL designs". Altera Corporation,

Amasaki, S., Y. Takagi, O. Mizuno and T. Kikuno (2005). "Constructing a Bayesian Belief Network to predict final quality in embedded system development." IEICE Transactions on Information and Systems E88-D(6): 1134-1141.

Amnell, T., E. Fersman, L. Mokrushin, P. Pettersson and W. Yi (2002). "Times: A tool for modelling and implementation of embedded systems". International Conference on Tools and Algorithms for Construction and Analysis of Systems, TACAS2002, Grenoble, France.

Amnell, T., E. Fersman, L. Mokrushin, P. Pettersson and W. Yi (2003). "TIMES: A tool for schedulability analysis and code generation of real-time systems". Proceedings of the 1st International Workshop on Formal Modeling and Analysis of Timed Systems (FORMATS 2003), Marseille, France.

Andel, T. R. and A. Yasinsac (2006). "On the credibility of Manet simulations." IEEE Computer 39(7): 48-54.

Arisholm, E. and D. I. K. Sjöberg (2004). "Evaluating the effect of a delegated versus centralized control style on the maintainability of Object-Oriented software " IEEE Transaction on Software Engineering 30(8): 521-534.

ARM (2001). "ARM7TDMI technical reference manual".

Aström, K. J. and B. Wittenmark (1990). "Computer-controlled systems, theory and design". New Jersey, Prentice-Hall International. [0131727842].

Audsley, N., A. Burns, K. Tindell, M. Richardson and A. Wellings (1993). "Applying new scheduling theory to static priority pre-emptive scheduling." Software Engineering Journal 8(5): 248-292.

Babbage, C. (1888). "The analytical engine". Proceedings of the British Association, Bath.

Ball, S. R. (1996). "Embedded microprocessor systems: real world design". Boston, Newnes. [0750697911].

Bannatyne, R. (2003). "Microcontrollers for automobiles." Micro Control Journal.

Basili, V., S. Asgari, J. Carver, L. Hochstein, J. K. Hollingsworth, F. Shull and M. V. Zelkowitz (2004). "A pilot study to evaluate development effort for high performance computing". CS-TR-4588 University of Maryland, Maryland, USA.

Basili, V., L. C. Briand and W. L. Melo (1996). "A validation of Object-Oriented design metrics as quality indicators." IEEE Transaction on Software Engineering 22(10): 751-761.

Basili, V. and R. W. Reiter (1979). "An investigation of human factors in software development." IEEE Computer 12(12): 21-38.

Basili, V., F. Shull and F. Lanubile (1999). "Building knowledge through families of experiments." IEEE Transaction on Software Engineering 25(4): 456-473.

Basili, V. and D. Weiss (1984). "A methodology for collecting valid software engineering data." IEEE Transaction on Software Engineering 10(6): 728-738.

Basili, V. R., R. W. Selby and D. H. Hutchens (1986). "Experimentation in software engineering." IEEE Transactions on Software Engineering 12(7): 733-743.

Bassin, K., S. Biyani and P. Santhanam (2002). "Metrics to evaluate vendor-developed software based on test case execution results." IBM Systems 41(13-30).

Bate, I. J. (1998). "Scheduling and timing analysis for safety-critical real-time systems". Ph.D Thesis. Department of Computer Science, University of York, York, England.

Bautista, R. and M. J. Pont (2006). "Is fuzzy logic a practical choice in resource-constrained embedded control systems implemented using general-purpose microcontrollers?" Proceedings of the 9th IEEE International Workshop on Advanced Motion Control, Istanbul, Turkey.

Bautista, R., M. J. Pont and T. Edwards (2005). "Comparing the performance and resource requirements of "PID" and "LQR" algorithms when used in a practical embedded control system: A pilot study". Proceedings of the 2nd UK Embedded Forum 2005, Birmingham, UK.

Becks, K. (2000). "Extreme programming explained". London, Addison-Wesley. [0201616416].

Berry, D. M. and W. F. Tichy (2003). "Comments on "Formal methods application: An empirical tale of software development"." IEEE Transaction on Software Engineering 29(6): 567-571.

Bevan, N. (1999). "Quality in use: Meeting user needs for quality." Journal of Systems and Software 49(1): 89-96.

Bosch, R. G. (1991). "CAN specification version 2.0". Robert Bosch GmbH, Postfach 50, D-7000 Stuttgart 1, Germany,

Bretz, E. A. (2001). "By-Wire cars turn the corner." IEEE Spectrum 38(4): 68-73.

Briand, L. C., J. Wüst, J. Daly and D. V. Porter (2000). "Exploring the relationships between design measures and software quality in Object-Oriented systems." Journal of Systems and Software 51(3): 245-274.

Brière, D., C. Favre and P. Traverse (1995). "A family of fault-tolerant systems: Electrical flight controls, from Airbus A320/330/340 to future military transport aircraft." Microprocessors and Microsystems 19(2): 75-82.

Brooks, R. E. (1980). "Studying programmer behaviour experimentally: The problems of proper methodology." Communications of the ACM 23(4): 207-213.

Carson II, J. S. (2005). "Introduction to modeling and simulation". Proceedings of the 2005 Winter Simulation Conference, Orlando, USA.

Castelpietra, P., Y. Q. Song, F. S. Lion and M. Attia (2001). "CAROSSE-Perf: A modular approach for simulation of in-vehicle embedded architectures". 15th European Simulation Multiconference - ESM'2001, Prague, Czech Republic.

Castelpietra, P., Y. Q. Song, F. S. Lion and M. Attia (2002). "Analysis and simulation methods for performance evaluation of a multiple networks embedded architecture." IEEE Transactions on Industrial Electronics 49(No. 6).

Cervin, A., D. Henriksson, B. Lincoln, J. Eker and K. Årzén (2003). "How does control timing affect performance? - Analysis and simulation of timing using jitterbug and TrueTime." IEEE Control Systems Journal 23(3): 16-30.

Chen, D. J. and M. Sandfridson (2000). "Introductions to distributed systems for real-time control". TRITA MMK 1998:22 Mechatronics Laboratory, Department of Machine Design, Royal Institute of Technology, Stockholm, Sweden.

Chidamber, S. R. and C. F. Kemerer (1994). "A metrics suite for Object-Oriented design." IEEE Transaction on Software Engineering 20(6): 476-493.

Chouliaras, V. A., J. L. Nunez, D. J. Mulvaney, F. S. Rovati and D. Alfonso (2005). "A multi-standard video accelerator based on a vector architecture." IEEE Transaction on Consumer Electronics 51(1): 160-167.

Ciolkowski, M., O. Laitenberger, S. Vegas and S. Biffl (2003). "Practical experiences in the design and conduct of surveys in empirical software engineering." Lecture Notes in Computer Science (LNCS) 2765/2003: 104-128.

Clarke, P. "Adaptive cruise control takes to the highway". EETimes Online. (Last accessed: 31/07/2006) *http://www.eetimes.com/story/OEG19981020S0007*

Coleman, D., D. Ash, B. Lowther and P. Oman (1994). "Using metrics to evaluate software systems maintainability." IEEE Computer 27(8): 44-49.

Cook, J. E., L. G. Votta and A. L. Wolf (1998). "Cost-effective analysis of in-place software processes." IEEE Transaction on Software Engineering 24(8): 650-663.

Cunningham, W. and K. Becks (1987). "Using pattern languages for Object-Oriented programs". OOPSLA'87 workshop on the Specification and Design for Object-Oriented Programming, Florida, USA.

Dahl, O. J. and K. Ngyaard (1966). "SIMULA - An ALGOL-based simulation language." Communications of the ACM 9(9): 671-678.

Dawson, L. and P. Swatman (1999). "The use of Object-Oriented models in requirement engineering: A field study". Proceedings of the 20th International Conference on Information Systems, Charlotte, North Carolina, USA, Association for Information Systems.

Dorf, D. and R. Bishop (2000). "Modern control systems". New Jersey, Prentice-Hall. [0130306606].

Dutton, K., S. Thompson and B. Barraclough (1997). "The art of control engineering". Harlow, Addison Wesley. [0201175452].

Dyba, T., B. A. Kitchenham and M. Jørgensen (2005). "Evidence-based software engineering for practitioners." IEEE Software 22(1): 58-65.

Edwards, T., M. J. Pont, P. Scotson and S. Crumpler (2004). "A testbed for evaluating and comparing designs for embedded control system". Proceedings of the 1st UK Embedded Forum 2004, Birmingham, UK.

Eker, J. and A. Cervin (1999). "A Matlab toolbox for real-time and control systems co-design". Proceedings of the 6th International Conference on Real-Time Computing Systems and Applications.

Ekiz, H., A. Kutlu, M. D. Baba and E. T. Powner (1996). "Design and implementation of a CAN/Ethernet bridge". 3rd International CAN Conference, Paris, France.

El-khoury, J. and M. Törngren (2001). "Towards a toolset for architectural design of distributed real-time control systems". IEEE Real-Time Symposium, London, England, IEEE.

Ellims, M., S. Parker and J. Zurlo (2002). "Design and analysis of a robust real-time engine control network." IEEE Micro 22(4): 20-27.

Endres, A. and H. D. Rombach (2003). "A handbook of software and systems engineering - Empirical observations, laws and theories". Harlow, Pearson Addison Wesley. [0321154207].

Engblom, J., G. Girard and B. Werner (2006). "Testing embedded software using simulated hardware". Proceedings of Embedded Real-Time Software (ERTS 2006), Toulouse, France.

Fang, J. and M. J. Pont (2006). "Exploring the links between different software architecture and PID parameters in embedded control systems". Proceedings of the 6th UKACC International Control Conference, Glasgow, Scotland.

Faraday, M. (2004). "Experimental researches in electricity", Dover Publications. [0486435059].

Farsi, M. and M. Barbosa (2000). "CANopen implementations: Applications to industrial networks". Exeter, Research Studies Press Ltd. [0863802478].

Fenton, N., S. L. Pfleeger and R. L. Glass (1994). "Science and substance: A challenge to software engineers." IEEE Software 11(4): 86-95.

Fitter, M. J. and P. J. Cruickshank (1983). "Doctors using computers: A case study". Designing for human-computer communication. M. E. Sime and M. J. Coombs. London, Academic Press: 239-260. [012643820x].

Florijn, G., M. Meijers and P. Winsen (1997). "Tool support for Object-Oriented patterns". 11th European Conference on Object-Oriented Programming, Jyväskylä, Finland.

Fredriksson, L. B. (1994). "Controller Area Networks and the protocol CAN for machine control systems." Mechatronics 4(2): 159-192.

Fredriksson, L. B. (2002). "CAN for critical embedded automotive networks." IEEE Micro 22(4): 28-35.

Gamma, E., R. Helm, R. Johnson and J. Vlissides (1995). "Design patterns: Elements of reusable Object-Oriented software". Reading, Addison-Wesley. [0201633612].

GAO (1992). "Patriot missile software problem". GAO/IMTEC-92-26 General Accounting Office, Information Management and Technology Division, Washington, USA.

Gergelait, M. and E. Nett (2002). "Scheduling transient overload with the TAFT Scheduler". Presented at GI/ITG specialised group of operating systems, Berlin.

Germain, E. and P. N. Robillard (2005). "Engineering-based processes and Agile methodologies for software development: A comparative case study." Journal of Systems and Software 75(1-2): 17-27.

Graziani, P. L. (2002). "Keynote speech: Addressing the challenges of aerospace and defense initiative using software analysis tools". Proceedings of the 2002 Winter Simulation Conference, San Diego, USA.

Greenwell, W. S. (2003). "Learning from accidents and incidents involving safety-critical software systems". M.Sc Thesis. Department of Computer Science, School of Engineering, University of Virginia, Virginia, USA.

Grimstad, S., M. Jørgensen and K. Moløkken-Østvold (2006). "Software effort estimation terminology: The Tower of Babel." Information and Software Technology 48(4): 302-310.

Hartwich, F., B. Muller, T. Fuhrer, R. Hugel and R. B. GmbH (2000). "CAN networks with time-triggered communication". 7th international CAN Conference.

Hartwich, F., B. Muller, T. Fuhrer, R. Hugel and R. B. GmbH (2002). "Timing in the TTCAN network". Proceedings 8th International CAN Conference.

Heintz, R. P. (1990). "Vehicle speed control device". U. S. Patent. United States, The Deaccelerator Corporation. 4,947,950.

Henriksson, D., O. Redell, J. El-khoury, M. Törngren and K. Årzén (2005). "Tools for real-time control systems co-design - A survey". ISRN LUTFD2/TFRT--7610--SE Department of Automatic Control, Lund Institute of Technology, Lund, Sweden.

Henry, J., S. Henry, D. Kafura and L. Matheson (1994). "Improving software maintenance at Martin Marietta." IEEE Software 9(4): 67-75.

Henry, S. M. and K. T. Stevens (1999). "Using Belbin's leadership role to improve team effectiveness: An empirical investigation." Journal of Systems and Software 44(3): 241-250.

Henzinger, T. A., C. M. Kirsch, M. A. A. Sanvido and W. Pree (2003). "From control models to real-time code using Giotto." IEEE Control Systems Journal 23(1): 50-64.

Hilmer, H., H. D. Kochs and E. Dittmar (1998). "A CAN-based architecture for highly reliable communication systems". 5th International CAN Conference, San Jose, USA.

Holcombe, M., M. Gheorghe and F. Marcias (2001). "Teaching XP for real: Some initial observations and plans". Proceedings XP 2001, Sardinia, Italy.

Holt, R. W., D. A. Boehm-Davis and A. C. Schultz (1987). "Mental representations of programs for students and software programmers". Empirical Studies of Programmers: Second Workshop, Ablex Publishing Corp.

Höst, M., B. Regnell and C. Wohlin (2000). "Using students as subjects - A comparative case study of students and professionals in lead-time impact assessment." Empirical Software Engineering 5(3): 201-214.

Huang, S. J. and N. H. Chiu (2006). "Optimization of Analogy Weights by Genetic Algorithm for Software Effort Estimation" Information and Software Technology 48(11): 1034-1045.

IAEA (2001). "Investigation of an accidental exposure of radiotherapy patients in Panama - Report of a team of experts". 92-0-101701-4 International Atomic Energy Agency, Vienna, Austria.

ICE "Engineering Heritage". Institute of Civil Engineers. (Last accessed: 26/07/2006) *http://www.ice.org.uk/knowledge/library_heritage.asp*

Infineon (2004). "Connecting C166 and C500 microcontroller to CAN". Infineon Technologies,

Isermann, R., R. Schwarz and S. Stölzl (2002). "Fault-tolerant Drive-by-Wire systems." IEEE Control Systems Journal 22(5): 64-81.

Jones, D. W. "FAQ: What is a PDP?" (Last accessed: 26/07/2006) *http://www.faqs.org/faqs/dec-faq/pdp8/section-1.html*

Jørgensen, M. (1999). "Software quality measurement." Advances in Engineering Software 30(12): 907-912.

Jørgensen, M. and D. I. K. Sjøberg (2001). "Impact of effort estimates on software project work." Information and Software Technology 43(15): 939-948.

Kan, S. H., V. Basili and L. N. Shapiro (1994). "Software quality: An overview from the total quality management perspective." IBM Systems 33(1): 4-19.

Kandasamy, N., J. P. Hayes and B. T. Murray (2005). "Time-constrained failure diagnosis of Distributed embedded systems: Applications to actuator diagnosis." IEEE Transactions on Parallel and Distributed Systems 16(3): 258-270.

Karatza, H. D. (2004). "Modeling and simulation of distributed systems and networks." Simulation Modelling Practice and Theory 12(3-4): 183-185.

Katira, N., L. Williams, E. Wiebe, C. Miller, S. Balik and E. Gehringer (2004). "On understanding compatibility of student pair programmers". ACM Technical Symposium on Computer Science Education, SIGCSE Norfolk, USA.

Kemerer, C. F. and S. Slaughter (1999). "An empirical approach to studying software evolution." IEEE Transactions on Software Engineering 25(4): 493-509.

Kiczales, G., J. Lamping, A. Mendhekar, C. Maeda, C. V. Lopes, J. M. Loingtier and J. Irwin (1997). "Aspect-Oriented programming". European Conference on Object-Oriented Programming (ECOOP), Finland, Springer-Verlag LNCS.

Kitchenham, B. A., S. L. Pfleeger, L. M. Pickard, P. W. Jones, D. C. Hoaglin, K. E. Emam and J. Rosenberg (2002). "Preliminary guidelines for empirical research in software engineering." IEEE Transaction on Software Engineering 28(8): 721-734.

Koopman, P. (2002). "Critical embedded automotive networks." IEEE Micro 22(4): 14-18.

Kopetz, H. (1995). "Automotive electronics - Present state and future prospects". 25th International Symposium on Fault-Tolerant Computing Systems: Special Issue, California, USA, IEEE Computing Society.

Kopetz, H. (1997). "Real-time systems: Design principles for distributed embedded applications". Boston, Kluwer Academic. [0792398947].

Kopetz, H. (1998). "A comparison of CAN and TTP". 15th IFAC Workshop on Distributed Computer Control Systems, Como, Italy.

Kopetz, H. (2001). "A comparison of TTP/C and FlexRay." Research Report 10/2001.

Kureemun, R. (1999). "Minimising the effects of electromagnetic interference in a cruise control system". M. Sc. Dissertation. Department of Engineering, University of Leicester, Leicester, UK.

Kurian, S. and M. J. Pont (2007). "The maintenance and evolution of resource-constrained embedded systems created using design patterns." Journal of Systems and Software 80(1): 32-41.

Kurkowski, S., T. Camp and M. Colagrosso (2005). "MANET simulation studies: the incredibles." ACM SIGMOBILE Mobile Computing and Communications Review (Special Issue on Medium Access and Call Admission Control Algorithms for Next Generation Wireless Networks) 9(4): 50-61.

Lanfear, C., S. Balacco and M. Volckmann (2006). "The 2005 embedded software strategic market intelligence program". Venture Development Corporation, Maryland, USA.

Leen, G. and D. Heffernan (2002). "TTCAN: A new time-triggered Controller Area Network." Microprocessors and Microsystems 26(2): 77-94.

Leen, G., D. Heffernan and A. Dunne (1999). "Digital networks in the automotive vehicle." Computing and Control 10: 257-266.

Lethbridge, T. C., S. E. Sim and J. Singer (2005). "Studying software engineers: Data collection techniques for software field studies." Empirical Software Engineering 10(3): 311-341.

Leveson, N. and C. S. Turner (1993). "An investigation of the Therac-25 accidents." IEEE Computer 26(7): 18-41.

Lewis, D. (2001). "Fundamentals of embedded software; where C and assembly meet". New Jersey, Prentice Hall. [0130615897].

Litterick, M. and M. Brenner (2005). "Utilizing Vera functional coverage in the verification of a protocol engine for the FlexRay automotive communication system". Fourteenth Annual Conference of Synopsys Users Group (SNUG) Europe, Munich, Germany.

Liu, J. W. S. (2000). "Real-time systems". New Jersey, Prentice Hall. [0130996513].

Lonn, H. (1999). "Synchronization and communication results in safety-critical real-time systems". Ph.D Thesis. Department of Computer Engineering, Chalmers University of Technology, Göteborg, Sweden.

Lonn, H. and J. Axelsson (1999). "A comparison of fixed-priority and static cyclic scheduling for distributed automotive control application". The Eleventh Euromicro Conference on Real-Time Systems, York, England.

Maaita, A. and M. J. Pont (2005). "Using 'planned pre-emption' to reduce levels of task jitter in a time-triggered hybrid scheduler". Proceedings of the 2nd UK Embedded Forum, Birmingham, UK.

Maier, R., G. Bauer, G. Stöger and S. Poledna (2002). "Time-triggered architecture: A consistent computing platform." IEEE Micro 22(4): 36-45.

Marcias, F., M. Holcombe and M. Gheorghe (2002). "Empirical experiments with XP". Proceedings of 3rd International Conference on Extreme Programming and Flexible Processes in Software Engineering (XP2002), Sardinia, Italy.

Marcias, F., M. Holcombe and M. Gheorghe (2003). "Design-led & design-less: One experiment and two approaches in 'Extreme Programming and Agile processes in software engineering". Proceedings of the 4th International Conference, XP 2003.

Marsden, B. (2004). "Watt's perfect engine: steam and the age of invention ". New York, Columbia University Press [0231131720].

Marti, P., J. M. Fuertes, G. Fohler and K. Ramamritham (2001). "Jitter compensation for real-time control systems". Proceedings of Real-Time Systems Symposium, London, UK.

Martin, R. A. and L. A. Shafer (1996). "Providing a framework for effective software quality measurement: Making a science of risk assessment". The 6th Annual International Symposium of International Council on Systems Engineering (INCOSE), Systems Engineering: Practices and Tools, Massachusetts, USA.

Marwedel, P. (2003). "Embedded system design". Boston, Kluwer Academic Publishers. [1402076908].

McCabe, T. (1976). "A software complexity measure." IEEE Transactions on Software Engineering 2: 308-320.

Misbahuddin, S. and N. Al-Holou (2003). "Efficient data communication techniques for Controller Area Network (CAN) protocol". ACS/IEEE International Conference on Computer Systems and Applications, Tunis, Tunisia.

Moløkken-Østvold, K. and M. Jørgensen (2003). "A review of surveys on software effort estimation". IEEE International Symposium on Empirical Software Engineering (ISESE 2003), Rome, Italy.

Munson, J. C. and T. M. Khoshgoftaar (1992). "Measuring dynamic program complexity." IEEE Software 9(6): 48-55.

Mwelwa, C., M. J. Pont and D. Ward (2003). "Towards a CASE tool to support the development of reliable embedded systems using design patterns". 1st International Workshop on Quality of Service in Component-Based Software Engineering, Toulouse, France.

Mwelwa, C., M. J. Pont and D. Ward (2004a). "Code generation supported by a pattern-based design methodology". Proceedings of the 1st UK Embedded Forum, Birmingham, UK.

Mwelwa, C., M. J. Pont and D. Ward (2004b). "Patterns to support the development and maintenance of software for reliable embedded systems: A case study". Proceedings of the IEE / ACM Postgraduate Seminar on "System-On-Chip Design, Test and Technology, Loughborough, UK.

Mwelwa, C., M. J. Pont and D. Ward (2005). "Developing reliable embedded systems using a pattern-based code generation tool: A case study". Proceedings of the 2nd UK Embedded Forum, Birmingham, UK.

Mwelwa, C., M. J. Pont and D. Ward (2006). "Rapid software development for reliable embedded systems using a pattern-based code generation tool". Society of Automotive Engineers (SAE) World Congress 2006, Detroit, USA.

Nissanke, N. (1997). "Realtime systems". New Jersey, Prentice Hall. [0136512747].

Nossal, R. and R. Lang (2002). "Model-based system development." IEEE Micro 22(4): 56-63.

NTSB (2000). "Aircraft accident report: Controlled flight into terrain, Korean Air Flight 801, Boeing 747-300, HL7468, Nimitz Hill Guam, August 6 1997". NTSB/AAR-00/01 National Transportation Safety Board,

Oman, P. and S. L. Pfleeger (1997). "Applying software metrics". Los Alamitos, IEEE Computer Society Press. [0818676450].

Ong, H. L. R. (2002). "Techniques intended to reduce the impact of program-flow errors on embedded systems". Ph.D Thesis. Department of Engineering, University of Leicester, Leicester, England.

Ören, T. I., S. K. Numrich, A. M. Uhrmacher, L. F. Wilson and E. Gelenbe (2000). "Agent-directed simulation - Challenges to meet defense and civilian requirements". Proceedings of the 2000 Winter Simulation Conference, Orlando, USA.

Palopoli, L., G. Lipari, L. Abeni, M. D. Abeni, P.Ancilotti and F. Conticelli (2001). "A tool for simulation and fast prototyping of embedded control systems". Proceedings of LCTES01, Snow Bird, Utah, United States, ACM Press.

Palopoli, L., G. Lipari, G. Lamastra, L. Abeni, G. Bolognini and P. Ancilotti (2002). "An Object-Oriented tool for simulating distributed real-time control systems." Software - Practice & Experience 32(9): 907-932.

Pazul, K. (1999). "Controller Area Network (CAN) basics". Microchip Technology Inc.,

Pfleeger, S. L. (1999). "Albert Einstein and empirical software engineering." IEEE Computer 32(10): 32-37.

Pfleeger, S. L. (2005). "Soup or art? The role of evidential force in empirical software engineering." IEEE Software 22(1): 66-73.

Philips (1996). "PCA82C250/251 CAN transceiver". Philips Semiconductors,

Philips (2004). "SJA1000 stand-alone CAN controller".

Pickard, L. M., B. A. Kitchenham and P. W. Jones (1998). "Combining empirical results in software engineering." Information and Software Technology 40(14): 811-821.

Pimentel, J. R. and J. A. Fonseca (2004). "FlexCAN: A flexible architecture for highly dependable embedded applications". 3rd International Workshop on Real-Time Networks, Catania, Italy.

Polson, P. G., E. Muncher and G. Engelbeck (1986). "A test of common elements theory of transfer". Proceedings of the SIGCHI Conference on Human Factors in Computing Systems Boston, Massachusetts, United States, ACM Press.

Pont, M. J. (2001). "Patterns for time-triggered embedded systems: building reliable applications with the 8051 family of microcontrollers". Harlow, Addison Wesley/ACM Press. [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 25(3): 217-238.

Pont, M. J., A. J. Norman, C. Mwelwa and T. Edwards (2003). "Prototyping time-triggered embedded systems using PC hardware". Eighth European Conference on Pattern Languages of Programs (EuroPLoP), Irsee, Germany.

Pop, P. (2003). "Analysis and synthesis of communication-intensive heterogeneous real-time systems". Ph.D Thesis. Department of Computer and Information Science, Linköping University, Linköping, Sweden.

Popp, L. M., G. Seemann and O. Dössel (2004). "A simulation study of the reaction of human heart to biphasic electrical shocks." BMC Cardiovascular Disorders 4(9).

Prechelt, L. and B. Unger (2000). "An experiment measuring the effects of Personal Software Process (PSP) training." IEEE Transaction on Software Engineering 27(5): 465-472.

Purushothaman, R. and D. E. Perry (2005). "Towards understanding the rhetoric of small source code changes." IEEE Transaction on Software Engineering 31(6): 511-526.

Rajnak, A. and M. Ramnerfors (2002). "The Volcano communication concept". International Congress on Transportation Electronics, Society of Automotive Engineers Inc.

Redell, O., J. El-khoury and M. Törngren (2004). "The AIDA toolset for design and implementation analysis of distributed real-time control systems." Microprocessors and Microsystems 28(4): 163-182.

Robillard, M. P., W. Coelho and G. C. Murphy (2004). "How effective developers investigate source code: An exploratory study." IEEE Transaction on Software Engineering 30(12): 889-903.

Robillard, P. N., P. d'Astous, F. Détienne and W. Visser (1998). "Measuring cognitive activities in software engineering". Proceedings of the 20th international conference on Software engineering, Kyoto, Japan, IEEE Computer Society.

Rosenberg, L., T. Hammer and J. Shaw (1998). "Software metrics and reliability". 9th International Symposium on Software Reliability Engineering, Germany.

Rosenberg, L. and L. Hyatt (1997). "Software quality metrics for Object-Oriented environments." Crosstalk - The Journal of Defense Software Engineering(April).

Sachitanand, N. N. (2002). "Embedded systems - A new high growth area". The Hindu. Bangalore.

Sandfridson, M. (2000). "Timing problem in distributed real-time computer control problems". ISSN 1400-1179 Mechatronics Lab, Department of Machine Design, Royal Institute of Technology, KTH, Stockholm, Sweden.

Sanz, R. and J. Zalewski (2003). "Pattern-based control systems engineering." IEEE Control Systems Journal 23(3): 43-60.

Scanlan, D. A. (1989). "Structured flowcharts outperform pseudocode: An experimental comparison." IEEE Software 6(5): 28-36.

Schmitt, V. R., J. W. Morris and G. D. Jenney (1998). "Fly-by-Wire: A historical and design perspective". Warrendale, Society of Automotive Engineers. [0768002184].

Schneidewind, N. F. (1992). "Methodology for validating software metrics." IEEE Transaction on Software Engineering 18(5): 410-421.

Schneidewind, N. F. (1999). "Measuring and evaluating maintenance process using reliability, risk, and test metrics." IEEE Transaction on Software Engineering 25(6): 769-781.

Schneidewind, N. F. (2004). "Recommended practice on software reliability". Software Technology Conference, Salt Lake City, USA.

Schoitsch, E. (2003). "Embedded systems - introduction." ERCIM News 52: 10-11.

Seaman, C. B. (1999). "Qualitative methods in empirical studies of software engineering." IEEE Transaction on Software Engineering 25(4): 557-572.

SEL (1995). "Software measurement guidebook". SEL-94-102 NASA/GSFC,

Sevillano, J. L., A. Pascual, G. Jimenez and Civit-Balcells (1998). "Analysis of channel utilization for Controller Area Networks." Computer Communications 21(16): 1446-1451.

Shaw, A. C. (2001). "Real-time systems and software". New York, John Wiley & Sons Inc. [0471354902].

Sheil, B. A. (1981). "The psychological study of programming." ACM Computing Surveys 13(1): 101-120.

Shepperd, M. and C. Schofield (1997). "Estimating software project effort using analogies." IEEE Transaction on Software Engineering 23(12): 736-743.

Shneiderman, B., R. Mayer, D. McKay and P. Heller (1977). "Experimental investigations of the utility of detailed flowcharts in programming." Communications of the ACM 20(6): 373-381.

Short, M. J., J. Fang, M. J. Pont and A. Rajabzadeh (2006). "Assessing the impact of redundancy on the performance of a Brake-by-Wire system". Society of Automotive Engineers (SAE) World Congress, Detroit, USA.

Short, M. J. and M. J. Pont (2006). "Predicting the impact of hardware redundancy on the performance of embedded control systems". Proceedings of the 6th UKACC International Control Conference, Glasgow, Scotland.

Short, M. J., M. J. Pont and Q. Huang (2004a). "Simulation of a vehicle longitudinal dynamics". ESL 04-01 Embedded System Laboratory, University of Leicester, Leicester, England.

Short, M. J., M. J. Pont and Q. Huang (2004b). "Simulation of motorway traffic flows". ESL04-02 Embedded Systems Laboratory, University of Leicester, Leicester, England.

Siemens (1996). "C167 derivatives - User's manual". Version 2.0.

Siemens (1997). "Proceedings of the European Pattern Languages of Programming Conference".

Singhoff, F., J. Legrand, L. Nana and L. Marcé (2004). "Cheddar: A flexible real time scheduling framework". Proceedings of the 2004 annual ACM SIGAda international conference on Ada, Atlanta, USA

Sobel, A. E. K. and M. R. Clarkson (2002). "Formal methods application: An empirical tale of software development." IEEE Transaction on Software Engineering 28(3): 308-320.

Solingen, R. V. and P. Stalenhoef (1997). "Effort measurement of support to software products". Proceeding of the International Workshop on Empirical Studies of Software Maintenance, Bari, Italy.

Specks, J. W. and A. Rajnak (2002). "LIN - Protocols, development tools, and software interfaces for Local Interconnect Networks in vehicles". 9th International Conference on Electronic Systems for Vehicles.

Stark, G., R. C. Durst and C. W. Vowell (1994). "Using metrics in management decision making." IEEE Computer 27(9): 42-48.

Storey, N. (1996). "Safety-critical computer systems". Harlow, Addison-Wesley. [0201427877].

Strong, G. W. (1995). "An evaluation of the PRC touch talker with Minispeak: Some lessons for speech prosthesis design". Extra-ordinary human-computer interaction: Interfaces for users with disability. A. D. N. Edwards. Cambridge, Cambridge University Press: 47-57. [0521434130].

Subramanian, G. and W. Corbin (2001). "An empirical study of certain Objects-Oriented software metrics." Journal of Systems and Software 59(1): 57-63.

Subramanyam, R. and M. S. Krishnan (2003). "Empirical analysis of CK metrics for Object-Oriented design complexity: Implications for software defects." IEEE Transaction on Software Engineering 29(4): 297-310.

Tanenbaum, A. S. (1995). "Distributed operating systems". London, Prentice Hall. [0131439340].

Tanenbaum, A. S. and M. van Steen (2002). "Distributed systems: Principles and paradigms". New Jersey, Prentice Hall. [0130888931].

Temple, C. (1998). "Avoiding the babbling-idiot failure in a time-triggered communication system". Proceedings of the Twenty-Eighth Annual International Symposium on Fault-Tolerant Computing München, Germany.

Thomesse, J. P. (1998). "A review of the fieldbuses." Annual Reviews in Control 22: 35-45.

Thwin, M. M. T. and T. S. Quah (2005). "Application of neural networks for software quality prediction using Object-Oriented metrics." Journal of Systems and Software 76(2): 147-156.

Tindell, K. and A. Burns (1994). "Guaranteed message latencies for distributed safety-critical hard real-time control networks". YCS 229 Real-Time Systems Research Group, University of York, York, England.

Tindell, K. and J. Clark (1994). "Holistic schedulability analysis for distributed hard real-time systems." Microprocessing and Microprogramming - Euromicro Journal 40: 117-134.

Törngren, M. (1998). "Fundamentals of implementing real-time control applications in distributed computer systems." Journal of Real-Time Systems 14: 219-250.

Törngren, M., J. El-khoury, M. Sandfridson and O. Redell (2001). "Modelling and simulation of embedded computer control systems: Problem formulation". ISRN KTH/MMK-{01/3}-SE Mechatronics Laboratory, Department of Machine Design, Royal Institute of Technology, Stockholm, Sweden.

Turski, W. M. (1986). "And no philosophers' stone either". Information Processing 86, Elsevier Science (Northern Holland).

Ulam, S., R. D. Richtmyer and J. von Neumann (1947). "Statistical methods in neutron diffusion". LAMS-551 Los Alamos Scientific Laboratory, Los Alomos, USA.

Vessey, I. and S. A. Conger (1994). "Requirement specification: Learning object, process and data methodologies." Communications of the ACM 37(5): 102-113.

Vidler, P. and M. J. Pont (2006). "Computer assisted source-code parallelisation". International Conference on Computational Science and its Applications (ICCSA 2006), Glasgow, Scotland.

von Hanxleden, R., A. Botorabi and S. Kupczyk (1998). "A codesign approach for safety-critical automotive applications." IEEE Micro 18(5): 66-79.

Weller, E. F. (1994). "Using metrics to manage software projects." IEEE Computer 27(9): 27-33.

Wilson, R. "Inventor recalls birth of the MPU". EETimes Online. (Last accessed: 09/06/2006) *http://www.eet.com/story/OEG20011212S0047*

Wittenmark, B., J. Nilsson and M. Törngren (1995). "Timing problems in real-time control systems". Proceedings of the 1995 American Control Conference, Seattle, USA.

Wolf, A. and C. Koller (1998). "16-bit microcontroller with two on-chip CAN modules". 5ht International CAN Conference, San Jose, USA.

Wood, M., J. Daly, J. Miller and M. Roper (1999). "Multi-method research - An empirical investigation of Object-Oriented methodology." Journal of Systems and Software 48(1): 13-26.

Zheng, N. N., S. Tang, H. Cheng, Q. Li, G. Lai and F. Y. Wang (2004). "Towards intelligent driver-assistance and safety warning systems." IEEE Intelligent Systems 19(2): 8-11.

Zuberi, K. M. and K. G. Shin (1995). "Non-preemptive scheduling of messages on Controller Area Network for real-time control applications". Proceedings of the First IEEE Real-Time Technology and Applications Symposium, Chicago, USA.

APPENDICES

Appendix-A Observing the development of reliable embedded systems

This appendix includes a copy of the Ayavoo et al. (2005a) paper.

Abstract. Distributed embedded systems are becoming ubiquitous and increasingly complex. It is frequently assumed that the use of simulation can support the design and implementation of such systems. However the contribution made by simulation towards the development process is rarely explored in depth and is incompletely understood. The pilot study described in this paper was intended to help identify techniques which may be used to provide a quantitative assessment of the contribution which simulation makes in this area. The study involved the observation of the "simulation first" development of a distributed embedded system. The results obtained in the study are described, and will form the basis for future investigations in this important area.

1. Introduction

Distributed embedded systems are becoming increasingly common and increasingly complex. For example, the designer of a modern passenger car may need to choose between the use of one (or more) network protocols based on CAN, TTCAN, LIN, FlexRay or TTP/C. The resulting network may be connected in, for example, a bus or star topology. The individual processor nodes in the network may use event-triggered or time-triggered software architectures, or some combination of the two. The clocks associated with these processors may be linked using, for example, shared-clock techniques or synchronization messages. These individual processors may, for example, be C167, ARM, MPC555 or 8051.

Overall, the number of possible system designs is enormous and prototyping even a small subset of these possibilities is impractical: an alternative approach is therefore required (Thane, 2000). According to Karatza: "The most straightforward way to evaluate the performance without a full-scale implementation is through a modelling and simulation approach. Detailed simulation models help determine performance bottlenecks inherent in the architecture and provide the basis for refining the system configuration."(Karatza, 2004, p. 183). Various other investigators have also argued for the use of simulation to support the development of this type of system (El-khoury and Törngren, 2001; Palopoli *et al.*, 2001; Törngren *et al.*, 2001; Castelpietra *et al.*, 2002; Cervin *et al.*, 2003; Redell *et al.*, 2004).

Intuitively, one might expect that use of an appropriate simulator may assist in the development process. However, there is very little empirical data available that can demonstrate that - for example - simulation reduces the overall effort in a project. Similarly, even if use of a simulator does reduce the required effort, it is not clear exactly how such a tool should be employed. Should we, for example, simulate the whole system then build it, or should we simulate part of the system, build this, then simulate the next part, and so on?

The lack of empirical evidence is not, of course, a problem unique to the use of simulators and the software-engineering community is becoming increasingly aware of the need to seek evidence of the effectiveness of any form of new technology (Pickard *et al.*, 1998;

Germain and Robillard, 2005), rather than relying on sweeping statements about its "obvious" effectiveness (e.g. see Turski, 1986; Fenton *et al.*, 1994).

In light of the observations above, the pilot study described in this paper was intended to help identify techniques which can be used in future experiments in order to provide a quantitative assessment of the contribution which simulation makes in this area. The study involved the observation of the development of a non-trivial distributed embedded control system, first using a simulator and then developing "real code" for a network of suitable microcontrollers.

The paper is organised as follows. Section 2 describes in detail the case study used in the project described here. A comparison of the results from the simulation process and the hardware implementation is presented in Section 3. Section 4 then presents the raw results from the measures of "effort" for the two development processes. Synchronisation of the timescales for these processes is then discussed in Section 5. An analysis of the synchronised results is then presented in Section 6. Finally, the results are discussed and conclusions are presented in Section 7.

2. The case study

As outlined in the introduction, the study described in this paper involved the observation of the "simulation first" development of a distributed embedded control system.

We describe the study in this section.

2.1 The testbed

The testbed used in this study was based on part of an X-by-Wire control system for a passenger car: it was adapted from a platform described by Castelpietra et al. (see Castelpietra *et al.*, 2002). In our version of Castelpietra's system, six embedded nodes were connected using a CAN bus (see Figure A-1). Each node contained an Infineon "167" microcontroller: such devices are widely used in the automotive sector (Siemens, 1996.

The case study was developed using a Shared-Clock CAN (SCC) architecture using a tick interval of 1ms and a CAN baudrate of 500Kbits/sec (Pont, 2001). A FIFO buffer was designed to ensure that the messages were queued correctly.

The characteristics of the tasks and messages are illustrated in Table A-1.

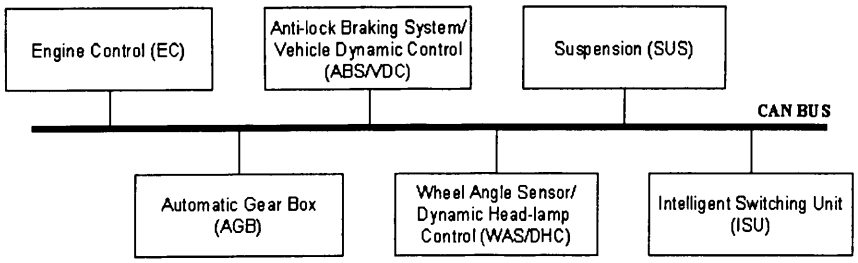

Figure A-1 Test bed used in this study (based on Castelpietra *et al.*, 2002).

Table A-1 Task and message characteristics of the six distributed nodes. The Initial Delay and Period values are in milliseconds. Each task on the node is associated with either sending or receiving a message on the CAN bus.

Task Name	Node	Priority	Initial Delay	Period	Input Message	Output Message
EC Task1	EC	1	1	10		M1
EC Task2		2	2	20		M3
EC Task3		3	3	100		M10
EC Task4		4	4	7	M4	
EC Task5		5	5	7	M2	
EC Task6		6	6	25	M8	
EC Task7		7	7	20	M6	
AGB Task1	AGB	1	1	15		M4
AGB Task2		2	2	50		M11
AGB Task3		3	3	25	M8	
AGB Task4		4	4	7	M2	
ABS Task1	ABS/VDC	1	1	20		M5
ABS Task2		2	2	40		M6
ABS Task3		3	3	15		M7
ABS Task4		4	4	100		M12
ABS Task5		5	5	10	M3	
ABS Task6		6	6	10	M9	
WAS Task1	WAS/DHC	1	1	14		M2
WAS Task2		2	2	10	M9	
SUS Task1	SUS	1	1	20		M9
SUS Task2		2	2	10	M5	
SUS Task3		3	3	5	M1	
SUS Task4		4	4	7	M2	
SUS Task5		5	5	7	M7	
ISU Task1	ISU	1	1	50		M8
ISU Task2		2	2	25	M11	
ISU Task3		3	3	5	M1	
ISU Task4		4	4	50	M10	
ISU Task5		5	5	20	M6	
ISU Task6		6	6	10	M9	
ISU Task7		7	7	50	M12	

2.2 Selecting a suitable simulator

To develop the system described in Section 2.1, a "simulation first" approach was employed. The simulation process was used to predict the response times of all the signals between the nodes, in order to avoid the expense of repeated prototyping at the implementation stage (Redell *et al.*, 2004).

A number of simulation tools have been developed by different research groups and companies. For example, El-khoury and Törngren (El-khoury and Törngren, 2001) described a toolset which integrates the modelling of schedulers and distributed systems with models of control system performance. This was later expanded to form a more versatile simulation tool called AIDA (Redell *et al.*, 2004). Another tool – RTSIM - was described by Palopoli *et al.* (Palopoli *et al.*, 2001) to help engineers to develop real-time distributed embedded control systems. Similarly Eker and Cervin (Eker and Cervin, 1999) described a Matlab toolbox simulating a real-time scheduler: this was later extended to develop the TrueTime Simulator (Cervin *et al.*, 2003).

In the present study, the TrueTime simulator was chosen to simulate the behaviour of the system for the following reasons:

- TrueTime is capable of simulating the system to be controlled (that is, the plant dynamics) and a wide range of software architectures and network protocols for the distributed control system.
- Many control engineers are familiar with Matlab and Simulink (e.g. see Dutton *et al.*, 1997; Dorf and Bishop, 2000). Since TrueTime is a Matlab / Simulink package, this makes it easy to integrate the software and network design process with the development of the control system.
- TrueTime is an open-source package. This provides obvious advantages in terms of cost, and also provides flexibility: both are important considerations in a research project such as that described in this paper.

2.3 How do we measure (and compare) the effort involved?

There has been comparatively little research into the measurement of effort involved in development of software-rich systems (Solingen and Stalenhoef, 1997; Basili *et al.*, 2004; Germain and Robillard, 2005).

In the present study we first needed to understand the work involved in the development of both the "simulated" and "real" systems.

The work involved in the simulation phase included creating block diagrams (with the appropriate connections as shown in Figure A-2) for the system and subsequently writing Matlab codes for the TrueTime simulator. An example of a task in Matlab is shown in Listing A-1.

All of the work involved in the implementation phase required coding in C. For example, the task specified in TrueTime in Listing A-1 is shown in C in Listing A-2.

Since both the simulation and implementation phases required coding, we used code-based metrics as the basis of the comparisons that were carried out. Please note that although the languages of the simulation and implementation are different, the structure and logic of the tasks are similar. In addition, for the simulation phase, we included the time required to build the block diagrams in the measures of effort required to develop the code since the code and diagrams are interdependent.

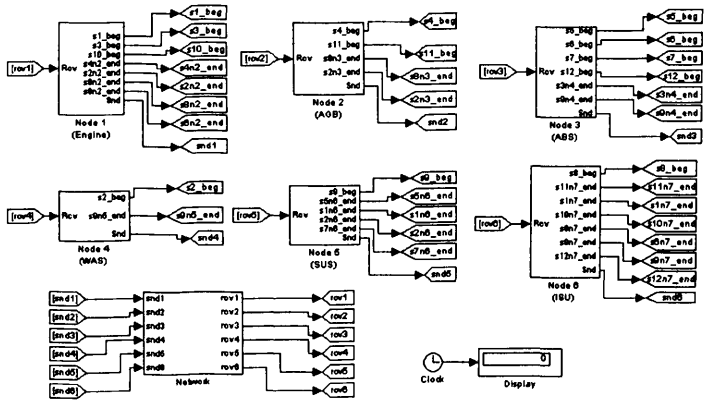

Figure A-2 The TrueTime block diagrams for the system described in Section 2.1.

```
function [exectime, data] = N1_Task7(seg, data)
global N1_MSG_6;
switch seg,
   case 1,
      Sig_6 = N1_MSG_6(3);
      % Turn on the LED on and off
      if (Sig_6 == 1)
         data.Sig_out = 1;
      else
         data.Sig_out = 0;
      end

      exectime = 0.0001;
   case 2,
      ttAnalogOut(data.out7Chan, data.Sig_out);
      exectime = -1;
end
```

Listing A-1 An example of a task specification created using the TrueTime simulator.

```
void N1_Task7(void)
   {
   // Turn on the LED on and off (reverse logic)
   if (Msg_6)
      {
      Sig_pin_7 = 0;
      }
   else
      {
      Sig_pin_7 = 1;
      }
   }
```

Listing A-2 The task specified in TrueTime in Listing A-1 is shown in C here.

Three parameters were chosen to assess the effort involved:

- Development time;
- Lines Of Code (LOC);
- McCabe's Cyclomatic Complexity v(G) (McCabe, 1976).

Time was chosen as it has been used before to measure the software development process (Solingen and Stalenhoef, 1997; Basili *et al.*, 2004). LOC was used as it can measure effort in terms of product size (Stark *et al.*, 1994; Weller, 1994). McCabe's Cyclomatic Complexity was chosen as it is a popular way of assessing and comparing code complexity (McCabe, 1976; Stark *et al.*, 1994).

2.4 Deciding on a suitable frequency of measurement

Having decided what to measure, we needed to decide when to measure. In general, we wish to have as much data as possible: thus a continuous sampling technique would seem to be ideal. However, carrying out the measurements is likely to have an impact on the process under observation (SEL, 1995): that is, the more frequently a measurement is taken, the more likely we are to influence the development process itself.

In this study, it was decided that measurements should be taken every time a "new version" of the software was developed: this technique is usually used to keep track of

code development that is constantly evolving (Tichy, 1985). Copies of all the versions created were retained for subsequent analysis.

2.5 The observer and observee

Finding suitable test subjects to carry out an experiment is a common problem in empirical software engineering (Pickard *et al.*, 1998). In this pilot study, we used one observee (one of the authors – DA): the same individual acted as the observer. This "one person" approach has previously been shown to be effective (see Basili and Turner, 1975).

3. Does the simulator work?

After the simulation and implementation processes were completed, the first verification test was to determine if the results obtained from the simulation matched those from the hardware implementation. To do this, measurements of the response time for each of the signals (19 in total) were made. Results for the TrueTime simulator were collected by running the experiment on Simulink and logging the response time in a .MAT file. The signals on the hardware implementation were measured using LabView via a National Instruments data acquisition card (NI PCI-6035E).

The results obtained showed that the response time of all the 19 signals from the simulation matched closely the measured results from the hardware (within 2 ms)

4. Raw results

A new version of the software was saved every time the project reached a significant milestone. The versions produced are summarised in Table A-2.

Table A-2 Description of all the versions that were created for the software simulation and hardware implementation and their corresponding development time in hours[1].

Version	Description	Duration
SIM v1	Created one Master (M) and two slaves (S1 & S2) SCC configuration. S1 sends a signal to S2 and S2 sends a signal to S1.	5
SIM v2	Similar to SIM v1 but uses a basic message queue.	4
SIM v3	Increased to five slaves with basic message queue but couldn't work.	2
SIM v4	Reverted back to SIM v2 and gradually added another three slaves (S3, S4 & S5). S3, S4 & S5 only took part in the SCC, but not in the message queue.	2
SIM v5	Added another two signals, one from S3 and one from M. S4 & S5 acted as receivers.	2
SIM v6	Ported Castelpietra's testcase to the existing platform (SIM v5).	5
HW v1	Implemented the basic SCC architecture on all the six nodes.	7
HW v2	Developed the initial message queue. Works for only a single message sent out from slave.	4
HW v3	Code was modified to send multiple messages out from the slaves.	3
HW v4	Master node was modified to also have the capability to send multiple messages. Got a basic system working on three nodes, where a message is sent from S1–S2 through M.	4
HW v5	Tested the message transfer for master-to-slave, slave-to-master and slave-to-slave.	4
HW v6	Implemented five nodes message queue system with all nodes sending a signal out.	4
HW v7	Begin to port Castelpietra's testcase for M, S1, S2, S4 and S5.	3
HW v8	Continued porting Castelpietra's testcase for S3. Performed checks for all tasks, signals and message properties.	2
HW v9	Modified the receiving signal for LabView measurement purpose.	1

[1] There was one problem with the raw data obtained for HW v1. The raw data showed duration of seven hours. However, this included the time (five hours) taken to detect a hardware error on a faulty 167 board. Assuming no hardware error was present, the actual development time for HW v1 would have been two hours, and this figure will be used in the remainder of the paper.

Please note that for the first version of the hardware implementation (HW v1) and software simulation (SIM v1), the number of nodes used in both processes were different. For the HW v1, all the six nodes were initially connected with the SCC approach to test the development boards for any potential faults. This check was not necessary on the simulator as there were no physical hardware component involved. Therefore, we decided to begin testing the SCC architecture with only three nodes.

5. Synchronising the timescales

As can be seen from Table A-2, there were six different simulation versions produced and nine versions of the hardware implementation. This makes comparison of the two development processes difficult. In order to be able to carry out a meaningful comparison of the two development processes, we needed to synchronise the two development processes.

The first step we took was to divide the development process into three stages:

- To develop the basic SCC
- To implement a message queue within the scheduler
- To port Castelpietra's testcase onto the existing platform

These three stages were then mapped onto the various versions of the software development as illustrated in Table A-3.

Table A-3 Mapping of versions to development stages.

Simulation		Hardware		Development Stage
Version X	Duration	Version X	Duration	
SIM v1	5	HW v1	2	Basic SCC (A)
SIM v2	4			
SIM v3	2	HW v2	4	Message Queue (B)
SIM v4	2	HW v3	3	
SIM v5	2	HW v4	4	
		HW v5	4	
		HW v6	4	
SIM v6	5	HW v7	3	Castelpietra (C)
		HW v8	2	
		HW v9	1	
Total	20	Total	27	

Based on Table A-3, it can be seen that for the three stages (A, B and C), the number of versions produced during simulation and hardware implementation were different. In order to produce the same number of versions in each stage, two approaches were employed:

- Grouping 'similar' versions together
- Adding 'dummy' versions

5.1 Grouping versions together

Based on Table A-3, it can be seen that Development Stage A and C seem to have only one version for the hardware and simulation respectively. On the simulator, there were two versions in Development Stage A. These two versions (SIM v1 and SIM v2) were grouped together into SIM v1. Note that the number of hours of the 'new' SIM v1 was the summation of SIM v1 and SIM v2. The similar process was also carried out for the hardware (Development Stage C). The resulting data are summarized in Table A-4.

Table A-4 Results after grouping versions together.

Simulation		Hardware		Development Stage
Version X	Duration	Version X	Duration	
SIM v1	9	HW v1	2	Basic SCC (A)
SIM v2	2	HW v2	4	Message Queue (B)
SIM v3	2	HW v3	3	
SIM v4	2	HW v4	4	
		HW v5	4	
		HW v6	4	
SIM v5	5	HW v7	6	Castelpietra (C)
Total	**20**	**Total**	**27**	

5.2 Adding dummy versions

Dummy versions are added to a particular stage when we wish to retain the resolution of the results (grouping them would reduce the resolution). From Table A-4, it can be seen that in Stage B the simulation process resulted in the creation of three versions while the hardware process generated five versions. Since Stage B showed significant development effort, it was decided that two dummy version will be added in the simulation results as shown in Table A-5. The dummy versions were added such that they were evenly distributed in the designated development stages.

Table A-5 Adding dummy task.

Simulation		Hardware		Development Stage
Version X	Duration	Version X	Duration	
SIM v1	9	HW v1	2	Basic SCC (A)
SIM v2	2	HW v2	4	Message Queue (B)
Dummy	2	HW v3	3	
SIM v3	2	HW v4	4	
Dummy	2	HW v5	4	
SIM v4	2	HW v6	4	
SIM v5	5	HW v7	6	Castelpietra (C)
Total	**20**	**Total**	**27**	

Note that for the task duration, the previous value before the dummy version was inherited for the dummy version. This will have the effect of 'stretching' Development Stage B of the simulation such that the timescale is synchronised with the hardware implementation. However, notice that the total task duration has not been changed. This will preserve the actual time properties of the simulation process. The similar procedure was carried out for LOC and McCabe's Cyclomatic Complexity. The final results are shown in Table A-6.

Table A-6 Results for all attributes after performing the timeline synchronisation.

Simulation				Hardware				Development Stage
Version X	Duration	LOC	v(G)	Version X	Duration	LOC	v(G)	
SIM v1	9	312	21	HW v1	2	6095	184	Basic SCC (A)
SIM v2	2	322	24	HW v2	4	2611	70	Message Queue (B)
Dummy	2	322	24	HW v3	3	2655	74	
SIM v3	2	649	52	HW v4	4	3954	109	
Dummy	2	649	52	HW v5	4	3997	116	
SIM v4	2	1025	91	HW v6	4	6612	218	
SIM v5	5	1190	101	HW v7	6	7752	250	Castelpietra (C)

6. Analysis of the synchronised results

Using the synchronised results (Table A-6), the measured attributes for the simulation and hardware implementation were plotted. Figure A-3 compares the effort (in terms of working hours) for the simulation and the hardware.

Part I on the graph took longer for the simulation than on the hardware because the SCC architecture was already available on the hardware (as a software pattern Pont, 2001) but

had to be built from scratch for the simulator. Part II was where the message queue was developed. Here, the simulation took less time than the hardware because a FIFO architecture was already available on the simulator (as a message box Henriksson and Cervin, 2004) but not on the hardware. Part III was where testing and further enhancement of the message queue was carried out. Again, this part required more effort on the hardware than on the simulator due to the availability of a message queue.

Finally, Castelpietra's testcase was ported onto the two platforms (Part IV). In this part, both the hardware and simulator show an increase in effort. This is due to the manual work of porting all the tasks and message properties to the existing platform.

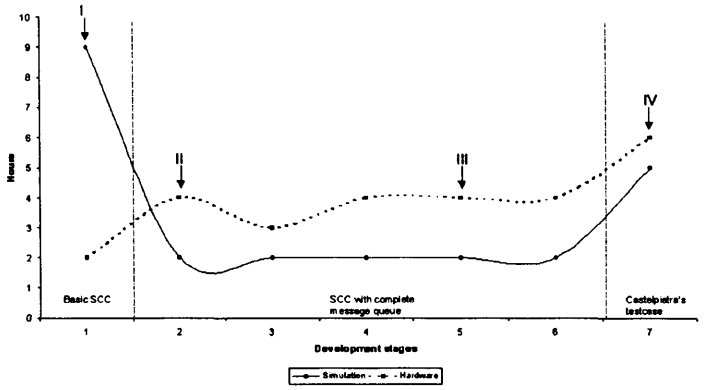

Figure A-3 Comparison of the effort (time) between the simulation and hardware.

Looking at Figure A-4 and Figure A-5, it can be seen that LOC and $v(G)$ follow a similar trend. Both the results indicate that more effort (in terms of LOC and $v(G)$) was needed on the hardware than on the simulator.

The difference in Part I was due to the number of nodes that were initially used to set up the SCC. The hardware used six nodes whereas the simulator only used three. As the development stage progressed, the growth rate of the LOC and $v(G)$ was much more rapid on the hardware than on the software.

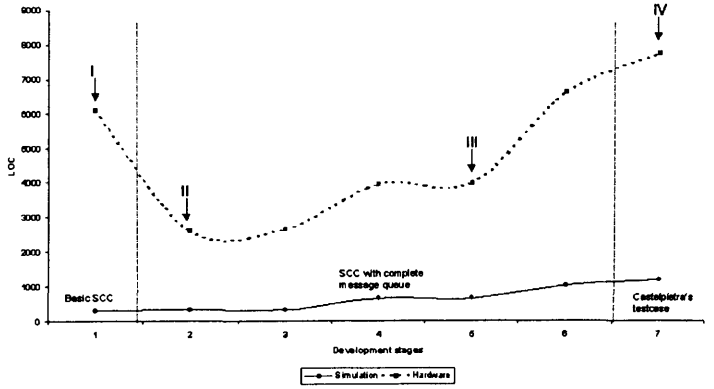

Figure A-4 Comparison of the effort (LOC) between the simulation and hardware.

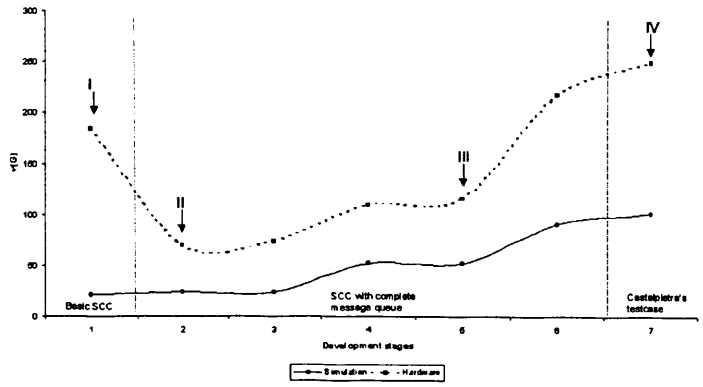

Figure A-5 Comparison of the effort (McCabe) between the simulation and hardware.

7. Discussion and conclusions

The pilot study described in this paper was intended to help identify techniques which can be used in future experiments in order to provide a quantitative assessment of the contribution which simulation makes in the development of distributed embedded systems. The study involved the observation of the development of a non-trivial distributed embedded control system, first using a simulator and then developing "real code" for a network of suitable microcontrollers.

The results presented here involved only a single study, involving one developer. It would be inappropriate to claim that the findings presented here provide solid evidence for (or against) the use of simulation in the development of distributed embedded control systems. However, the results from this study do suggest that the measurements made here, used in conjunction with techniques for timescale synchronisation, are worth pursuing in a future study involving larger numbers of test subjects.

Acknowledgements – This work is supported by an ORS award (to DA) from the UK Government (Department for Education and Skills), and by Pi Technology. Work on this paper was completed while MJP was on Study Leave from the University of Leicester.

References

Basili, V., Asgari, S., Carver, J., Hochstein, L., Hollingsworth, J.K., Shull, F. and Zelkowitz, M.V. A pilot study to evaluate development effort for high performance computing *Technical report CS-TR-4588*, University of Maryland, 2004.

Basili, V.R. and Turner, A.J. Iterative enhancement: A practical technique for software development. *IEEE Transaction on Software Engineering, 1* (4). 390-396.

Castelpietra, P., Song, Y.Q., Lion, F.S. and Attia, M. Analysis and simulation methods for performance evaluation of a multiple networks embedded architecture. *IEEE Transactions on Industrial Electronics, 49* (No. 6).

Cervin, A., Henriksson, D., Lincoln, B., Eker, J. and Årzén, K. How does control timing affect performance? - Analysis and simulation of timing using jitterbug and TrueTime. *IEEE Control Systems Journal, 23* (3). 16-30.

Dorf, D. and Bishop, R. *Modern control systems*. Prentice-Hall, New Jersey, USA, 2001.

Dutton, K., Thompson, S. and Barraclough, B. *The art of control engineering*. Addison Wesley, 1997.

Eker, J. and Cervin, A., A Matlab toolbox for real-time and control systems co-design. in *Proceedings of the 6th International Conference on Real-Time Computing Systems and Applications*, (1999).

El-khoury, J. and Törngren, M., Towards a toolset for architectural design of distributed real-time control systems. in *IEEE Real-Time Symposium*, (London, England, 2001), IEEE.

Fenton, N., Pfleeger, S.L. and Glass, R.L. Science and substance: A challenge to software engineers. *IEEE Software, 11* (4). 86-95.
Germain, E. and Robillard, P.N. Engineering-based processes and Agile methodologies for software development: A comparative case study. *Journal of Systems and Software, 75* (1-2). 17-27.
Henriksson, D. and Cervin, A. TrueTime 1.2—reference manual, Department of Automatic Control, Lund Institute of Technology, Lund, Sweden, 2004.
Karatza, H.D. Modeling and simulation of distributed systems and networks. *Simulation Modelling Practice and Theory, 12* (3-4). 183-185.
McCabe, T. A software complexity measure. *IEEE Transactions on Software Engineering, 2*. 308-320.
Palopoli, L., Lipari, G., Abeni, L., Abeni, M.D., P.Ancilotti and Conticelli, F., A tool for simulation and fast prototyping of embedded control systems. in *Proceedings of LCTES01*, (Snow Bird, Utah, United States, 2001), ACM Press.
Pickard, L.M., Kitchenham, B.A. and Jones, P.W. Combining empirical results in software engineering. *Information and Software Technology, 40* (14). 811-821.
Pont, M.J. *Patterns for time-triggered embedded systems*. Addison Wesley, 2001.
Redell, O., El-khoury, J. and Törngren, M. The AIDA toolset for design and implementation analysis of distributed real-time control systems. *Microprocessors and Microsystems, 28* (4). 163-182.
SEL. Software measurement guidebook *Software Engineering Laboratory Series*, NASA/GSFC, 1995.
Siemens. C167 derivatives - User's manual *Version 2.0*, 1996.
Solingen, R.V. and Stalenhoef, P., Effort measurement of support to software products. in *Proceeding of the International Workshop on Empirical Studies of Software Maintenance*, (Bari, Italy, 1997).
Stark, G., Durst, R.C. and Vowell, C.W. Using metrics in management decision making. *IEEE Computer, 27* (9). 42-48.
Thane, H. Monitoring, testing and debugging of distributed real-time systems *Mechatronics Laboratory, Department of Machine Design*, Royal Institute of Technology, Sweden, 2000.
Tichy, W.F. RCS - A system for version control. *Software - Practice & Experience, 15* (7). 637-654.
Törngren, M., El-khoury, J., Sandfridson, M. and Redell, O. Modelling and simulation of embedded computer control systems: Problem formulation, Mechatronics Laboratory, Department of Machine Design, Royal Institute of Technology, Sweden, 2001.
Turski, W.M., And no philosophers' stone either. in *Information Processing 86*, (1986), Elsevier Science (Northern Holland), 1077-1080.
Weller, E.F. Using metrics to manage software projects. *IEEE Computer, 27* (9). 27-33.

Appendix-B Two novel shared-clock scheduling algorithms for use with CAN-based distributed systems

This appendix includes a copy of the Ayavoo et al. (accepted) paper on the various SCC techniques. An earlier version of this paper was published in Ayavoo et al. (2005b).

Abstract. The Controller Area Network (CAN) protocol is widely employed in the development of distributed embedded systems. Previous studies have illustrated how a "Shared-Clock" (S-C) algorithm can be used in conjunction with CAN-based microcontrollers to implement time-triggered network architectures. This study explores some limitations of the existing S-C algorithms ("TTC-SC1" and "TTC-SC2"), and introduces two new algorithms (TTC-SC3 and TTC-SC4). The results presented in the paper suggest that TTC-SC3 and TTC-SC4 are useful additions to the range of shared-clock algorithms.

1. Introduction

Over recent years, we have considered various ways in which time-triggered software architectures can be employed in low-cost embedded systems where reliability is a key design consideration (e.g. Pont, 2001; Pont, 2003; Pont and Banner, 2004). Our previous work in this area has focused on the development of both single- and multi-processor designs. In the case of multi-processor designs, we have sought to demonstrate that a "Shared-Clock" (S-C) architecture provides a simple, flexible platform for many systems (Pont, 2001). In such designs, the Controller Area Network (CAN) protocol - introduced by Robert Bosch GmbH in the 1980s (Bosch, 1991) - provides high-reliability communications at low cost (Fredriksson, 1994; Sevillano *et al.*, 1998; Thomesse, 1998; Farsi and Barbosa, 2000). Since the CAN protocol has become widely used in many sectors, such as automotive and automation (Fredriksson, 1994; Zuberi and Shin, 1995; Sevillano *et al.*, 1998; Thomesse, 1998; Pazul, 1999; Farsi and Barbosa, 2000; Misbahuddin and Al-Holou, 2003), most modern microprocessor families now have members with on-chip support for this protocol (e.g. Philips, 1996; Siemens, 1997; Infineon, 2004; Philips, 2004).

The original S-C protocols were introduced in 2001 (Pont, 2001). In this paper, we consider some of the features and limitations of two such protocols, which will be referred to here as "TTC-SC1" and "TTC-SC2"[2]. We go on to present two new S-C protocols – TTC-SC3 and TTC-SC4 – which have features better matched to the needs of some applications.

The paper is organised as follows. Section 2 and Section 3 of the paper gives an overview of the TTC-SC1 and TTC-SC2 algorithms, respectively. Section 4 discusses some of the limitations of TTC-SC1 and TTC-SC2. TTC-SC3 and TTC-SC4 are introduced in Section 5 and Section 6, respectively. An initial evaluation of the TTC-SC3 and TTC-SC4 algorithms is presented in Section 7. Section 8 goes on to discuss some of the weaknesses of the TTC-SC3 and TTC-SC4 algorithms. Our conclusions are presented in Section 9.

[2] Please note that the algorithm referred to here as "TTC-SC1" was referred to as "SCC Scheduler" in the original publication (Pont, 2001). "TTC-SC2" was originally viewed as a variant of TTC-SC1.

2. The TTC-SC1 algorithm

Although CAN is often viewed as an event-triggered protocol (Leen and Heffernan, 2002), it has been shown that time-triggered behaviour can be achieved using TTC-SC1 (Pont, 2001). An overview of the TTC-SC1 algorithm is presented in this section.

2.1 Synchronising the nodes

The TTC-SC1 algorithm is a time division multiple access (TDMA) protocol based on CAN. The key idea behind TTC-SC1 is to synchronise the clocks on the individual nodes by sharing a single clock source between the various processor boards (see Figure B-1).

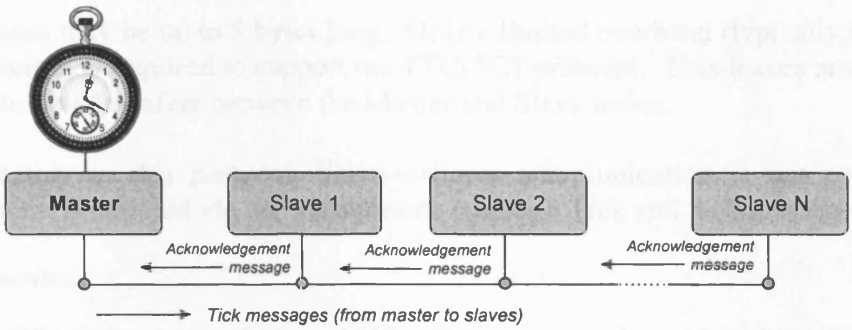

Figure B-1 Communication between Master and Slaves nodes in S-C architectures.

In this case, we have a single accurate clock on the Master node that generates periodic timer interrupts to drive the scheduler (for example a time-triggered co-operative scheduler in Pont, 2001) of the Master node. In addition, the Master node also generates a Tick message that is sent to the Slave nodes connected on the network using a CAN message. All the Slave nodes respond to the Tick message by generating an interrupt (from the CAN hardware). This interrupt will in turn be used to drive the scheduler of the Slave nodes.

2.2 Detecting communication and node failures

Besides synchronising the individual nodes on the network, TTC-SC1 is also responsible for detecting network and node failures. TTC-SC1 does this by having the Slave nodes return an "Acknowledgement" (Ack) message back to the Master node (Figure B-1). This way, the Master node will know the status of all its Slaves after one TDMA round

With each Tick message, the Master node identifies which Slave should return an Ack message by embedding that particular Slave ID in the data stream. Only the Slave with this particular ID will send an Acknowledgement back to the Master. The Master will then check the status of this Slave, and send the next Tick message out with a new Slave ID.

Figure B-2 illustrates an example of the TDMA round for a network with one Master and three Slaves, where Tick messages originate from the Master while Ack-X message is transmitted from Slave-X.

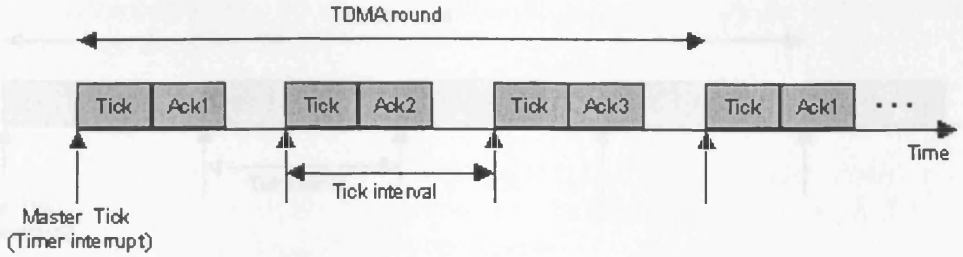

Figure B-2 The round-robin TDMA configuration using TTC-SC1.

2.3 Exchanging data between the nodes

CAN messages may be up to 8 bytes long. Only a limited overhead (typically up to 1 byte in each message) is required to support the TTC-SC1 protocol. This leaves around 7 bytes / message for data transfers between the Master and Slave nodes.

Please note that in this protocol, Slave-to-Slave communication is not permitted: all communication is directed via the Master node (through Tick and Ack messages).

2.4 Implementation

In the TTC-SC1 algorithm, only two CAN messages are exchanged within a Tick interval. Typically, on the Master node, CAN Message Object (CMO) 0 will be configured to send the Tick messages. A second CMO will be configured to receive the Ack messages from all the Slaves. Similarly, on the Slave nodes, CMO 0 will usually be configured to receive Tick messages and CMO 1 will be configured to transmit the Ack messages.

On the Master node, no CAN interrupts should be employed. On the Slave nodes, the CAN interface will be configured to generate a CAN interrupt upon receipt of a valid Tick message.

3. The TTC-SC2 algorithm

In some networks, the round-robin approach used to communicate with the Slave nodes in TTC-SC1 may not be efficient. For example, it may be that the Master node is required to check the status of a particular Slave node more frequently than the other Slaves. To do this, a modified version of the TTC-SC1 algorithm can be used: this is referred to here as "TTC-SC2".

In the TTC-SC2 algorithm, the configuration of the TDMA round is assumed to be flexible. For example, for the similar system illustrated in Figure B-2, it may be necessary that the status of Slave 1 is checked more frequently. Using TTC-SC2, the TDMA round illustrated in Figure B-3 may be more suitable.

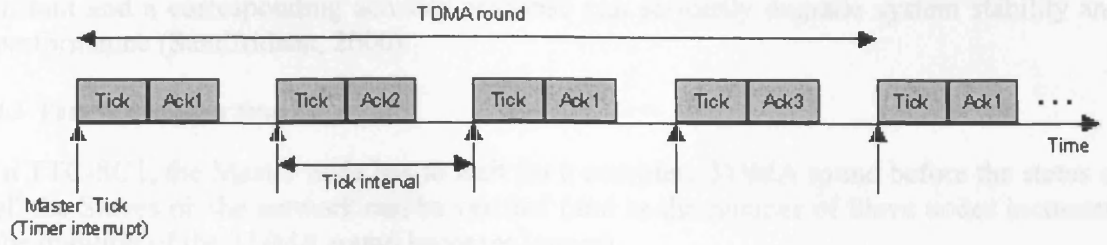

Figure B-3 A different TDMA round for a four-node system using TTC-SC2.

4. Problems with TTC-SC1 and TTC-SC2

We consider some of the drawbacks of TTC-SC1 and TTC-SC2 in this section.

4.1 Overview

The TTC-SC1 and TTC-SC2 algorithms are very simple and allow the creation of low-cost, time triggered, CAN-based networks with highly predictable patterns of behaviour. The algorithms are flexible and can also be used with a range of other network protocols, including RS485, without difficulty (see Pont, 2001).

However – inevitably – neither algorithm is a perfect match for all applications. In particular, when used with CAN, both TTC-SC1 and TTC-SC2 have the following limitations:

i) Direct transfer of messages between Slave nodes is not supported, with the consequence that Slave-to-Slave transmission times are comparatively long.

ii) To detect the failure of a given Slave node will take up to $(N+1) \times T$ seconds (where N is the number of Slave nodes, and T is the network Tick interval).

iii) They suffer from task jitter, due to CAN bit stuffing.

We consider each of these issues in more detail in the remainder of this section.

4.2 Slave-to-Slave message latency

In TTC-SC1 and TTC-SC2, the design of the (TDMA) protocol means that all communication between Slave nodes is directed via the Master node. This makes bus traffic easy to predict, but increases the delays involved in Slave-to-Slave communications.

For example, if all the nodes on the network – including the Master – are sending eight-byte data messages, this makes message piggy-backing (see Tindell and Burns, 1994) impossible. Therefore, for each Tick, the Master needs to decide which data message should be relayed out to the Slaves. The technique used to relay the messages could be either priority-based or round-robin. Relaying the messages out to the Slaves will cause additional delay to the Slave-to-Slave message latency.

Additional delays such as these can sometimes have a detrimental impact on overall system performance. In control systems, for example, large delays between a sampling

instant and a corresponding actuator response can seriously degrade system stability and performance (Sandfridson, 2000).

4.3 Failure detection time

In TTC-SC1, the Master node has to wait for a complete TDMA round before the status of all the Slaves on the network can be verified (and as the number of Slave nodes increases, the duration of the TDMA round becomes longer).

The worst-case failure detection time for the TTC-SC1 algorithm is given by Equation 1:

$$\textit{Failure detection time} = (\textit{Number of slaves} + 1) \times \textit{Tick} - \textit{CANTickmsg} \qquad (1)$$

where *CANTickMsg* refers to the time taken for the Master node to transmit a Tick message

Take the system illustrated in Figure B-4 as an example. Here, it would take up to four Tick intervals for the Master to detect a failure on Slave 1.

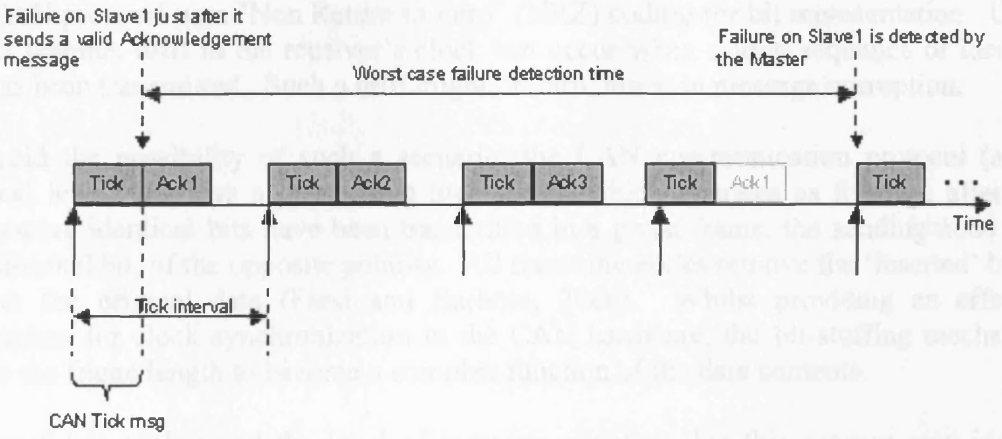

Figure B-4 Failure detection time for TTC-SC1.

This delay could be slightly reduced by using the TTC-SC2 algorithm, as illustrated in Figure B-5. However, a failure of a "low priority" Slave (one that is sent Tick messages infrequently) - such as Slave2 - will still take a long time to be detected by the Master.

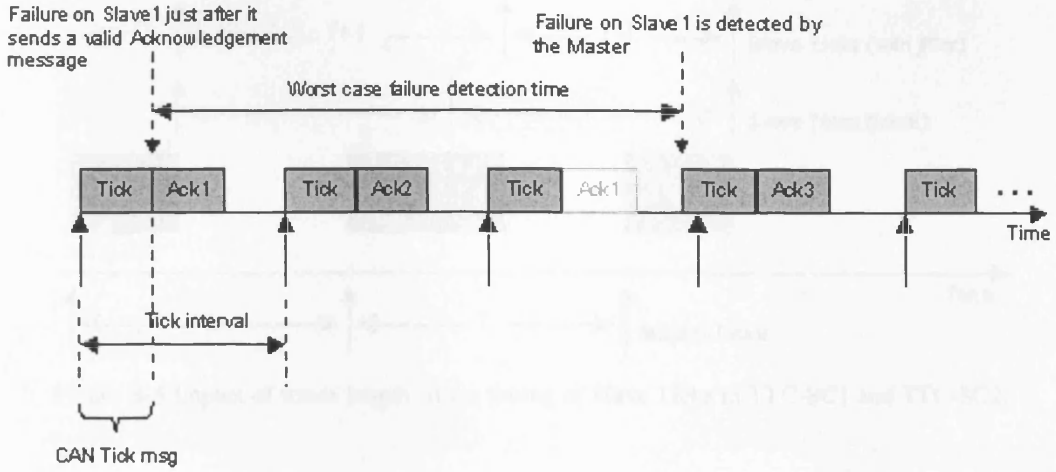

Figure B-5 Failure detection time for TTC-SC2.

4.4 Jitter due to bit-stuffing

The CAN protocol uses "Non Return to Zero" (NRZ) coding for bit representation. Under such a scheme, drift in the receiver's clock can occur when a long sequence of identical bits has been transmitted. Such a drift might, in turn, result in message corruption.

To avoid the possibility of such a scenario, the CAN communication protocol (at the physical level) employs a bit-stuffing mechanism which operates as follows: after five consecutive identical bits have been transmitted in a given frame, the sending node adds an additional bit, of the opposite polarity. All receiving nodes remove the 'inserted' bits to recover the original data (Farsi and Barbosa, 2000). Whilst providing an effective mechanism for clock synchronization in the CAN hardware, the bit-stuffing mechanism causes the frame length to become a complex function of the data contents.

It is useful to understand the level of message variation that this process may induce. When using (for example) 8-byte data and standard CAN identifiers, the minimum message length will be 111 bits (without bit stuffing) and the maximum message length will be 135 bits (with the worst-case level of bit stuffing): see Nolte et al., 2003 for details. At the maximum CAN baud rate (1 Mbit/sec), this translates to a possible variation in message lengths of 24 μs.

These variations in message transmission times can have important implications in any real-time systems in which it is important to be able to predict event timing at the microsecond level. For example, in systems using TTC-SC1 and TTC-SC2, variations in the duration of "Tick" messages can have a significant impact on the levels of task jitter in the Slave nodes (see Nahas and Pont, 2004; Nahas et al., 2005).

This process is illustrated in Figure B-6.

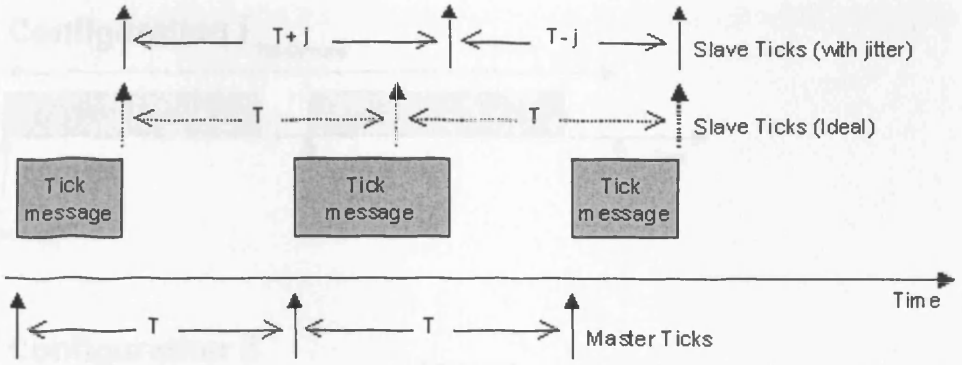

Figure B-6 Impact of frame length on the timing of Slave Ticks in TTC-SC1 and TTC-SC2.

5. TTC-SC3 algorithm

To resolve some of the shortcomings of the TTC-SC1 and TTC-SC2 algorithms, we developed TTC-SC3. An overview of this new protocol is presented in this section.

5.1 More than one Slave can reply in a Tick interval

In the TTC-SC3 algorithm, more than one Slave is allowed to reply within one Tick interval. Each time a Tick message is sent from the Master, an ID is also sent within the message (similar to TTC-SC1 and TTC-SC2). However, with TTC-SC3, it is possible to have more than one Slave reply to each ID. In this case, we let the CAN controller handle any message collisions. The Master node then checks that the appropriate Slaves have replied to a designated Tick ID before transmitting the next Tick message.

For example, let us assume a four-node system is to be implemented on a single CAN bus. Figure B-7 shows the typical message exchange on the CAN network.

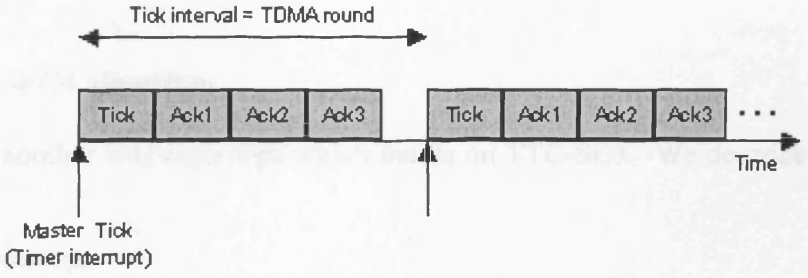

Figure B-7 Tick and Acknowledgements for the TTC-SC3 algorithm.

As an example of a more complicated configuration, suppose that we have a system with N Slaves, it is possible that all N Slaves reply within one Tick interval, or M Slaves reply within the first Tick interval and N-M Slaves reply in the second Tick interval; where M<N. In the latter instance, the TDMA round is extended across two tick intervals. The algorithm can be reconfigured such that the TDMA round is extended across more Tick intervals. The example in Figure B-8 illustrates two possible examples of how a seven-node system can be configured.

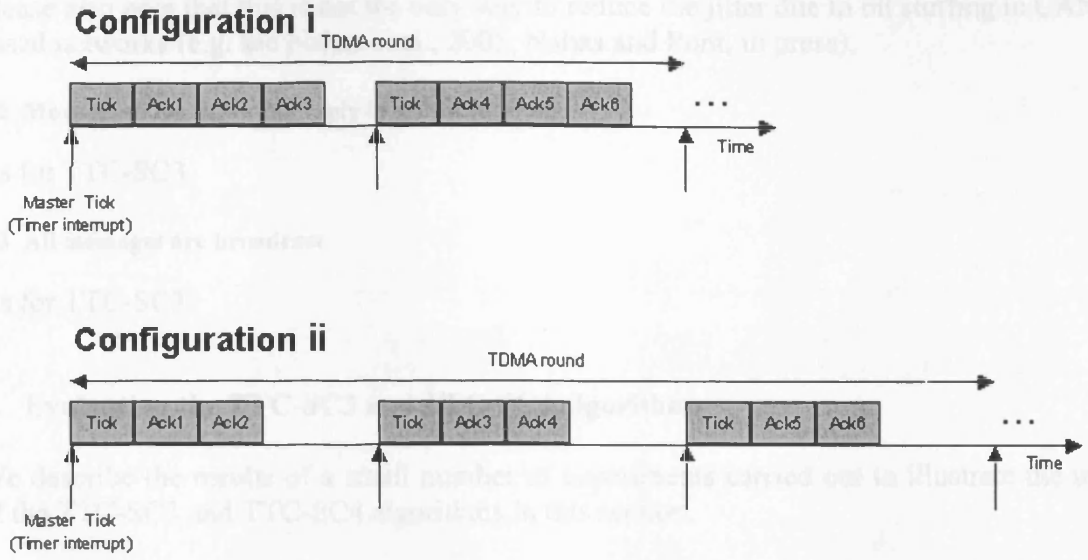

Figure B-8 Two possible TDMA configurations using the TTC-SC3 algorithm for a seven-node system.

5.2 All messages are broadcast

In the TTC-SC3 algorithm, all messages sent from the Slave nodes are broadcasted to all nodes (including the other Slaves).

5.3 Implementation

The broadcasting of Slave messages is made possible (on a CAN network) by assigning to each Slave node a unique CMO for its Ack message.

Please note that - as with TTC-SC1 and TTC-SC2 - these Ack messages should not trigger CAN interrupts.

6. The TTC-SC4 algorithm

TTC-SC4 is another S-C algorithm which builds on TTC-SC3. We describe TTC-SC4 in this section.

6.1 Tick only messages

When using TTC-SC4, the Master node is configured to send out "empty" Tick messages. These messages synchronise the network but – after the initialisation process – will not generally contain data. As such, the Master node simply generates the "heartbeat" of the network, but does no data processing. This approach allows the Tick message to have a fixed data content, which results in a constant CAN Tick message length. Thus, jitter caused by the Tick messages can be reduced.

Please note that, compared to TTC-SC1, TTC-SC2 and TTC-SC3 (where the Master can be involved in the system data processing), the total number of nodes on the system will – usually - increase by one.

Please also note that this is not the only way to reduce the jitter due to bit stuffing in CAN-based networks (e.g. see Nahas et al., 2005, Nahas and Pont, in press).

6.2 More than one Slave can reply in a Tick interval

As for TTC-SC3.

6.3 All messages are broadcast

As for TTC-SC3.

7. Evaluating the TTC-SC3 and TTC-SC4 algorithms

We describe the results of a small number of experiments carried out to illustrate the use of the TTC-SC3 and TTC-SC4 algorithms in this section.

7.1 Reduced failure detection time

TTC-SC3 and TTC-SC4 allows the Master node to quickly obtain Ack messages from the Slaves.

For example, Figure B-9 illustrates an example where Slave 1 suffers a failure as soon as it has transmitted its Ack message. We assume we have all Slaves reply to each Tick message. As a result, the longest possible time that the Master node takes before a failure on the Slave node can be detected is calculated using Equation 2. This duration is slightly less than two Tick intervals (which is significantly less than TTC-SC1 / TTC-SC2 in non-trivial networks).

$$failure\ detection\ time = 2 \times Tick - CANtickmsg \qquad (2)$$

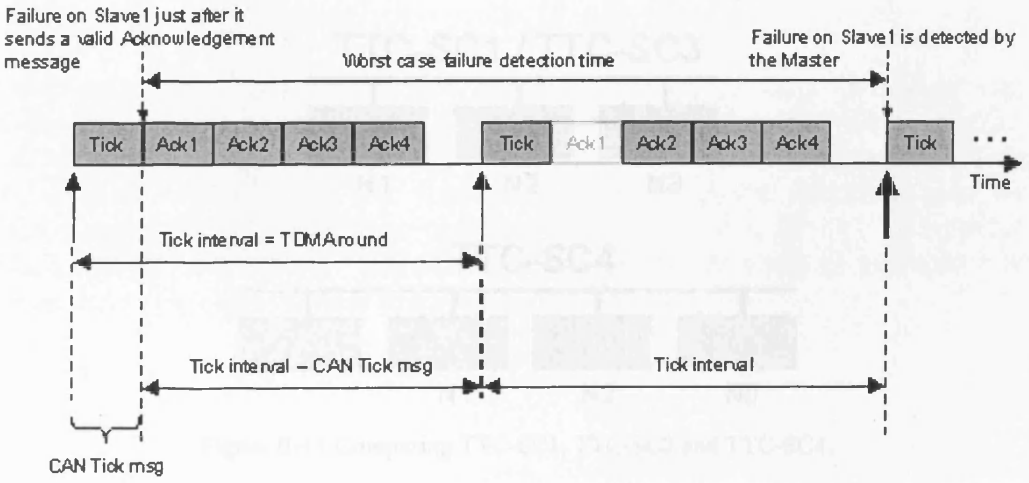

Figure B-9 Calculating the worst-case Slave failure detection time of TTC-SC3/TTC-SC4.

Of course, the precise failure detection time will depend very much on the way the TDMA round was scheduled. If the TDMA round is extended across more than one Tick interval, then Equation 3 is used to calculate the failure detection time. This is illustrated more

clearly in Figure B-10 where Slave 1 suffers a failure as soon as it has transmitted its Ack message.

$$failure\ detection\ time = TDMAround - CANtickmsg + Tick \quad (3)$$

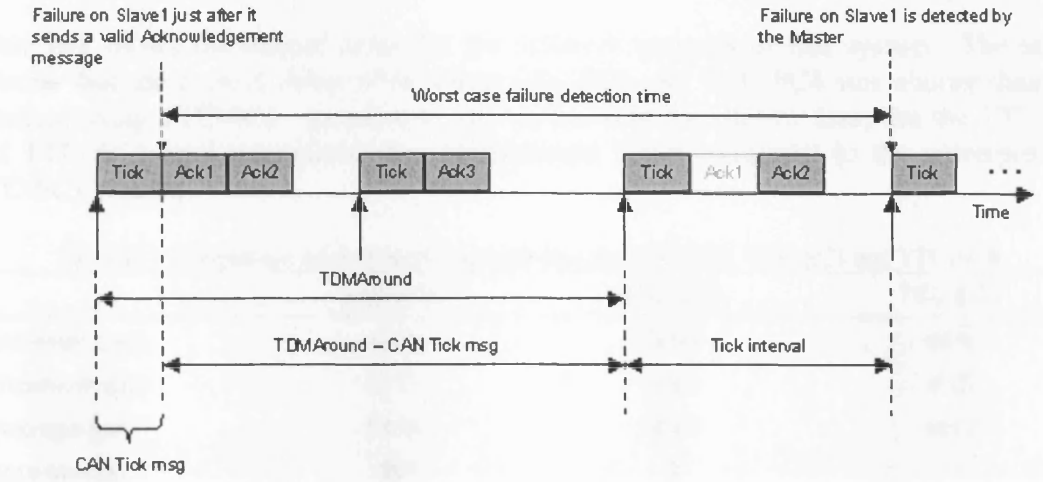

Figure B-10 Calculating the worst-case Slave failure detection time for TTC-SC3 when the TDMA round extends across more than one tick interval.

7.2 Reduced Slave-to-Slave message latency

When compared with TTC-SC1 and TTC-SC2, the latency of message transmission between Slaves is reduced in both TTC-SC3 and TTC-SC4.

To illustrate this, a comparison of the implementation between TTC-SC1, TTC-SC3 and TTC-SC4 was carried out (see Figure B-11)[3].

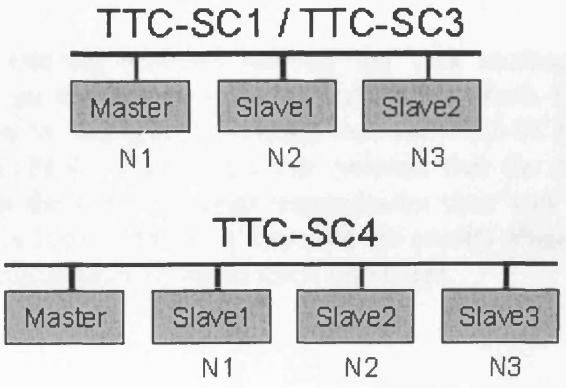

Figure B-11 Comparing TTC-SC1, TTC-SC3 and TTC-SC4.

Referring to Figure B-11, each of the main nodes (N1, N2, N3) executes several periodic tasks which exchange data around the network.

[3] Note that, in both the TTC-SC3 and TTC-SC4 implementations, all Slaves replied to each Tick message.

We made our measurements using a (dummy) control system that involves all three nodes. Specifically, N1 had a control task that issued a periodic request for data from N2 and N3. N2 and N3 each sent data back to N1 as soon as they received the request. N1 then produced a control value. We measured the interval between the data request (on N1) and the completion of the control value calculation on this node.

Table B-1 shows the control delay for the different versions of this system. The results indicate that the control delay when using TTC-SC3 and TTC-SC4 was shorter than that obtained using TTC-SC1. In addition, the variation in the control delay for the TTC-SC3 and TTC-SC4 implementations was insignificant when compared to the corresponding TTC-SC1 results.

Table B-1 Comparison of measured control delays for TTC-SC1, TTC-SC3 and TTC-SC4.

	TTC-SC1	TTC-SC3	TTC-SC4
Minimum (μs)	4013	3003	4008
Maximum (μs)	6012	3003	4022
Average (μs)	5012	3003	4012
Max-Min(μs)	1999	0	14
Std. Deviation	816	0	3

7.3 Jitter due to bit-stuffing

With the TTC-SC3 algorithm, the level of jitter due to bit stuffing remains the same as that obtained using TTC-SC1 and TTC-SC2.

To illustrate the reduction in jitter obtained with TTC-SC4, two versions of a three-node system (each with 1 Master and 2 Slaves) were implemented using TTC-SC3 and TTC-SC4. A CAN baudrate of 500kbits/sec was used in each system. For TTC-SC3, the content of the Tick data messages were periodically rotated among three different values (the Tick messages were "empty" when using TTC-SC4).

Measurements of the interval between sending the Tick message on the Master and receiving the message on the Slaves were carried out (for both the Slave nodes). The results obtained (shown in Table B-2) indicate that the TTC-SC3 algorithm had higher jitter compared to TTC-SC4. The results also indicate that the maximum jitter for the TTC-SC4 algorithm on the CAN message transmission time was +/- 1 bit time (at 500 kbits/sec, one bit time is 2μs). This is in line with the results obtained previously (Nahas and Pont, 2004) for minimal jitter levels in CAN messages.

Table B-2 Message transmission times for TTC-SC3 and TTC-SC4.

	TTC-SC3		TTC-SC4	
	Slave1	**Slave2**	**Slave1**	**Slave2**
Minimum (μs)	186	186	122	122
Maximum (μs)	200	200	126	126
Average (μs)	192	192	124	124
Max-Min (μs)	14	14	4	4
Std. Deviation	5	5	1	1

8. Discussion

Although we have shown that TTC-SC3 and TTC-SC4 algorithms have several benefits (when compared to TTC-SC1 and TTC-SC2), there are also some drawbacks. We consider these here.

8.1 Number of network nodes

Using the TTC-SC4 algorithm, the total number of nodes required in each network will be increased by one. This is because a separate Master node is required to function as the network synchroniser. This will obviously add to the system cost.

8.2 Tick interval

In most cases, TTC-SC3 and TTC-SC4 require all the Slaves to reply within one Tick interval. As such, the following relationship must hold:

Tick interval > Time take for all Ack messages to be received by the Master

That is the Tick interval of the system is related to the number of Slaves, and the size of each of the Slave's Ack messages. This may be a significant drawback in networks requiring a low Tick interval.

8.3 Portability

The TTC-SC3 and TTC-SC4 algorithms cannot be easily ported to other network protocols, such as RS485. This is due to the fact that the algorithms require more than one Slave to reply to each Tick message. Most CAN controllers can deal with this because they can handle message conflicts and support multiple receive buffers. By contrast, RS485 has one receive buffer and no direct support (in hardware) for handling message conflicts.

Please note:

- Some CAN controllers (such as MCP2510) do not have 15 message receive buffers. Overall, this reduces the portability of the TTC-SC3 and TTC-SC4 algorithms when compared with TTC-SC1 and TTC-SC2.
- The number of nodes connected to the CAN bus will depend on the number of message objects supported by the CAN hardware (in most cases, this will be up to 15: see, for example, Siemens, 1996). If more than 15 nodes need to be connected, a

second CAN controller can be used. Many microcontrollers now have two or more on-chip CAN controllers and can support such requirements. However, such an arrangement further reduces portability (and further adds to costs).

8.4 Babbling Slaves

TTC-SC3 and TTC-SC4 will – typically - rely on all the Slaves to send an Acknowledgement back to the Master within one Tick interval. If one of the Slaves have a "babbling idiot" problem (Kopetz, 1998) or there is constant message retransmission from one of the Slaves, then lower priority CAN messages from other Slaves will not have access to the network. This will cause the Master to "think" that the Slaves with the lower priority CAN messages are faulty when in-fact it is only a single node that is causing the problem.[4]

To reduce the impact of this problem, the CAN controller can be configured to disable its automatic message retransmission. However, this is not a complete solution, and the feature is only available on certain CAN implementations (such as that used in the Infineon XC167).

9. Conclusions

This study has investigated the use of shared-clock (S-C) algorithms with CAN-based systems. Specifically, we have looked at two new S-C algorithms (TTC-SC3 and TTC-SC4) which are – when compared with TTC-SC1 and TTC-SC2 - intended to reduce Slave-to-Slave message transmission times, reduce failure detection times and (in the case of TTC-SC4) reduce task jitter.

While no single algorithm is ever likely to provide a perfect solution to all networking problems, the results and discussion presented here suggest that TTC-SC3 and TTC-SC4 are very useful additions to the range of S-C algorithms.

Acknowledgements – This work is supported by an ORS award (to DA) from the UK Government (Department for Education and Skills), and by Pi Technology. Work on this paper was completed while MJP was on Study Leave from the University of Leicester.

References

Bosch, R.G., 1991. CAN specification version 2.0, Robert Bosch GmbH, Postfach 50, D-7000 Stuttgart 1, Germany.

Farsi, M. and Barbosa, M., 2000. CANopen implementations: Applications to industrial networks. Research Studies Press Ltd.

Fredriksson, L.B., 1994. Controller Area Networks and the protocol CAN for machine control systems. Mechatronics, 4(2): 159-192.

Infineon, 2004. Connecting C166 and C500 microcontroller to CAN, Infineon Technologies.

Kopetz, H., 1998. A comparison of CAN and TTP, 15th IFAC Workshop on Distributed Computer Control Systems, Como, Italy.

Leen, G. and Heffernan, D., 2002. TTCAN: A new time-triggered Controller Area Network. Microprocessors and Microsystems, 26(2): 77-94.

[4] Although TTC-SC1 and TTC-SC2 also suffer from problems caused by "babbling idiot" failures, the impact is more severe (and harder to guard against) in TTC-SC3 and TTC-SC4, due to the smaller error margins.

Misbahuddin, S. and Al-Holou, N., 2003. Efficient data communication techniques for Controller Area Network (CAN) protocol, ACS/IEEE International Conference on Computer Systems and Applications, Tunis, Tunisia.
Nahas, M. and Pont, M.J., 2004. Reducing task jitter in shared-clock embedded systems using CAN. In: A. Koelmans, A. Bystrov and M.J. Pont (Editors), Proceedings of the UK Embedded Forum, Birmingham, UK.
Nahas, M., Short, M.J. and Pont, M.J., 2005. The impact of bit stuffing on the real-time performance of a distributed control system, 10th international CAN Conference, Rome, Italy.
Nolte, T., Hansson, H., Norström, C. and Punnekkat, S., 2003. Using bit-stuffing distributions in CAN analysis, IEEE/IEE Real-Time EMbedded Systems Workshop, (Satellite of the IEEE Real-Time Systems Symposium) London.
Pazul, K., 1999. Controller Area Network (CAN) basics, Microchip Technology Inc.
Philips, 1996. PCA82C250/251 CAN transceiver, Philips Semiconductors.
Philips, 2004. SJA1000 stand-alone CAN controller.
Pont, M.J., 2001. Patterns for time-triggered embedded systems. Addison Wesley.
Pont, M.J., 2003. Supporting the development of time-triggered co-operatively scheduled (TTCS) embedded software using design patterns. Informatica, 27(1): 81-88.
Pont, M.J. and Banner, M.P., 2004. Designing embedded systems using patterns: A case study. Journal of Systems and Software, 71(3): 201-213.
Sandfridson, M., 2000. Timing problem in distributed real-time computer control problems. ISSN 1400-1179, Mechatronics Lab, Department of Machine Design, Royal Institute of Technology, KTH, Stockholm, Sweden.
Sevillano, J.L., Pascual, A., Jimenez, G. and Civit-Balcells, 1998. Analysis of channel utilization for Controller Area Networks. Computer Communications, 21(16): 1446-1451.
Siemens, 1996. C167 derivatives - User's manual, Version 2.0.
Siemens, 1997. Proceedings of the European Pattern Languages of Programming Conference.
Thomesse, J.P., 1998. A review of the fieldbuses. Annual Reviews in Control, 22: 35-45.
Tindell, K. and Burns, A., 1994. Guaranteed message latencies for distributed safety-critical hard real-time control networks. YCS 229, Real-Time Systems Research Group, University of York.
Zuberi, K.M. and Shin, K.G., 1995. Non-preemptive scheduling of messages on Controller Area Network for real-time control applications, Proceedings of the First IEEE Real-Time Technology and Applications Symposium, Chicago, USA, pp. 240-249.

Appendix-C A 'hardware-in-the-loop' testbed representing the operation of a cruise-control system in a passenger car

This appendix includes a copy of the Ayavoo et al. (2005c) paper on the detailed implementation of an automotive cruise-control system testbed.

Abstract. The developer of a modern embedded system faces a bewildering range of design options. One way in which the impact of different design choices can be explored in a rigorous and controlled manner is through the use of appropriate hardware-in-the loop (HIL) simulator. HIL simulators – unlike software-only equivalents - allow studies to be carried out in real time, with real signals being measured. In this paper, we describe a HIL testbed that represents an automotive cruise-control system (CCS). A case study is used to illustrate how this testbed may be used to compare the different implementation options for single-processor and multi-processor system designs.

1. Introduction

The developer of a modern embedded system faces a bewildering range of design options. For example, the designer of a modern passenger car may need to choose between the use of one (or more) network protocols based on CAN (Rajnak and Ramnerfors, 2002), TTCAN (Hartwich *et al.*, 2002), LIN (Specks and Rajnak, 2002), FlexRay or TTP/C (Kopetz, 2001). The resulting network may be connected in, for example, a bus or star topology (Tanenbaum, 1995). The individual processor nodes in the network may use event-triggered (Nissanke, 1997) or time-triggered (Kopetz, 1997) software architectures, or some combination of the two. The clocks associated with these processors may be linked using, for example, shared-clock techniques (Pont, 2001) or synchronisation messages (Hartwich *et al.*, 2000). These individual processors may, for example, be C167 (Siemens, 1996), ARM (ARM, 2001), MPC555 (Bannatyne, 2003) or 8051 (Pont, 2001).

One way in which we can explore the impact of different design choices in a rigorous and controlled manner is through the use of appropriate hardware-in-the loop (HIL) simulators (see for example Hanselmann, 1996; Dynasim, 2003). HIL simulators – unlike software-only equivalents - allow studies to be carried out in real time, with real signals being measured.

The basic setup for the HIL simulator is illustrated in Figure C-1, where the HIL simulation and the embedded system interconnect with each other by exchanging information through the necessary I/Os, such as digital, analogue and serial ports.

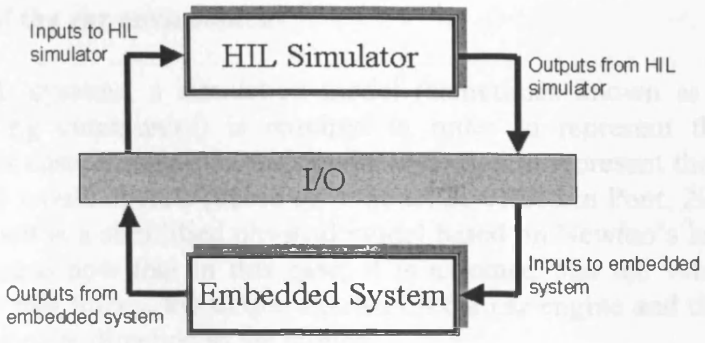

Figure C-1 The HIL approach.

We have recently described a detailed HIL simulation of an adaptive cruise-control system (ACCS) for a passenger car (Short *et al.*, 2004b; Short *et al.*, 2004c; Short *et al.*, 2004a; Short and Pont, in press), and shown how this can be employed to assess new network protocols (Nahas *et al.*, 2005).

The complexity of the full ACCS simulation ensures accurate results at the cost of system complexity. In some cases, it can be useful to be able to eliminate inappropriate design options more quickly through this use of a less detailed HIL simulation. To this end a simple (non-adaptive) cruise-control design was developed. The design, implementation and evaluation of this simple simulator is described in detail in this paper.

The paper is organised as follows. Section 2 to Section 7 introduce the cruise-control testbed and describe the model and implementation details. A case study that employs the testbed is then presented in Section 8. Our conclusions are presented in Section 10.

2. An overview of the CCS testbed

An automotive cruise-control system (CCS) is intended to provide the driver with an option of maintaining the vehicle at a desired speed without further intervention, by controlling the throttle (accelerator) setting. Such a driver assistance system can reduce the strain on the driver especially while travelling on long journeys.

Such a CCS will typically have the following features:
1) An ON / OFF button to enable / disable the system.
2) An interface through which the driver can change the set speed while cruising.
3) Switches on the accelerator and brake pedals that can be used to disengage the CCS and return control to the driver.

For the purpose of our study, the specification of the CCS was simplified such that the vehicle was assumed to be always in "cruise" mode. While in cruise mode, a "speed dial" was available to allow the driver to dynamically change the car speed. The control process ensures that the vehicle would travel at the desired set speed.

3. The design of the car environment

As with any HIL systems, a simulation model (sometimes known as plant within the control engineering community) is required in order to represent the system to be controlled. In this case, a computational model was used to represent the car environment in which the CCS would operate (based on a model described in Pont, 2001). The core of the car environment is a simplified physical model based on Newton's law of motion (see Figure C-2). Please note that in this case, it is assumed that the vehicle is under the influence of only two forces, the torque exerted on the car engine and the frictional force that acts in the opposite direction to the motion.

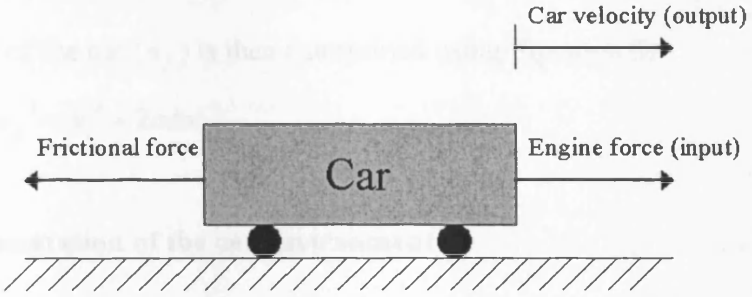

Figure C-2 The car environment for the CCS.

To model this mathematically, the summation of all the forces acting on the car is calculated, beginning with Newton's Second Law of Motion. The terms listed below are used in the equations that follow in this section:

A	Acceleration
v_i	Initial speed
v_f	Final speed
M	Mass of car
Δx	Displacement
Θ	Throttle setting
Fr	Frictional coefficient
T	Engine torque

The frictional force and the engine force of the car are incorporated into Equation 1.

$$EngineForce - FrictionalForce = ma \tag{1}$$

The frictional force is a function of the velocity of the car, whereas the engine force is a product of the throttle setting and the engine torque. The engine torque is assumed to be constant over the speed range. The following model (Equation 2) is thus produced.

$$\theta\tau - v_i Fr = ma \tag{2}$$

This model is then used to determine the output of the car environment, which is the final velocity of the car (v_f). Solving for a, the instantaneous acceleration of the vehicle is first calculated (Equation 3).

$$a = (\theta\tau - v_i Fr)/m \tag{3}$$

Once this acceleration has been obtained, the distance travelled by the car (Δx) is solved using Equation 4.

$$\Delta x = v_i t + \frac{1}{2}at^2 \tag{4}$$

The final speed of the car (v_f) is then determined using Equation 5.

$$v_f^2 = v_i^2 + 2a\Delta x \tag{5}$$

4. The implementation of the car environment

The car environment was implemented using a time-triggered co-operative scheduling architecture on a basic desktop PC (Intel Pentium II 300 MHz processor). The advantages of using PC hardware for such studies is described in detail elsewhere (see Pont *et al.*, 2003).

Four main tasks were implemented as shown in Table C-1. The source code was written and compiled using the Open Watcom C compiler[5].

Table C-1 The car model task structure.

Task Names	Task Description	Task Period (in ms)
Car Dynamics Update	Updates the car dynamics (speed) based on the input throttle position	5
Car Display Update	Displays the speed of the car and the throttle position on the monitor	100
PRM Update	Sends out the speed of the car as a train of pulses	1
Write To File	Records the speed and the throttle position of the car in a text file	1000

We wanted to keep the cost of this simulator as low as possible (so that it can be widely used). In order to access the PC hardware level, an operating system (OS) was required. DOS (Disk Operating System) was chosen because it offers the required flexibility at low cost[6]. DOS in-turn control the BIOS (Basic Input Output System) of the PC to access the PC hardware level (Figure C-3).

[5] The Watcom compiler can be downloaded (without charge) here: http://www.openwatcom.org/.
[6] Free versions of DOS are available. See for example http://www.handyarchive.com/free/dos/

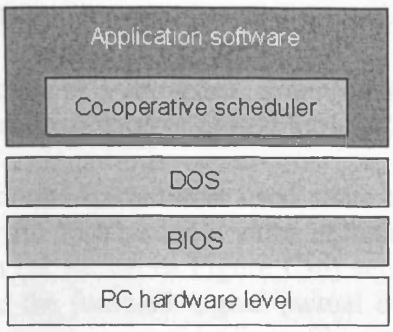

Figure C-3 The operating layers for a PC hardware implementation.

To interface the real world, a low-cost (but effective) option is to use the PC's parallel port. The parallel port has three registers: the data register, status register and control register (Messmer, 2002). In our version of the CCS, the port's data register (LPT1, 0x378) was used to store an 8-bit throttle position as the input signal to the car environment. The output signal from the car environment is the current speed of the car, represented as a train of pulses at the automatic line feed (ALF) pin of the port's control register (LPT1, 0x37A).

The connections are illustrated in Figure C-4.

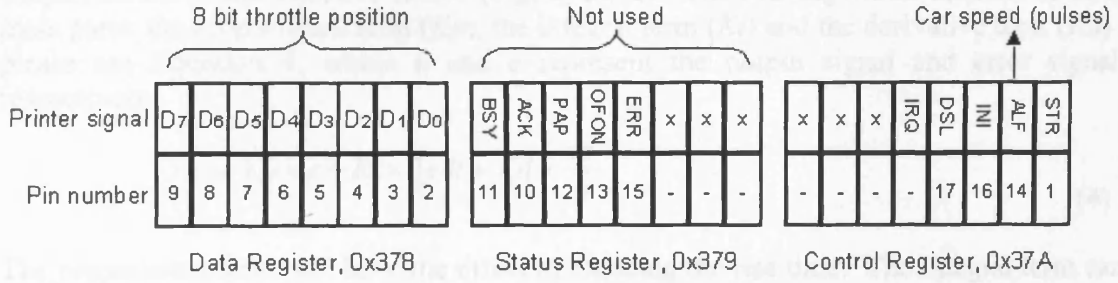

Figure C-4 Connections on the PC's parallel port.

Figure C-5 shows a screenshot of the car environment model running on a desktop PC.

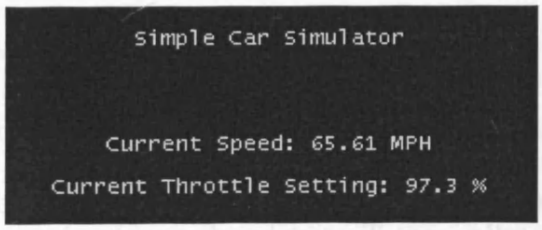

Figure C-5 A screenshot of the CCS system.

The source code for the car environment model is discussed in Appendix A. The complete source code is available from the Embedded Systems Laboratory website[7].

[7] http://www.le.ac.uk/eg/embedded/SimpleCCS.htm

5. The design of the controller

To control the velocity of the car at a set speed, a control algorithm was required. Two basic controllers can be chosen: open loop or closed loop.

In this case, a closed-loop control system was used since the output value from the car environment has an impact on the process input value to maintain the desired output value. A closed loop control system (as shown in Figure C-6) sends the difference of the input signal (the desired value) and the feedback signal (actual output value) to the controller. The controller's job is to reduce this error (ideally to 0). To achieve this, a wide range of control algorithms are available (see, for example, Dutton *et al.*, 1997; Dorf and Bishop, 2000).

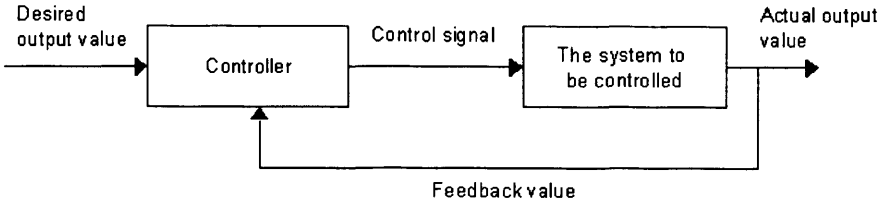

Figure C-6 A closed-loop or feedback control system (Pont, 2001).

A "Proportional Integral Differential" (PID) controller was used in the CCS as it is a simple, common and effective choice (Ogata, 2002). The PID algorithm consists of three main parts: the proportional term (*Kp*), the integral term (*Ki*) and the derivative term (*Kd*) : please see Equation 4, where u and e represent the output signal and error signal, respectively.

$$u = Kp \times e + Ki \times \int e dt + Kd \times \frac{de}{dt} \tag{4}$$

The proportional term will have the effect of reducing the rise time. The integral term can eliminate the steady-state error, but it may make the transient response worse. The derivative term can be used to add "damping" to the response, in order to reduce the signal overshoot (Franklin *et al.*, 1998).

6. Implementation of the controller

A typical control system can be divided into three main sections: sampler, control algorithm and actuator. In the first section, data are sampled from the environment model. In the second section these data are processed using an appropriate control algorithm. In the third section, an output signal is produced that will (generally) alter the system state.

In a single-processor system, all three functions will be carried out on the same node. In a distributed environment, these functions may be carried out on up to three nodes, linked by an appropriate network protocol. For example, a two-node design might carry out the sampling operations on Node 1, and the control and actuation operations on Node 2 (see, for example Lonn and Axelsson, 1999; El-khoury and Törngren, 2001).

Figure C-7 and Figure C-8 illustrate the implementation of a one-node and two-node CCS respectively[8].

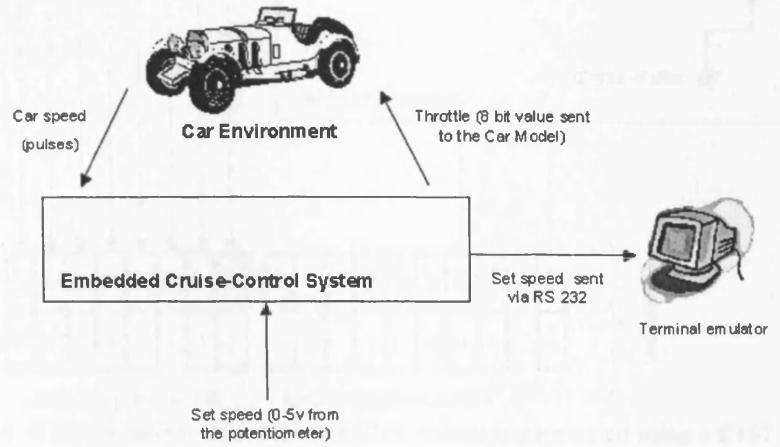

Figure C-7 One-node CCS.

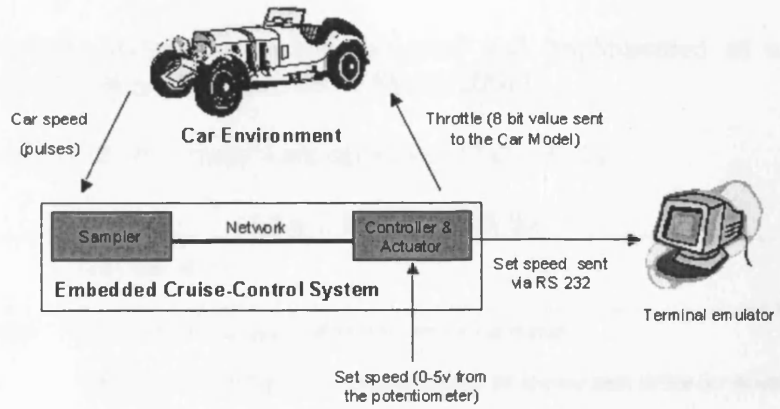

Figure C-8 Two-node CCS.

In both the one-node and two-node systems, the input to the CCS is the car speed (represented as a train of pulses from the car environment) and the desired set speed comes from a potentiometer. The output from the CCS is an 8-bit throttle position that is sent to the car environment and (to support the simulation) the desired set speed value that is sent via an RS232 link to a PC or similar terminal.

Figure C-9 shows an example of the wiring involved for a one-node CCS on a C167 microcontroller.

[8] Please note that the system could also be expanded to have more than two nodes to incorporate various design options such as bus guardians, back-up nodes and redundancy. These options are not considered here.

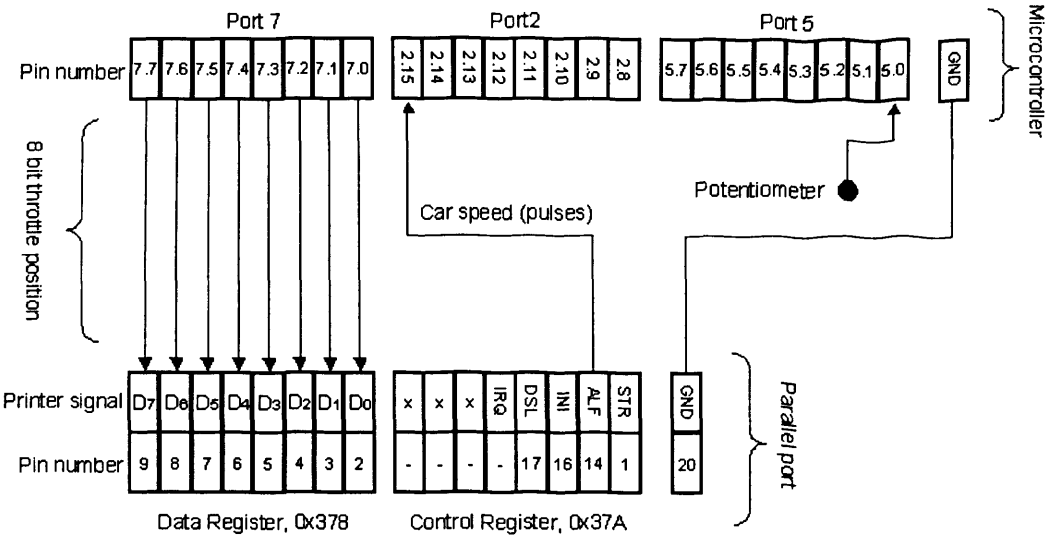

Figure C-9 Wiring example for a one-node CCS system implemented using a C167 processor.

7. The CCS tasks

Embedded software systems are often designed and implemented as a collection of communicating tasks (e.g. Nissanke, 1997; Shaw, 2001).

To implement the CCS, five tasks were employed (Table C-2).

Table C-2 The CCS task list.

Task Names	Task Description	Task Period (in ms)
Compute Car Speed	Computes the car speed obtained from the car model	50
Compute Throttle	Calculates and sends the required throttle to be applied back to the car model	50
Get Ref Speed	Gets the desired speed from the driver	1000
PC Link Update	Sends a character to the serial port	10
Display Ref Speed	Updates the string that displays the desired car speed	1000

Each task is described in the subsections that follow.

7.1 Task: Compute Car Speed

This task acts as the signal sampler. The signal – in this case, the speed of the car – is represented as a train of pulses. To obtain the correct representation of the car speed, a hardware pulse counter is used to store the number of pulses that has arrived. This value is then filtered (using software) to remove any noise that may be present in the signal. The filtered value will then to be scaled to represent the current speed of the car.

A partial code listing for this task is show in Listing C-1.

Please note that the processor chosen for this application must have at least two hardware timers – one for the periodic timer and one for the hardware counter.

```
void Sens_Compute_Speed(void)
   {
   tWord raw_speed;

   raw_speed = Get_Raw_Speed();

   Scaled_speed_G = ((float)(FILTER_COEFF * Old_speed_G) + (float)
      ((1 - FILTER_COEFF) * (raw_speed * SCALING_FAC)));

   Old_speed_G = Scaled_speed_G;
   }
```

Listing C-1 An example of the compute car speed task.

7.2 Task: Compute Throttle

This task functions as the controller and actuator. The PID algorithm was implemented in this function, and "anti-windup" was included[9].

Once the necessary control value has been calculated, this value is then scaled to an 8-bit throttle position (in the range of 0-255). The partial code listing is illustrated in Listing C-2.

```
void Compute_Throttle(void)
   {
   unsigned char pc_throttle  = 0;
   static float  throttle     = 0;
   float         car_speed    = 0;
   float         set_speed    = 0;
   float         speed_error  = 0;

   car_speed = Scaled_speed_G;
   set_speed = Ref_Speed_G;

   speed_error = set_speed - car_speed;

   throttle = PID_Control(speed_error, throttle);

   pc_throttle = (unsigned char)(throttle * 255);

   Throttle_Port = pc_throttle;
   }
```

Listing C-2 An example of the compute throttle task.

7.3 Task: Get Ref Speed

The purpose of this task is to obtain the reference or desired set speed that the driver may want the car to travel at. The reference speed of the car is obtained using a 0-5 volts potentiometer. An on-board analogue to digital converter (ADC) is used to capture the signal. The signal is then scaled to the reference speed within the required range. The code listing for the implementation on a C167 microcontroller is illustrated in Listing C-3.

---

[9] Anti-windup protection ensures that – when the throttle is at a maximum or minimum value – the error value does not continue to accumulate. For further details, see Aström, 2002.

```
void Act_Get_Ref_Speed(void)
   {
   tWord Time_out_loop = 1;
   tWord AD_result = 0;

   ADST = 1;

   while ((ADBSY == 1) && (Time_out_loop != 0))
      {
      Time_out_loop++;
      }

   if (!Time_out_loop)
      {
      AD_result_G = 0;
      }
   else
      {
      AD_result_G = ADDAT;
      }
   Ref_Speed_G = (tByte)(( AD_result_G / 1023.0f) *
                  MAX_CAR_SPEED);
   }
```

Listing C-3 An example of the get ref speed task.

7.4 Task: Display Ref Speed

This task uses the task "PC Link Update" to display the required operating speed of the car.

7.5 Task: PC Link Update

The purpose of this task is to display information on a terminal emulator (for example HyperTerminal running on a PC) by means of an RS232-based serial connection from the processor node on which this task is running.

8. Case study

To obtain some preliminary results from the CCS testbed, we compared the control performance of one-node and two-node CCS implementations. In each case the CCS nodes were implemented using an Infineon 16-bit microcontroller (Phytec C167CR development board): such devices are widely used in the automotive sector (Siemens, 1996).

8.1 Implementation of one-node CCS

The description of the one node CCS was given in Section 6 and 7. The tasks were implemented using a time-triggered co-operative scheduler (see Pont, 2001). A picture of the single node setup is shown in Figure C-10. The source code for a single node implementation on the C167 is discussed in Appendix B.

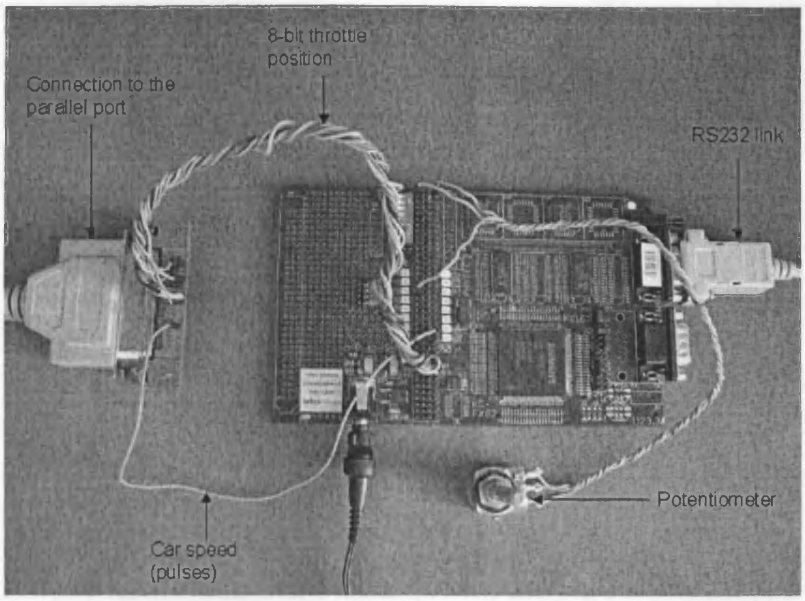

Figure C-10 Implementation of a one-node CCS on the C167.

8.2 Implementation of a two-node CCS

In the second design, the CCS was designed to operate as a distributed system using two nodes: a sampler node and a controller / actuator (CA) node (Figure C-11). The sampler node was used to calculate the vehicle speed. The calculated car speed was then sent over a network to the CA node. On the CA node, the PID algorithm was used to calculate the required throttle position. The CA node was also responsible for obtaining the required "set speed" value (from the driver). The nodes were linked using a CAN bus running at 333.3 kbits/s.

In this particular implementation, a "Time-Time-Time" system was employed. As such, the scheduling on both nodes was time-triggered and the network protocol was also time-triggered, using shared-clock scheduling (Pont, 2001). On both nodes, the tasks were scheduled to execute periodically. A "tick" message was sent from the sensor node at the beginning of every sensor node "tick". This message was used to synchronize the CA node. The sensor status and the car speed data were also included in this message. An acknowledgement message from the CA node was then sent back to the sensor node.

The source codes for the two-node implementation on the C167 boards is discussed further in Appendix C.

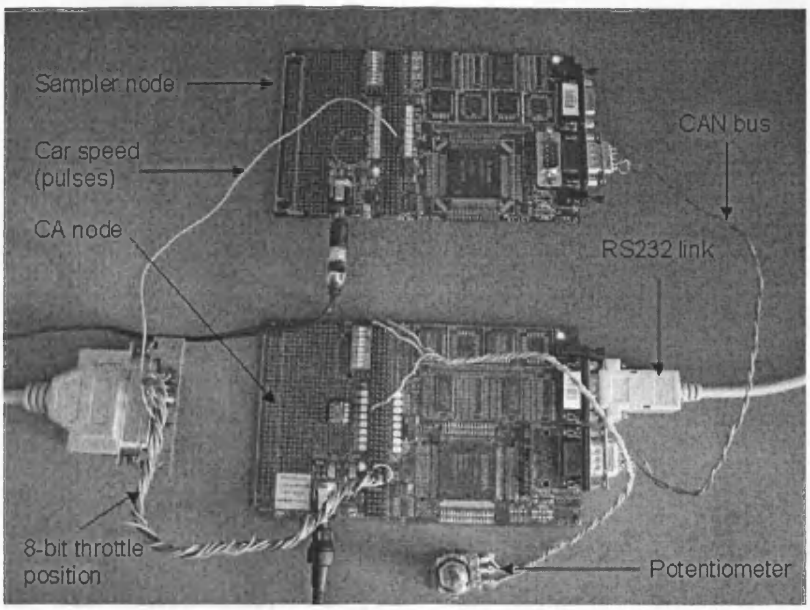

Figure C-11. Implementation of a two-node CCS on the C167.

9. Results

The results for the two different implementation options were compared at four different set speed values (60mph, 100mph, 170mph and 120 mph). Figure C-12 shows the car speed for the two different implementations. Although both the implementation options were very similar, there was some slight differences. The results show that the single node implementation could maintain the car speed more accurately to the desired speed compared to the two-node system.

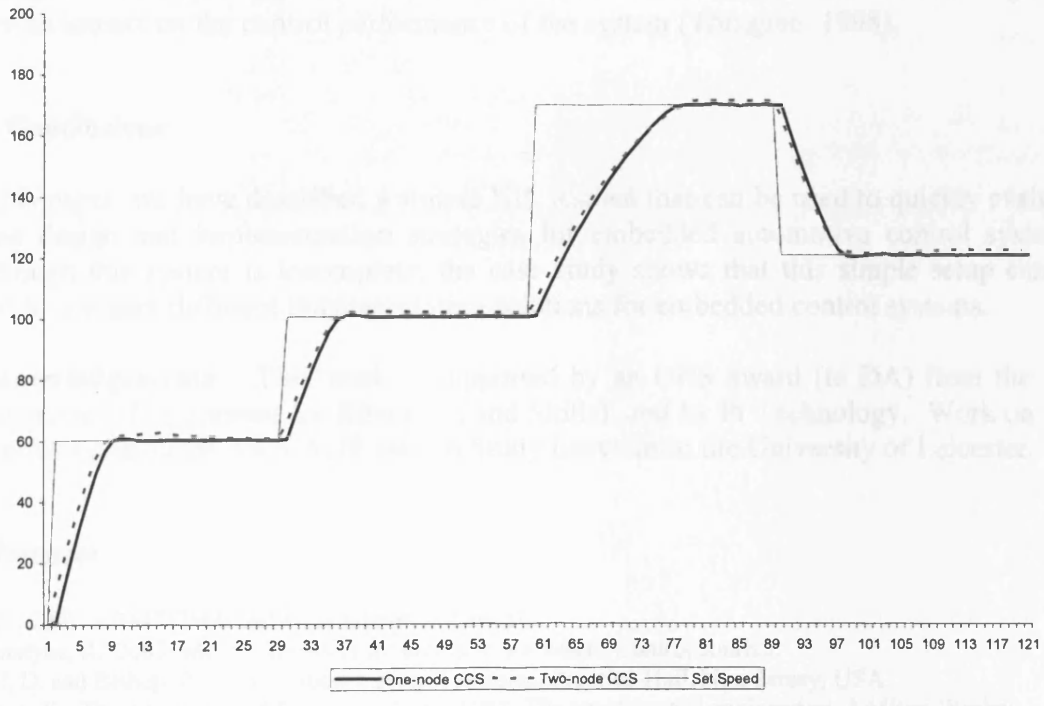

Figure C-12. The performance (speed) of the car for two different implementation options.

Figure C-13 shows the differences in the throttle performance for the different implementations. The results indicate that the responsiveness of the controller to variations in the output signal for the single node system was – again – slightly better than the two-node implementation.

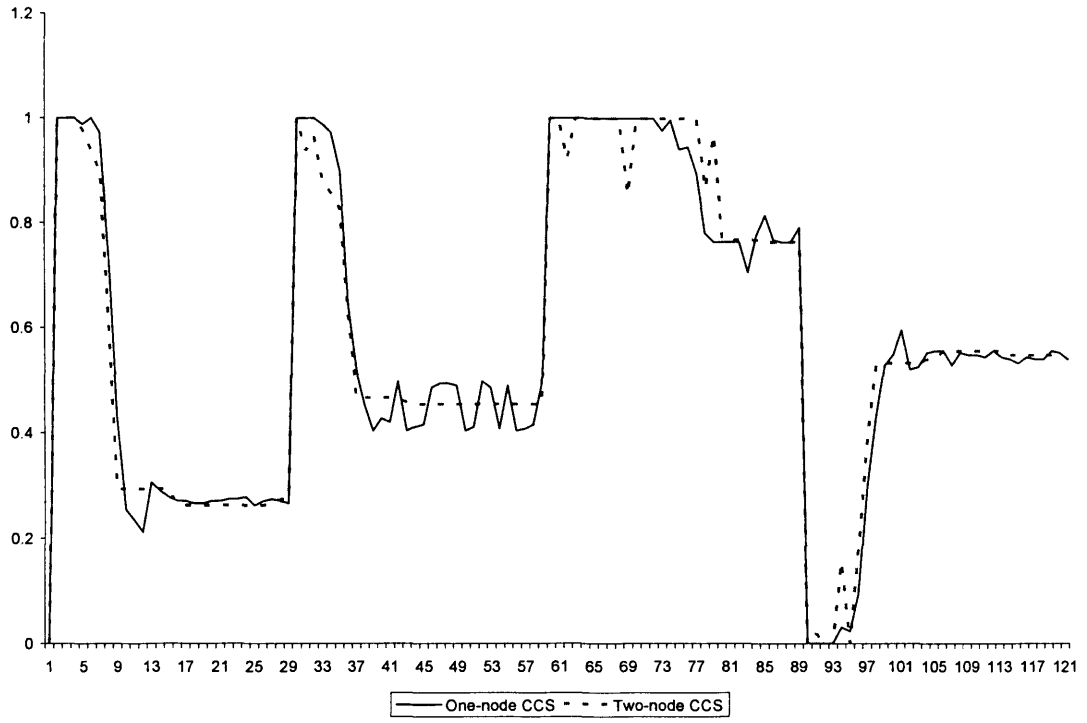

Figure C-13 The performance (throttle) of the car for two different implementation options.

These differences in the results can be attributed to the fact that the distributed system has an additional delay element involved in the network communication. Such delays can have an impact on the control performance of the system (Törngren, 1998).

10. Conclusions

In this paper, we have described a simple HIL testbed that can be used to quickly evaluate some design and implementation strategies for embedded automotive control systems. Although this system is incomplete, the case study shows that this simple setup can be used to compare different implementation solutions for embedded control systems.

Acknowledgements – This work is supported by an ORS award (to DA) from the UK Government (Department for Education and Skills), and by Pi Technology. Work on this paper was completed while MJP was on Study Leave from the University of Leicester.

References

ARM, 2001. ARM7TDMI technical reference manual.
Bannatyne, R., 2003. Microcontrollers for automobiles. Micro Control Journal.
Dorf, D. and Bishop, R., 2001. Modern control systems. Prentice-Hall, New Jersey, USA.
Dutton, K., Thompson, S. and Barraclough, B., 1997. The art of control engineering. Addison Wesley.

Dynasim, 2003. Hardware-in-the-loop simulation of physically based automotive model with Dymola, Dynasim, Lund, Sweden.
El-khoury, J. and Törngren, M., 2001. Towards a toolset for architectural design of distributed real-time control systems, IEEE Real-Time Symposium. IEEE, London, England.
Franklin, G.F., Powell, J.D. and Workman, M.L., 1998. Digital control of dynamic systems. Addison-Wesley.
Hanselmann, H., 1996. Hardware-in-the-loop simulation testing and its integration into a CACSD toolset, The IEEE International Symposium on Computer-Aided Control System Design, Michigan, USA.
Hartwich, F., Muller, B., Fuhrer, T., Hugel, R. and GmbH, R.B., 2000. CAN networks with time-triggered communication, 7th international CAN Conference.
Hartwich, F., Muller, B., Fuhrer, T., Hugel, R. and GmbH, R.B., 2002. Timing in the TTCAN network, Proceedings 8th International CAN Conference.
Kopetz, H., 1997. Real-time systems: Design principles for distributed embedded applications. Kluwer Academic.
Kopetz, H., 2001. A comparison of TTP/C and FlexRay. Research Report 10/2001.
Lonn, H. and Axelsson, J., 1999. A comparison of fixed-priority and static cyclic scheduling for distributed automotive control application, The Eleventh Euromicro Conference on Real-Time Systems, York, England.
Messmer, H.P., 2002. The indespensable PC hardware book. Addison-Wesley.
Nahas, M., Short, M.J. and Pont, M.J., 2005. The impact of bit stuffing on the real-time performance of a distributed control system, 10th international CAN Conference, Rome, Italy.
Nissanke, N., 1997. Realtime systems. Prentice Hall.
Ogata, K., 2002. Modern control engineering. Prentice Hall.
Pont, M.J., 2001. Patterns for time-triggered embedded systems. Addison Wesley.
Pont, M.J., Norman, A.J., Mwelwa, C. and Edwards, T., 2003. Prototyping time-triggered embedded systems using PC hardware. In: K. Henney and D. Schutz (Editors), Eighth European Conference on Pattern Languages of Programs (EuroPLoP), Irsee, Germany.
Rajnak, A. and Ramnerfors, M., 2002. The Volcano communication concept, International Congress on Transportation Electronics. Society of Automotive Engineers Inc.
Shaw, A.C., 2001. Real-time systems and software. John Wiley & Sons Inc.
Short, M.J. and Pont, M.J., in press. Hardware in the loop simulation of embedded automotive control systems, IEEE International Conference on Intelligent Transportation Systems 2005, Vienna, Austria.
Short, M.J., Pont, M.J. and Huang, Q., 2004a. Development of a hardware-in-the-loop test facility for automotive ACC implementations. ESL04-03, Embedded Systems Laboratory, University of Leicester.
Short, M.J., Pont, M.J. and Huang, Q., 2004b. Simulation of a vehicle longitudinal dynamics. ESL 04-01, Embedded System Laboratory, University of Leicester.
Short, M.J., Pont, M.J. and Huang, Q., 2004c. Simulation of motorway traffic flows. ESL04-02, Embedded Systems Laboratory, University of Leicester.
Siemens, 1996. C167 derivatives - User's manual, Version 2.0.
Specks, J.W. and Rajnak, A., 2002. LIN - Protocols, development tools, and software interfaces for Local Interconnect Networks in vehicles, 9th International Conference on Electronic Systems for Vehicles.
Tanenbaum, A.S., 1995. Distributed operating systems. Prentice Hall.
Törngren, M., 1998. Fundamentals of implementing real-time control applications in distributed computer systems. Journal of Real-Time Systems, 14: 219-250.

Appendix-D Background on design patterns

This appendix presents some background material on design patterns. Parts of this material also appears in Pont et al. (submitted).

In recent years, some developers have found that design patterns offer a means of achieving effective "design recycling". Current work on patterns was inspired by Christopher Alexander and his colleagues (Alexander *et al.*, 1977; Alexander, 1979). Alexander is an architect who first described what he called "a pattern language" relating various architectural problems (in buildings) to good design solutions.

This concept of descriptive problem-solution mappings was adopted by Ward Cunningham and Kent Becks who used some of Alexander's techniques as the basis for a small "pattern language" intended to provide guidance to novice Smalltalk programmers (Cunningham and Becks, 1987). This work was subsequently built upon by Erich Gamma and colleagues who, in 1995, published an influential book on general-purpose object-oriented software patterns (Gamma *et al.*, 1995).

Over the last decade, the development of pattern-based design techniques has become an important area of research in the software engineering community. Gradually, the focus has shifted from the use, assessment and refinement of individual patterns, to the creation of complete pattern languages, in areas including telecommunications systems (see: Rising, 2001) and systems with hardware constraints (Noble and Weir, 2001).

In 1996, researchers in the ESL began to assemble a collection of patterns to support the development of time-triggered software for embedded systems. The first versions of these patterns were used "in house", primarily for teaching and training purposes. The ESL researchers then began to publish and discuss the next versions of the patterns more widely (Pont, 2000) and not just at pattern workshops but also at more general technical conferences (see for example Pont *et al.*, 1998; Pont, 1999; Pont, 2000). Through this process, a great deal of useful feedback on the project was obtained, and the collection was refined again. The end result was a set of more than seventy patterns, which is referred to as the Patterns for Time-Triggered Embedded Systems (PTTES) collection (Pont, 2001); see Table D-1 for a list of these patterns. It is important to appreciate that all of the PTTES patterns are intended to support the development of software for systems with time-triggered co-operative/time-triggered hybrid architectures.

In summary, the PTTES collection is organised as follows:

- The processor patterns (STANDARD 8051, SMALL 8051 and EXTENDED 8051) allow selection of a processor with performance levels appropriate for the application.
- The oscillator patterns (Crystal Oscillator and CERAMIC RESONATOR) allow an appropriate choice of oscillator type, and oscillator frequency to be made, taking into account system performance (and, hence, task duration), power-supply requirements, and other relevant factors.
- The various Shared-Clock schedulers (SCC SCHEDULER, SCI SCHEDULER (DATA), SCI SCHEDULER (TICK), SCU SCHEDULER (LOCAL), SCU SCHEDULER (RS-232) and SCU SCHEDULER (RS-485)) describe how to schedule tasks on multiple processors with a

time-triggered architecture. Using one of these schedulers as a foundation, the pattern Long Task describes how to migrate longer tasks onto another processor without compromising the basic time-triggered architecture.

- LOOP TIMEOUT and HARDWARE TIMEOUT describe the design of timeout mechanisms that may be used to ensure that tasks complete within their allotted time.
- MULTI-STAGE TASK discusses how to split up a long, infrequently triggered task into a short task that can be called more frequently. PC LINK (RS232) and LCD CHARACTER PANEL both implement this architecture.
- HYBRID SCHEDULER describes a scheduler that has most of the desirable features of the (pure) co-operative scheduler, but also allows a single long (pre-emptive) task to be executed.

Table D-1 This table lists the 72 patterns in the PTTES collection. Table adapted from Mwelwa *et al.* (submitted).

STANDARD 8051	SMALL 8051	EXTENDED 8051
CRYSTAL OSCILLATOR	CERAMIC OSCILLATOR	RC RESET
ROBUST RESET	ON-CHIP MEMORY	OFF-CHIP DATA MEMORY
OFF-CHIP CODE MEMORY	NAKED LED	NAKED LOAD
IC BUFFER	BJT DRIVER	IC DRIVER
MOSFET DRIVER	SSR DRIVER (DC)	EMR DRIVER
SSR DRIVER (AC)	SUPER LOOP	PROJECT HEADER
PORT I/O	PORT HEADER	HARDWARE DELAY
SOFTWARE DELAY	HARDWARE WATCHDOG	CO-OPERATIVE SCHEDULER
HARDWARE TIMEOUT	LOOP TIMEOUT	MULTI-STAGE TASK
MULTI-STATE TASK	HYBRID SCHEDULER	PC LINK (RS232)
SWITCH INTERFACE (SOFTWARE)	SWITCH INTERFACE **(HARDWARE)**	ON-OFF SWITCH
MULTI-STATE SWITCH	KEYPAD INTERFACE	MX LED DISPLAY
LCD CHARACTER PANEL	I^2C PERIPHERAL	SPI PERIPHERAL
SCI SCHEDULER **(TICK)**	SCI SCHEDULER **(DATA)**	SCU SCHEDULER **(LOCAL)**
SCU SCHEDULER **(RS-232)**	SCU SCHEDULER (RS-485)	SCC SCHEDULER
DATA UNION	LONG TASK	DOMINO TASK
HARDWARE PULSE-COUNT	SOFTWARE PULSE-COUNT	HARDWARE PRM
SOFTWARE PRM	ONE-SHOT ADC	ADC PRE-AMP
SEQUENTIAL ADC	A-A FILTER	CURRENT SENSOR
HARDWARE PWM	PWM SMOOTHER	3-LEVEL PWM
SOFTWARE PWM	DAC OUTPUT	DAC SMOOTHER
DAC DRIVER	PID CONTROLLER	255-TICK SCHEDULER
ONE-TASK SCHEDULER	ONE-YEAR SCHEDULER	STABLE SCHEDULER

As an example of a pattern from the PTTES collection, consider HEARTBEAT LED, which is summarised in Figure D-1 and Figure D-2.

HEARTBEAT LED

Context

- You are developing (or maintaining) an embedded application based on a microcontroller or microprocessor.
- You are programming in C (or a similar language).
- Your application has an architecture based on some form of scheduler.

Problem

How can you tell, at a glance, if your system is "alive"?

Design constraints

Many embedded systems have little or no user interface. There is not generally a screen on which you can display error messages or warnings to the user. If you are working on a system prototype, or performing maintenance in the field, how can you tell that the system is "alive" - that it has power and (at least) the scheduler is running?

You could, of course, hook up a debugging link (e.g. a JTAG link), or a simpler serial link (based on RS-232), but this takes time and including suitable ports on your production system may not be practical or cost effective. Often a very simple, low-cost solution is required.

Solution

Every time we implement an embedded system, the first task we include is one that flashes a "heartbeat" LED. Wherever possible, this LED stays with the system, right into production.

We tend to use a 50% duty cycle and a frequency of 0.5 Hz (that is, the LED runs continuously, on for one second, off for one second, and so on) but this is – of course – up to you.

Use of this simple technique provides the following key benefit:

- The development team, the maintenance team and, where appropriate, the users, can tell at a glance that the system has power, and that the scheduler is operating normally.

In addition, during development, there are two less significant (but still useful) side benefits:

- After a little practice, the developer can tell "intuitively" - by watching the LED - whether the scheduler is running at the correct rate: if it is not, it may be that the timers have not been initialized correctly, or that an incorrect crystal frequency has been assumed.
- By adding the "Heartbeat" task to the scheduler array after all other tasks have been included, the developer can tell immediately if the task array is large enough to match the needs of the application (if the array is not large enough, the LED will never flash).

Reliability and safety implications

Use of this simple technique may help to improve system reliability since it provides those developing the system with an indication of its health throughout the development lifecycle.

Hardware requirements

HEARTBEAT LED has minimal hardware requirements. The only requirements are a port pin connected to an appropriate LED (with an appropriate resistor if required).

Cost implications

As noted above, the hardware requirements are very limited. The time taken to implement this pattern is also likely to be minimal. Overall, the costs are very low.

Overall strengths and weaknesses

☺ HEARTBEAT LED provides a simple, low-cost way of determining whether your system is "alive".

☹ Uses a port pin and associated LED hardware.

Figure D-1 A summary of the pattern HEARTBEAT LED. Figure adapted from Mwelwa and Pont (2003).

```
/*-------------------------------------------------------------------*-
   Heartbeat_LED.C

   Simple 'Heartbeat LED' PIE for an Infineon C515C microcontroller.
   If everything is OK, flashes at 0.5 Hz

-*-------------------------------------------------------------------*/
#include "Main.H"
#include "Port.H"
#include "Heartbeat_LED.H"

// ------ Private variable definitions -----------------------------
static bit Heartbeat_led_state_G;

/*-------------------------------------------------------------------*-

  HEARTBEAT_LED_Init()
  Prepare for HEARTBEAT_Update() task.

-*-------------------------------------------------------------------*/
void HEARTBEAT_LED_Init(void)
   {
   Heartbeat_led_state_G = 0;
   }

/*-------------------------------------------------------------------*-

  HEARTBEAT_LED_Update()

  Flashes an LED on a specified port pin.

  Must schedule at twice the required flash rate: thus, for 0.5 Hz
  flash (on for 1 second, off for 1 second) must schedule at 1 Hz.

-*-------------------------------------------------------------------*/
void HEARTBEAT_LED_Update(void)
   {
   // Change the LED from OFF to ON (or vice versa)
   if (Heartbeat_led_state_G == 1)
      {
      Heartbeat_led_state_G = 0;
      Heartbeat_led_pin = 0;
      }
   else
      {
      Heartbeat_led_state_G = 1;
      Heartbeat_led_pin = 1;
      }
   }
/*-------------------------------------------------------------------*-
  ---- END OF FILE ------------------------------------------------
-*-------------------------------------------------------------------*/
```

Figure D-2 A HEARTBEAT LED PIE for the 8051 platform. Figure adapted from Mwelwa and Pont (2003).

As you examine Figure D-1 and Figure D-2, please note it is sometimes assumed that a (software) pattern is simply a code library. As HEARTBEAT LED should help to make clear, this is not the case (in fact, it includes no code at all). Instead, a pattern includes a broad discussion of the problem area, a discussion of the consequences of applying this solution, as well as suggestions about alternative approaches. Of course, code will also be required in many cases: this may be included in an "example" section in the pattern or – in more recent pattern libraries (Pont *et al.*, (submitted 10 August)); - in the form of a set of linked "Pattern Implementation Examples" (PIEs), as illustrated in Figure D-3 .

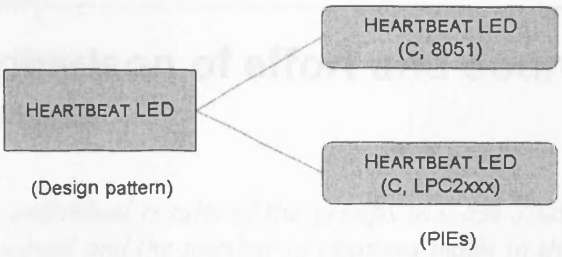

Figure D-3 Link between a pattern and its Pattern Implementation Examples (PIEs). Figure adapted from Mwelwa *et al.* (submitted).

As the name might suggest, a PIE illustrates how a particular pattern can be implemented. This is important (in the field of embedded systems) because there are great differences in system environments, caused by variations in the hardware platform (e.g. 8-bit, 16-bit, 32-bit, 64-bit) and programming language (e.g. Assembly and C). The possible implementations are not sufficiently different to be classified as distinct patterns: however, they do contain useful information.

References

Alexander, C. (1979). "The timeless way of building", Oxford University Press.

Alexander, C., S. Ishikawa, M. Silverstein, M. Jacobson, I. Fiksdahl-King and S. Angel (1977). "A pattern language", Oxford University Press.

Cunningham, W. and K. Becks (1987). "Using pattern languages for Object-Oriented programs". OOPSLA'87 workshop on the Specification and Design for Object-Oriented Programming, Floriday, USA.

Gamma, E., R. Helm, R. Johnson and J. Vlissides (1995). "Design patterns: Elements of reusable Object-Oriented software", Addison-Wesley.

Kurian, S. and M. J. Pont (2005). "Building reliable embedded systems using Abstract Patterns, Patterns, and Pattern Implementation Examples". Proceedings of the 2nd UK Embedded Forum, 20th October. A. Koelmans, A. Bystrov and M. J. Pont. Birmingham, UK.

Noble, J. and C. Weir (2001). "Small Memory Software". Reading, Massachusetts, USA, Addison Wesley.

Pont, M. J. (1999). "Pattern for embedded systems". Invited presentation to IEE East Midland Centre Lincolnshire, UK.

Pont, M. J. (2000). "Designing and implementing reliable embedded systems using patterns". Proceedings of the 4th European Conference on Pattern Languages of Programming and Computing.

Pont, M. J. (2001). "Patterns for time-triggered embedded systems", Addison Wesley.

Pont, M. J., Y. Li, C. Parikh and C. P. Wong (1998). "The design of embedded systems using software patterns". Proceedings of Condition Monitoring, Swansea, UK.

Rising, L., Ed. (2001). "Design Patterns in Communications Software". New York, USA, Oxford University Press.

Appendix-E Comparison of effort and source code changes

This appendix presents the individual results of the groups in Case Study 8 to illustrate the trend between the effort involved and the number of changes made to the source code.

In Chapter 8, the results suggest that there was a similar trend between the effort involved and the number of source code changes made by each group. This section presents the individual results of each group that illustrates the effort spent and the amount of changes made to the source code for the various development phases.

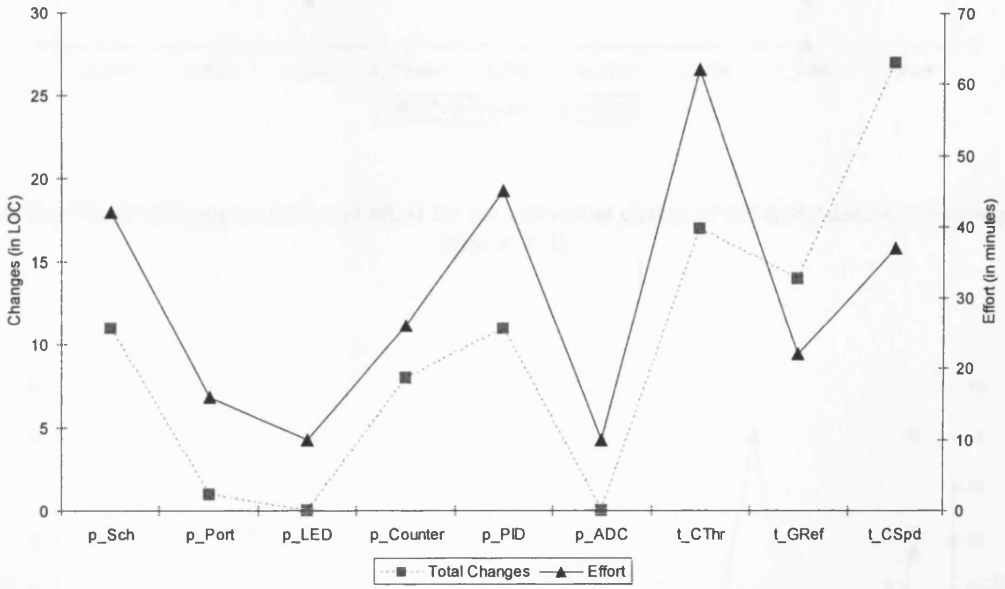

Figure E-1 Trend of changes made and effort for the individual phases of the study described in Chapter 8 (Group A-I).

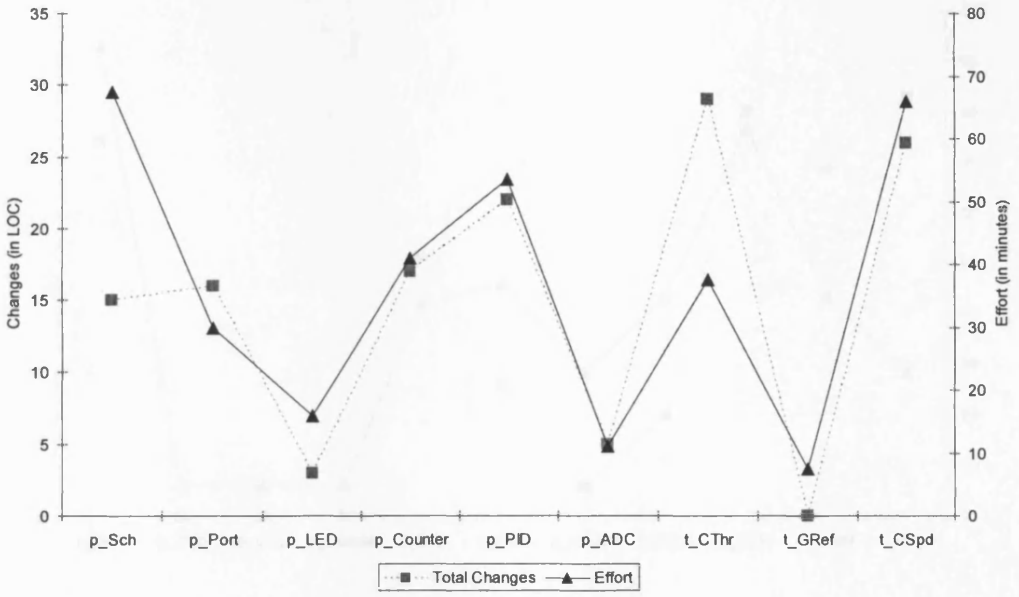

Figure E-2 Trend of changes made and effort for the individual phases of the study described in Chapter 8 (Group B-I).

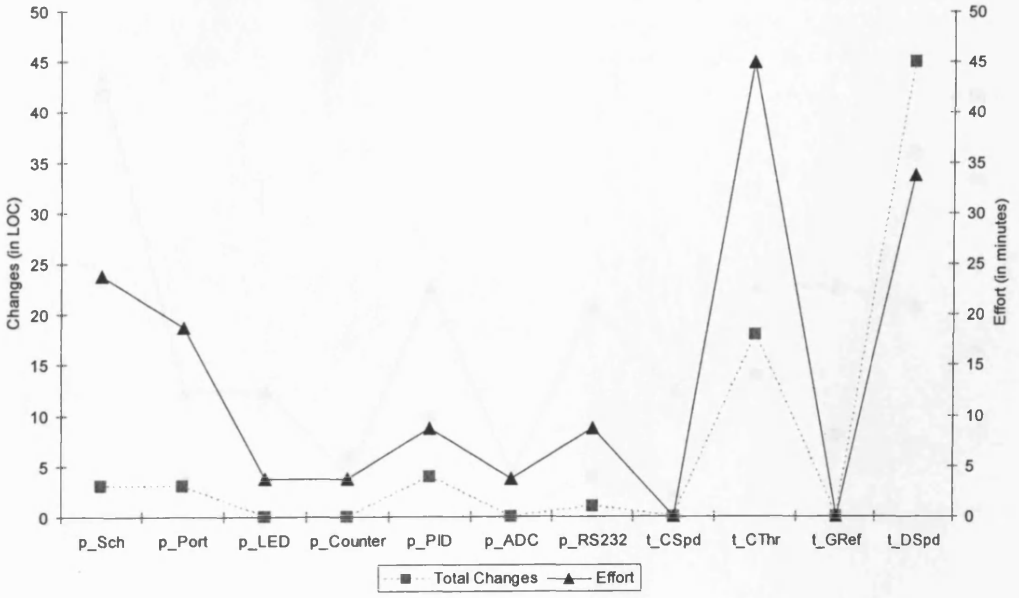

Figure E-3 Trend of changes made and effort for the individual phases of the study described in Chapter 8 (Group A-II).

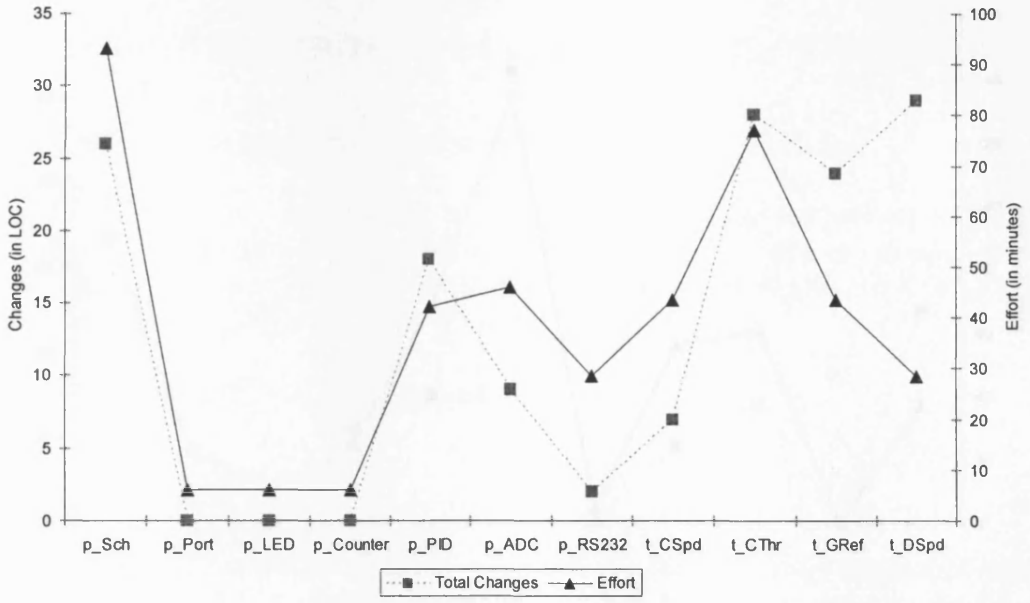

Figure E-4 Trend of changes made and effort for the individual phases of the study described in Chapter 8 (Group B-II).

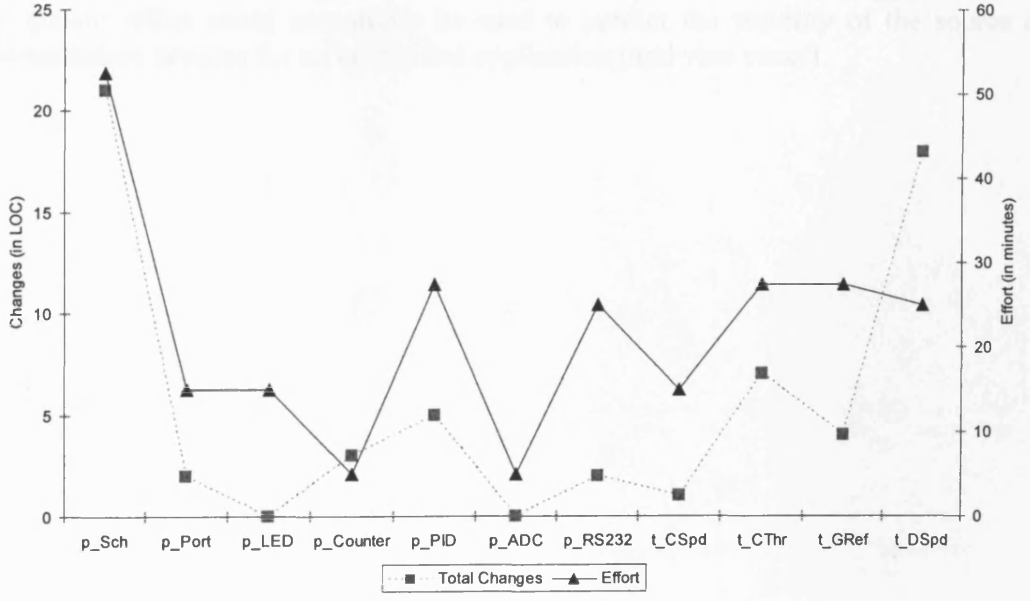

Figure E-5 Trend of changes made and effort for the individual phases of the study described in Chapter 8 (Group C-II).

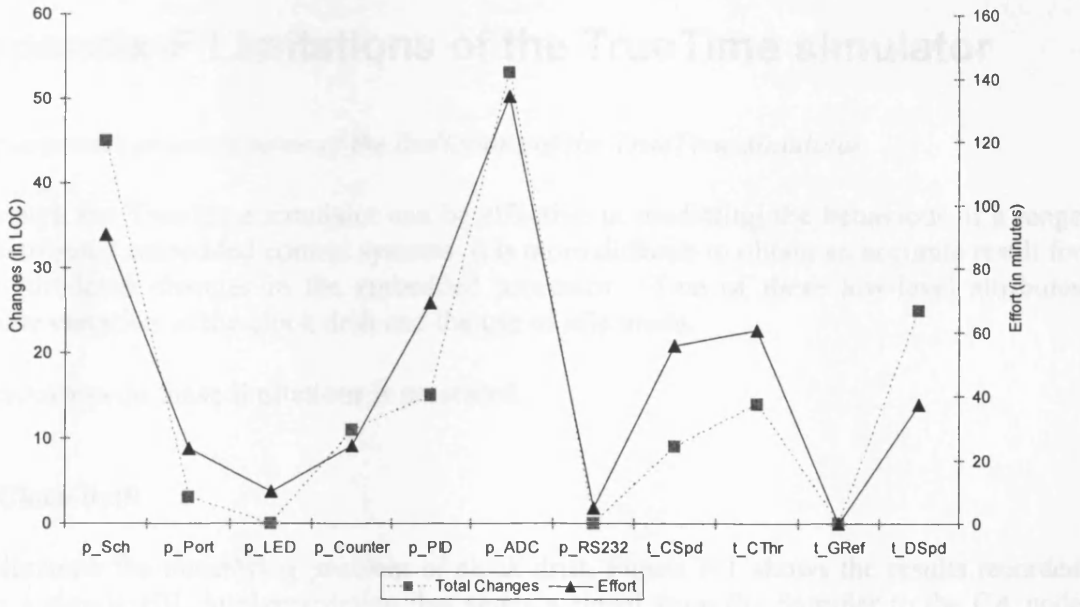

Figure E-6 Trend of changes made and effort for the individual phases of the study described in Chapter 8 (Group D-II).

Based on these results (Figure E-1 to Figure E-6), it could be suggested that the development effort could potentially be used to predict the stability of the source code implementation process for an embedded application (and vice versa).

Appendix-F Limitations of the TrueTime simulator

This appendix presents some of the limitations of the TrueTime simulator.

Although the TrueTime simulator can be effective in predicting the behaviour of a range of distributed embedded control systems, it is more difficult to obtain an accurate result for very low-level changes in the embedded processor. Two of these low-level attributes include variation in the clock drift and the use of idle mode.

A discussion on these limitations is presented.

1. Clock drift

To illustrate the underlying problem of clock drift, Figure F-1 shows the results recorded from a simple HIL implementation that sends a signal from the Sampler to the CA node across the CAN bus periodically. On the nodes, the sampling and actuation tasks were (both) scheduled every 5ms, individually. In this case, the communication delay was approximately 390μs: this represented the minimum possible response time. The maximum response time was 390μs plus the 5ms (actuation) task period. The recorded results lie in this range of values, as illustrated in Figure F-1.

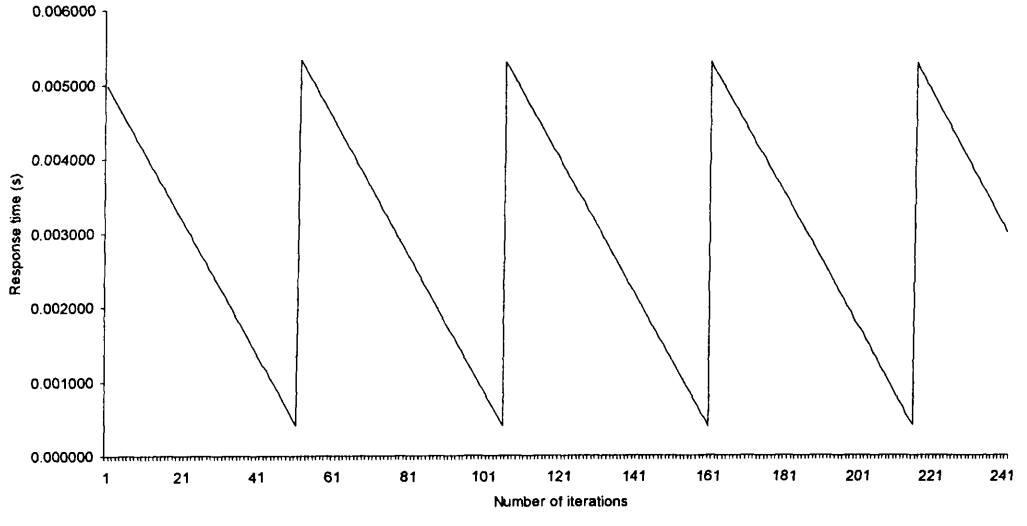

Figure F-1 Drift in the signal response time for "T-E-T" system using the HIL testbed. Figure adapted from Ayavoo *et al.* (2004).

To investigate this further, a simple test of a T-E-T system was employed, where a message was transmitted periodically from one node to the other, and the response time was measured. In addition, the frequencies of the individual crystal oscillators on the microcontrollers were also measured. These values were then used to calculate and modify the individual periods of the task on each of the simulated nodes such that the new task periods on the simulator were matched up with the actual crystal frequency.

Figure F-2 illustrates the comparative response time between the TrueTime simulation and the HIL results when clock drift is not considered in the TrueTime simulation. The result shows that the response time measured on the simulator was constant due to the assumption that both the nodes were operating at the same crystal frequency. Figure F-3 illustrates the result when the individual clock drifts of the crystals were taken into account in the TrueTime simulation process. The result from the TrueTime simulator now shows a closer correspondence to the characteristics of the hardware implementation.

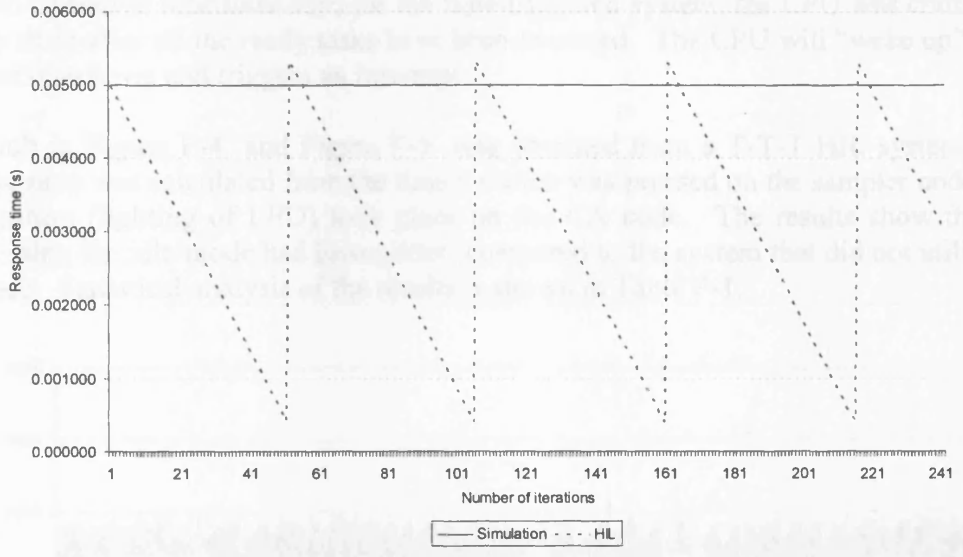

Figure F-2 Response time of the simulation and HIL results when clock drift is not taken into account on the TrueTime simulator.

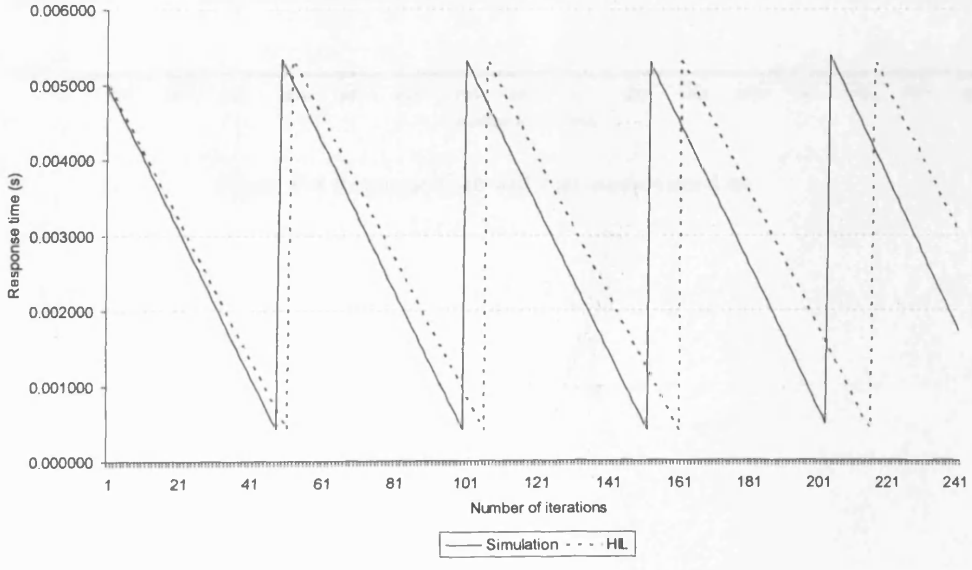

Figure F-3 Response time of the simulation and HIL results when clock drift is incorporated in the TrueTime simulation.

2. Idle mode

The idle mode is a state at which the microcontroller "goes to sleep". The benefits of using idle mode include a reduction in power consumption and CPU utilisation of the microcontroller. The microcontroller will come out of the idle mode or "wake up" when an interrupt occurs.

In the co-operative scheduler used for the time-triggered system, the CPU was configured to go to sleep after all the ready tasks have been executed. The CPU will "wake up" when the timer overflows and triggers an interrupt.

The result in Figure F-4 and Figure F-5 was obtained from a T-T-T HIL system. The response time was calculated from the time a switch was pressed on the sampler node until the actuation (lighting of LED) took place on the CA node. The results show that the system using the idle mode had lower jitter, compared to the system that did not utilise the idle mode. Statistical analysis of the results is shown in Table F-1.

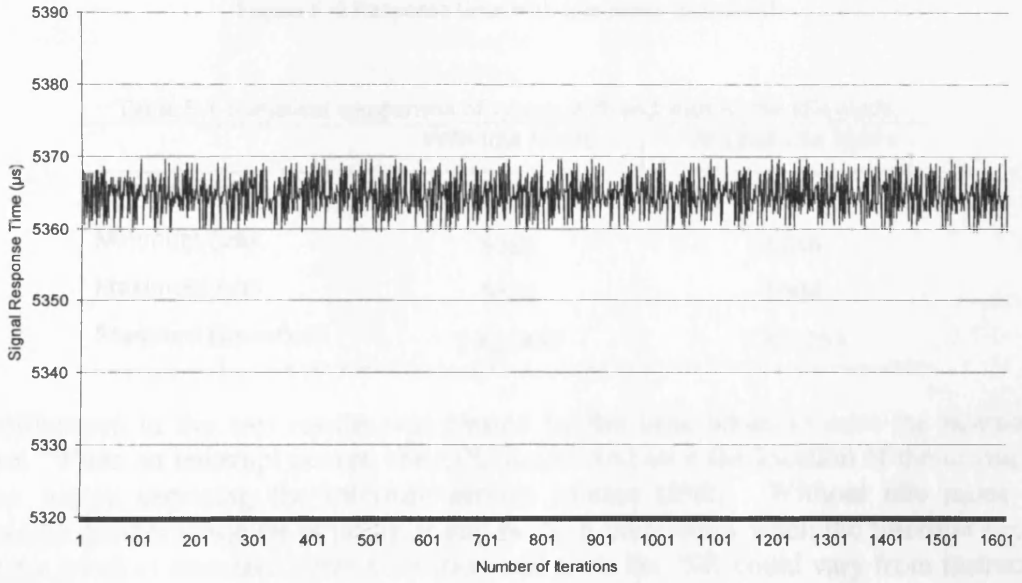

Figure F-4 Response time with idle mode turned on.

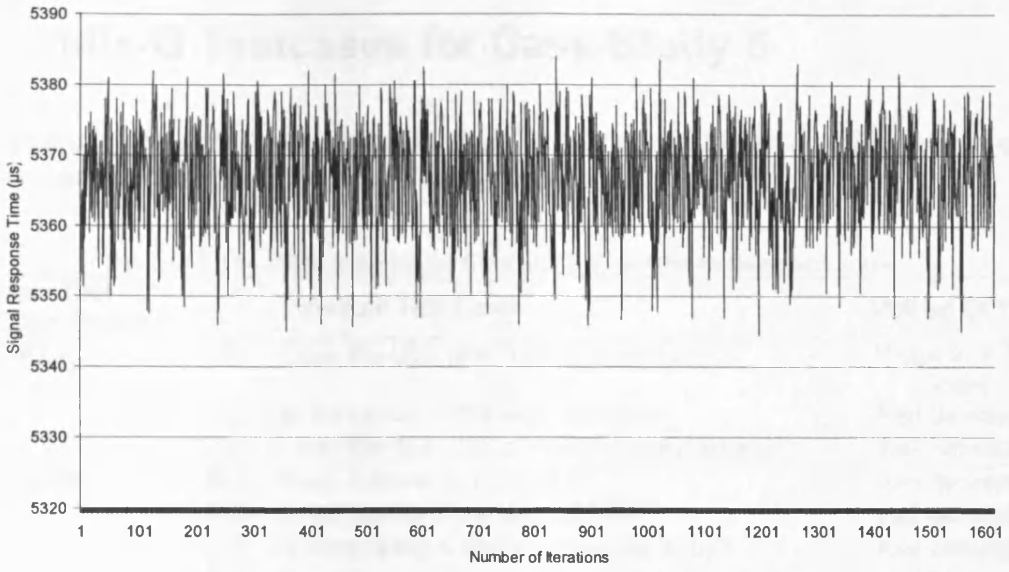

Figure F-5 Response time with idle mode turned off.

Table F-1 Statistical comparison of system with and without the idle mode.

	With Idle Mode	Without Idle Mode
Average (µs)	5365	5367
Minimum (µs)	5360	5345
Maximum (µs)	5370	5384
Standard Deviation	2.422447	7.831764

The difference in the two results was caused by the time taken to save the instruction pointer. When an interrupt occurs, the CPU has to first save the location of the instruction pointer before servicing the interrupt service routine (ISR). Without idle mode, the instruction pointer could be pointing at any random instruction when the interrupt occurs. The time taken to save that current location and go to the ISR could vary from instruction to instruction. By using idle mode, the processor will be at a known state when an interrupt occurs. Therefore, the time taken to save the present location into the stack before servicing the ISR is always constant. This reduces the jitter in the system.

However, this low-level characteristic of an embedded microcontroller is much more difficult to simulate using TrueTime.

Appendix-G Testcases for Case Study 8

This appendix presents the testcases used in Case Study 8 (Phase-II). The results of the testcase evaluation for each group are also presented.

Table G-1 The list of testcases for Case Study 8, and the method used to test.

	Main Test Case Groups		Individual Test Cases	Method Of Test
A	LED	A-1	Does the LED (P4.7) blink on and off?	Visually on the board
		A-2	Is the period of the task 1000ms?	Keil Simulator
		A-3	Does the task start correctly the first time?	Keil Simulator
B	Counter	B-1	Does it count up correctly?	Keil Simulator
		B-2	Is the period of the task 50 ms?	Keil Simulator
		B-3	Is the scaling & filtering done correctly?	Keil Simulator
		B-4	Does the task start correctly the first time?	Keil Simulator
C	Control & Actuation	C-1	Is the PID algorithm called every 50ms?	Keil Simulator
		C-2	Is this task called after the sensor task?	Keil Simulator
		C-3	Is the scaling done correctly?	Keil Simulator
		C-4	Is the value displayed on P5?	Keil Simulator
		C-5	Does the task start correctly the first time?	Keil Simulator
D	ADC	D-1	Is the ADC task called every 1000 ms?	Keil Simulator
		D-2	Is data from the ADC scaled correctly?	Keil Simulator
		D-3	Can the ADC work for varying values of the potentiometer (0,1,2,3,4,5?)	Keil Simulator
		D-4	Does the task start correctly the first time?	Keil Simulator
E	PC_LINK_O	E-1	Is this task called every 10ms?	Keil Simulator
		E-2	Is a value displayed on the screen?	Serial init on the screen
		E-3	Does the task start correctly the first time?	Keil Simulator
F	RS232_Display	F-1	Is this task called every 1000ms?	Keil Simulator
		F-2	Does it display the correct value on the terminal?	View on screen
		F-3	Is the value displayed in MPH?	Keil Simulator
		F-4	Does the task start correctly the first time?	Keil Simulator
G	Overall test	G-1	Does the car speed stabilise at a particular value?	HIL test
		G-2	Is the set speed of the car displayed on the screen?	HIL test
		G-3	Does the set speed and car speed match up?	HIL test

Table G-2 The record of testcase evaluation for Group A-II.

	1	2	3	4	5
A-1		√	√	√	√
A-2		√	√	√	√
A-3		√	√	√	√
B-1		√	√	√	√
B-2		√	√	√	√
B-3					√
B-4		√	√	√	√
C-1					√
C-2					√
C-3					√
C-4					√
C-5					√
D-1		√	√	√	√
D-2					√
D-3		√	√	√	√
D-4		√	√	√	√
E-1		√	√	√	√
E-2		√	√	√	√
E-3		√	√	√	√
F-1		√	√	√	√
F-2					√
F-3		√	√	√	√
F-4		√	√	√	√
G-1					√
G-2					√
G-3					√
	0	15	15	15	26

Table G-3 The record of testcase evaluation for Group B-II.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
A-1		√	√	√	√	√	√	√	√	√	√	√	√	√
A-2												√	√	√
A-3		√	√	√	√	√	√	√	√	√	√	√	√	√
B-1		√	√	√	√	√	√	√	√	√	√	√	√	√
B-2												√	√	√
B-3						√	√	√	√	√	√	√	√	√
B-4												√	√	√
C-1												√	√	√
C-2		√	√	√	√	√	√	√	√	√	√	√	√	√
C-3		√	√	√	√	√	√	√	√	√	√	√	√	√
C-4		√	√	√	√	√	√	√	√	√	√	√	√	√
C-5												√	√	√
D-1												√	√	√
D-2							√	√	√	√	√		√	√
D-3		√	√	√	√	√			√	√	√	√	√	√
D-4												√	√	√
E-1													√	√
E-2													√	√
E-3													√	√
F-1													√	√
F-2														√
F-3													√	√
F-4													√	√
G-1								√	√			√	√	√
G-2														√
G-3														√
	0	7	7	7	7	8	8	9	10	9	9	16	23	26

Table G-4 The record of testcase evaluation for Group C-II.

	1	2	3	4	5	6	7	8
A-1				√	√	√	√	√
A-2				√	√	√	√	√
A-3				√	√	√	√	√
B-1					√	√	√	√
B-2					√	√	√	√
B-3					√	√	√	√
B-4					√	√	√	√
C-1					√	√	√	√
C-2					√	√	√	√
C-3					√	√	√	√
C-4					√	√	√	√
C-5					√	√	√	√
D-1					√	√	√	√
D-2					√	√	√	√
D-3					√	√	√	√
D-4					√	√	√	√
E-1								√
E-2							√	√
E-3							√	√
F-1								√
F-2								√
F-3								√
F-4							√	√
G-1					√	√	√	√
G-2								√
G-3						√	√	√
	0	0	0	3	17	18	21	26

Table G-5 The record of testcase evaluation for Group D-II.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
A-1	√			√	√	√	√	√	√			√	√	√	√	√	√
A-2	√	√	√	√	√	√	√	√	√			√	√	√	√	√	√
A-3	√	√	√	√	√	√	√	√	√			√	√	√	√	√	√
B-1				√	√	√	√	√				√	√	√	√	√	√
B-2				√	√		√	√				√	√	√	√	√	√
B-3				√	√	√	√	√				√	√	√	√	√	√
B-4				√	√		√	√				√	√	√	√	√	√
C-1										√	√	√	√	√	√	√	√
C-2				√	√	√	√	√				√	√	√	√	√	√
C-3				√	√	√						√	√	√	√	√	√
C-4				√	√	√	√	√				√	√	√	√	√	√
C-5				√	√		√	√		√	√	√	√	√	√	√	√
D-1												√	√	√	√	√	√
D-2																	
D-3																	√
D-4							√	√				√	√	√	√	√	√
E-1												√	√	√	√	√	√
E-2							√	√				√	√	√	√	√	√
E-3							√	√				√	√	√	√	√	√
F-1	√											√	√	√	√	√	√
F-2															√	√	√
F-3	√						√	√				√	√	√	√	√	√
F-4	√						√	√				√	√	√	√	√	√
G-1															√	√	
G-2															√	√	
G-3															√	√	
	6	2	2	11	11	8	15	15	3	2	2	20	20	20	24	24	22

Appendix-H Examples of some analytical models

This appendix presents some examples of analytical models for real-time embedded systems. This material is referred to in Chapter 2.

1. Detailed CAN communication model

This section illustrates some examples of analytical models for the CAN communication. The worst-case response time for the CAN communication is composed of two delays, the physical transmission delay (C_m) and the queing delay (W_m).

C_m is defined as the worst-case time taken to physically transmit the CAN message m. C_m may be calculated using Equation 1. The term S_m in this equation refers to the number of data bytes transmitted for a single CAN message. This variable can take any value from one to eight. τ_{bit} is the time resolution: for example, if the CAN bus is running at a speed of 1 Mbps, then τ_{bit} is 1μs.

$$C_m = \left(\left\lfloor \frac{34 + 8S_m}{4} \right\rfloor + 47 + 8S_m \right) \tau_{bit} \qquad (1)$$

Note that Equation 1 is for a standard CAN message. If an extended CAN message is used, then the equation can be easily modified, to give Equation 2:

$$C_m = \left(\left\lfloor \frac{54 + 8S_m}{4} \right\rfloor + 67 + 8S_m \right) \tau_{bit} \qquad (2)$$

For an extended CAN identifier, there are 67 bits of message overhead compared to 47 bits in a standard message (this doesn't include any data). This difference arises because of the different lengths of the Arbitration Field in the two versions of the protocol.

For a standard CAN message, 34 bits in the message overhead is subjected to bit stuffing. One bit is stuffed for every four bits: therefore, a maximum of four similar consecutive bits can be encountered before a stuffed bit is inserted (Nolte *et al.*, 2001). Early work assumed that the worst-case scenario would involve insertion of a stuffed bit for every five bits; however, the inserted stuff bit contributes to the first bit of the next sequence (Ellims *et al.*, 2002). Therefore, the sequence 00000111100001111 becomes 00000(1)1111(0)0000(1)1111 where the bits in ( ) are the stuffed bits. The data field of the CAN frame is also subject to bit stuffing, where one bit is stuffed for every four similar consecutive bits. Therefore, C_m takes into account the message overhead, the data transmitted and the number of bits stuffed.

The queuing delay is denoted as W_m. W_m may be calculated using Equation 3:

$$W_m^{n+1} = B_m + \sum_{\forall j \in hp(m)} \left\lceil \frac{w_m^n + J_j + \tau_{bit}}{T_j} \right\rceil C_j \qquad (3)$$

(Note that Equation 3 will be calculated recursively until W_m^{n+1} converges to W_m^n.)

In Equation 3, B_m is the blocking delay. This delay is caused by a lower priority message that is already transmitting its message over the CAN bus when message m enters the queue. B_m may be calculated using Equation 4:

$$B_m = \max_{\forall k \in lp(m)} (c_k) \qquad (4)$$

(Note: Equation 1 and Equation 2 are from Ellims *at al.* (2000). Equation 3 and Equation 4 are from the work of Tindell and Burns (1994).

2. Analytical models for a two node distributed system

Analytical models have also been created for distributed systems. The models take into account the various scheduling technique and message communication. Table H-1 below illustrates the analytical models created by Lonn and Axelsson (1999) for the minimum and maximum control delays for a two-node distributed network.

Table H-1 Control delays for different scheduling policy. Table adapted from Lonn and Axelsson (1999).

Communication Scheduling	Processor Scheduling	Global Time	Control Delay	
Fixed-Priority	Fixed-Priority	No	Min	$C_M + O_{MA} + C_A$
			Max	$B_M + C_M + I_M + O_{MA} + B_A + C_A + I_A$
		Yes	Min	$O_A + C_A - (B_S + I_S + C_S)$
			Max	$O_A + B_A + I_A + C_A - C_S$
Static-Cyclic	Fixed-Priority	No	Min	$C_M + O_{MA} + C_A$
			Max	$T_{TDMA} + C_M + O_{MA} + B_A + C_A + I_A$
		Yes	Min	$O_A + C_A - (B_S + I_S + C_S)$
			Max	$O_A + B_A + I_A + C_A - C_S$
Static-Cyclic	Static-Cyclic	No	Min	$C_M + O_{MA} + C_A$
			Max	$T_{TDMA} + C_M + T + O_{MA} + C_A$
		Yes	Min	$O_A + C_A$
			Max	$O_A + C_A$
Fixed-Priority	Static-Cyclic	No	Min	$C_M + O_{MA} + C_A$
			Max	$B_M + I_M + C_M + T + O_{MA} + C_A$
		Yes	Min	$O_A + C_A$
			Max	$O_A + C_A$

References

Ellims, M., S. Parker and J. Zurlo (2002). "Design and analysis of a robust real-time engine control network." IEEE Micro 22(4): 20-27.

Lonn, H. and J. Axelsson (1999). "A comparison of fixed-priority and static cyclic scheduling for distributed automotive control application". The Eleventh Euromicro Conference on Real-Time Systems, York, England.

Nolte, T., H. Hansson, C. Norström and S. Punnekkat (2001). "Using bit-stuffing distributions in CAN analysis". IEEE/IEE Real-Time Embedded Systems Workshop (RTES'01), London, UK.

Tindell, K. and A. Burns (1994). "Guaranteed message latencies for distributed safety-critical hard real-time control networks". YCS 229 Real-Time Systems Research Group, University of York, York, England.

Appendix-I Examples of student questionnaires

This appendix presents some example questionnaires used by the student groups in the SGM.

1. Questionnaire for non-simulation groups

i) What were the areas that were thought to be difficult and / or time consuming when implementing the system?

ii) What were the areas that were thought to be easy when implementing the system?

iii) Do you think that having a simulator available (for use before code implementation) would have helped to make it easier to implement your system?

iv) Assuming that you feel a simulator could be useful, what features do you think you would require in such a program?

v) While developing the system, did you fully understand the software architecture of the scheduler that you used? (Please explain your answer)

vi) How else do you think you could have created the code for this system? What were the drawbacks/failures of the method you used?

vii) Did you find the software architecture that you used posed a problem at any point in your development? (ie – would you have preferred to have used a co-operative instead of a pre-emptive scheduler and vice versa?) Please explain your answer.

viii) Any other comments?

2. Questionnaire for simulation groups

i) What were the areas that were thought to be difficult and / or time consuming when simulating the system on the TrueTime simulator?

ii) What more could have been done to make the TrueTime software simulation process much easier and user friendly?

iii) What were the areas that were thought to be easy when simulating the system on TrueTime?

iv) Were there any particular features of the TrueTime simulator that you thought was particularly useful?

v) What were the areas that were thought to be difficult and / or time consuming when implementing the system on hardware?

vi) For such areas, do you think the TrueTime software simulator could have been modified to make it easier to implement the system on the hardware? Please give some examples/suggestions etc…

vii) What were the areas that were thought to be easy when implementing the system on hardware?

viii) Do you think that having TrueTime simulator available (for use before code implementation) has helped to make it easier to implement your system?

ix) While developing the system, did you fully understand the software architecture of the scheduler that you used? (Please explain your answer)

x) How else do you think you could have created the code for this system? What were the drawbacks/failures of the method you used?

xi) Did you find the software architecture that you used posed a problem any point in your development? (ie – would you have preferred to have used a co-operative instead of a pre-emptive scheduler and vice versa?) Please explain your answer.

xii) Any other comments?